K-Means and K Nearest Neighbor Methods on Optical Region Recognition of Image Data: Comparison to Model Based Methods

Boemseok Seo, Sina Sanei
Statistics 557: Project-2
Spring 2018

March 17, 2018

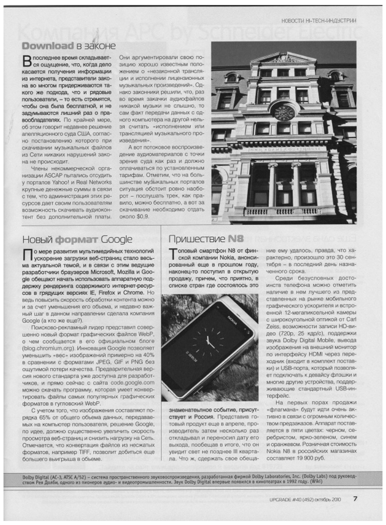
1 Introduction

1.1 Motivation and goal

In the previous work, "Linear Classification Methods on Optical Region Recognition of Image Data(Seo and Sanei)", we addressed optical region recognition techniques on paper magazine image data, and discussed about model based methods - LDA, QDA, and Logistic Regression. Model based methods are easy to interpret and have advantages of providing decision boundaries related to considered attributes. However, since model based methods rely on assumptions of models, one of which commonly included is Gaussian distribution, it sometimes fails to obtain proper results. So now we introduce algorithm based classification methods, K-means and KNN, on our image data and compare the results to previous work.

With the recent advances in machine learning techniques, the digital image processing can be done on the more sophisticated image mining fields. One example is optical character recognition(OCR) which is detecting text from image. As a myriad of historical records exist in a form of images, transforming them to machine readable digital text is getting more attention. The matter of this problem is the speed of transforming image to text. We introduced classification methods to accelerate the work by separating text region from picture and background region, which is optical region recognition(ORR). ORR is expected to reduce the size of data by limiting the number of pixels that OCR machine takes into consideration. According to Gonzales & Wintz[1], an image can be defined as a two-dimensional function, $f(x, y)$, where the value of f at any pair of coordinates, (x, y), is called the *intensity* or *gray level* of the image at that point. The digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as *picture elements*, *image elements* , or *pixels*.

Objective: Objective of this project is to use algorithm based classification methods to train and validate them in classifying regions of documents to **text**, **picture** and **background areas**. So one challenge is finding accurate classifier using the gray value of each pixel, in order to classify each pixel into three area. To address this issue, we apply cross validation technique and investigate additional methods to increase accuracy, such as dimension reduction.


(a) Original Image

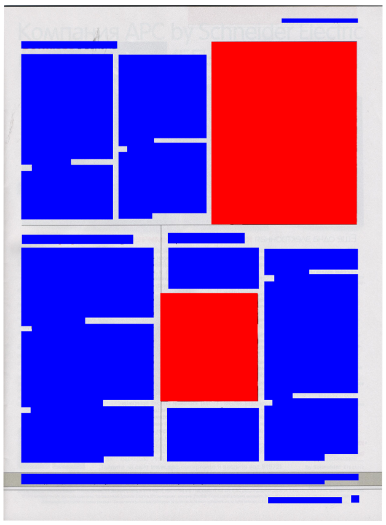

(b) Pixel Based Mask

Figure 1: a: original image (Input) b: Pixel Based Mask(Response)

1.2 Data description

The data was obtained from The UCI Machine Learning Repository. It is called " Newspaper and magazine images segmentation dataset" [2] (Creators: Aleksey Vilkin and Ilia Safonov, NRNU MEPhI, Moscow, Russia, Date: 2012) This dataset was collected for training and validation of machine learning algorithm for classification regions of documents on text, picture and background areas. It contains 101 scanned images of various newspapers and magazines in Russian. Most of the images have resolution 300 dpi and size A4, about 2400x3500 pixels. For all images ground truth pixel-based masks were manually created. There are three classes: text area, picture area, background. Pixels on the mask with color 255, 0, 0 (rgb, red color) correspond to picture area, pixels with color 0, 0, 255 (rgb, blue color) correspond to text area, all other pixels correspond to background. Images with background of different colors are in the dataset. An example image and corresponding pixel-based masks is presented in figure1. The image shown in *figure 1* is used as test data in all methods used for classifications in this report.

2 Preprocessing and Variable selection

2.1 Data Shrinkage

Original image has $3,500 \times 2,400 =$ around 8 million pixels which have gray scale values between [0,1] for each image, and each marked ground truth has same size of data. It means that only with 10 images, we need to manage 160 million values from original data. It is computationally inefficient. Reading 160 million values in program R takes time but distinguishing text and photos does not need to be done on each pixel. In this reason, data shrinkage is applied prior to analysis.

Shrinkage can be done in various way but one of the simplest way is choosing a pixel for small regions that the original image is divided into. For example, as like Figure 2, we can choose the first upper left pixel for $k \times k$ pixel matrix.

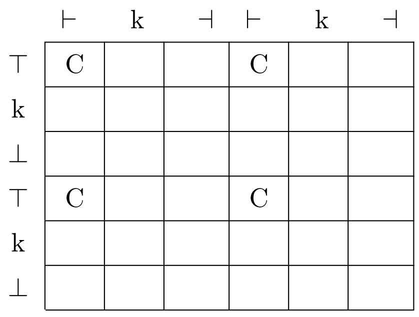
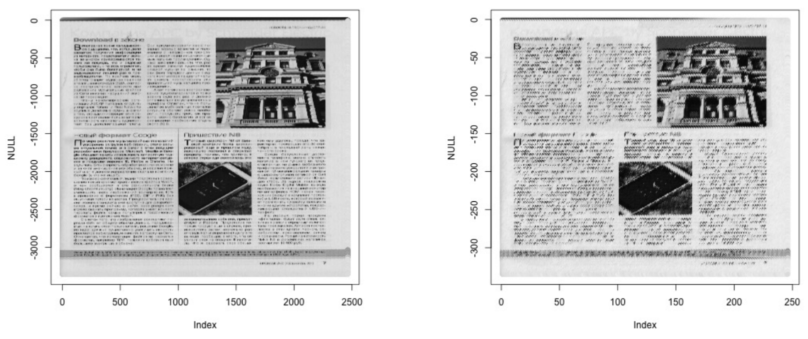


Figure 2: The way to shrink image data. Choose only C marked pixel and remove other pixels for each $k \times k$ pixel matrix.

We considered $k = 10$ on an ad hoc basis. There is a trade off for choosing k here. If k is too big then data become light but it may difficult to recognize the important feature in the image, whereas if k is too small then more feature in the image remain in the data but we can not achieve shrinkage of data. For our data set, by $k = 10$, we can reduce data size by $\frac{1}{100}$ from 8 million values for each photo to $350 \times 240 =$ around 80 thousand values. The result of shrinkage is shown in Figure 3.



(a) Before shrinking (b) After shrinking

Figure 3: Comparison of original data and shrunk data

2.2 Variable Selection

Opposed to usual classification situation in which researchers need to choose the best set of feature variables among tens of relevant variables, for image data there is basically one variable which is the value of each pixel. For our data set, we have only one explanatory variable that is gray scaled value which is between [0,1], and one response variable that is categorical variable {0:Background, 1:Picture, 2:Text}. However, the gray value itself does not have much information about if the pixel is text or picture. It is because there exist pixels in picture region that have the same gray value as the value of pixels in text and background. As we will see in Section 3.1, any algorithm does not work well with only gray value of each pixel.

So we need to use spatial information. There are various ways to extract spatial information from image data. But one simple way to do is using neighbor information. We considered $\pm r$ pixels' information to figure out category of each pixel. That is in 2 dimensional image, for pixel $P_{(i,j)}$, we check the neighbor $P(i-r, j-r), \cdots, P_{i+r,j+r}$. The Figure 4 shows the neighbor.

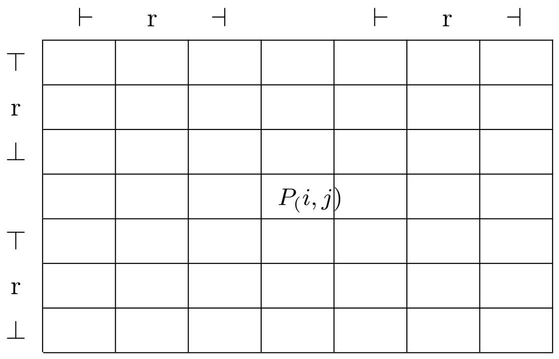


Figure 4: $r \times r$ neighbor of each pixel

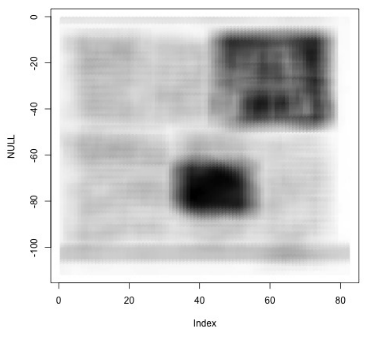
For $r \times r$ neighbor, we calculated average and variance of pixel gray values. The result of newly calculated variables with $r = 10$ for our data is shown in Figure 10. Here r is chosen by an ad hoc basis considering the average size of text in image.

- Average of neighbor pixels :

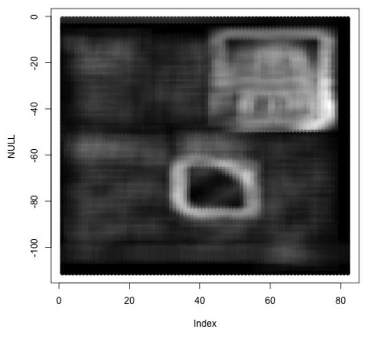
$$X'_{(i,j)} = \frac{1}{(r+1)^2} \sum_{k=-r}^{r} \sum_{l=-r}^{r} X_{(i+k,j+l)}$$

- Variance of neighbor pixels :

$$X''_{(i,j)} = \frac{1}{(r+1)^2} \sum_{k=-r}^{r} \sum_{l=-r}^{r} (X_{(i+k,j+l)} - X'_{(i,j)})^2$$



(a) Average of neighbor(r=10)



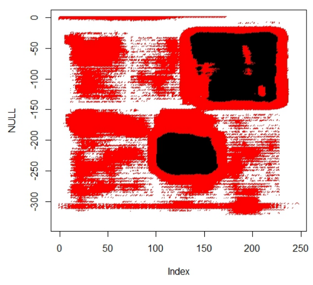
(b) Variance of neighbor(r=10)

Figure 5: Newly calculated variables with $r = 10$ visualization.

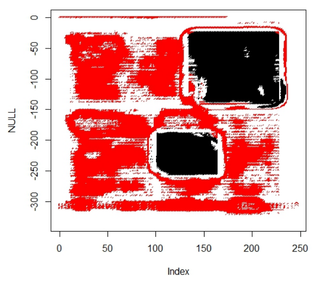
3 Summary of the Previous Work

After preparing the data for analysis, LDA, QDA and logistic regression analysis are applied. Since response variable, the pixel based mask, has three levels, we coded three binary response variables corresponding to each level. In each of these response variables, 1 represents the presence of that response level and 0 for other levels. We then fitted three regressions of responses on our training data set using the variables described in Section 2 as explanatory variables, and determine the final predicted value based on softmax function.

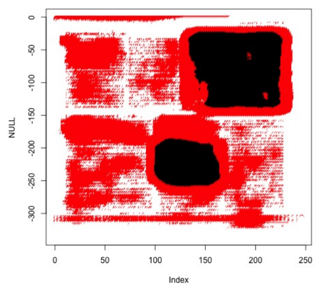
Using the fitted model, we predict the test data set. The results of three methods are presented in Figure 6. And the error rate for each model is in Table **??**.



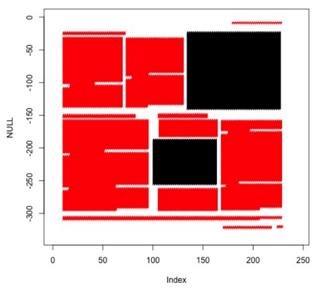
(a) LDA



(b) QDA



(c) Logistic



(d) Truth

Figure 6: Comparison between predicted class from model based classification methods and truth

Classification Methods	Error Rate
LDA	27.20%
QDA	29.04%
Logistic Regression	22.67%

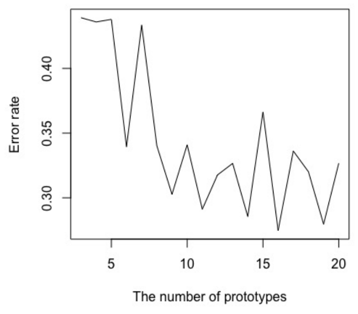
Table 1: Error rates on the test sets for model based classification.

In the expectation of increasing accuracy rates, we considered demension reduction technique - principal component analysis(PCA), but since we have only 3 variables, it did not improve the result at all.

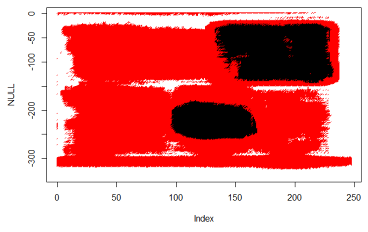
4 Algorithm Based Classification

4.1 K-means as Classification

Now we apply K-means method to our data set. In order to find the best K, the number of prototypes, we use cross validation technique. We consider the number of prototypes from 3 to 20. Figure 7 shows that the error rate of cross validation is minimized when the number of prototype is 16. Hence we apply K-means algorithm with K=16 on our training data set and then classify each 16 clusters to one of three class - text, picture, and background. (Scheme 2 in lecture note.) The predicted image on our test data set is in Figure 7 and its error rate is 20.03%.



(a) Error rate for the number of prototypes



(b) Predicted image

Figure 7: (Left) cross validation error rate on training data set and (right) predicted image on test set.

4.2 K-means as Clustering

Since K-means algorithm is more popular for unsupervised learning, we also attempt to get clustering result by K-means. Three clusters are considered directly and the result is in Figure 9 and the error rate is 46.54%, which is much higher than classification result but it is not surprising because unsupervised leaning use less information compared to supervised learning.

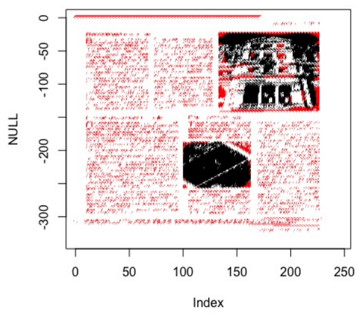


Figure 8: K-means clustering with three prototypes.

4.3 K Nearest Neighbor

Fot fitting KNN, we used the package *cluster* in R . With using our training data , and test data , we fit the model for different tunning parameter which is the number of neighbors, we obtained the error values for test data, as presented in table 3. As we can see, the error rates keep decreasing with increasing the number

number of neighbors											
	1	3	5	10	15	17	20	25	35	50	100
Error rate	0.22	0.194	0.184	0.174	0.169	0.168	0.168	0.166	0.165	0.164	0.163

Table 2: Error rate of KNN for different tuning parameter

of neighbors increase. We stopped the calculations at 100 , since the rate of error decreasing was small after k=25, and the computations were time consuming. We take the k=100 to be our best model in the sense the it yields the lowest error rate of 16.3%. The corresponding prediction from this model is presented in Figure 9.

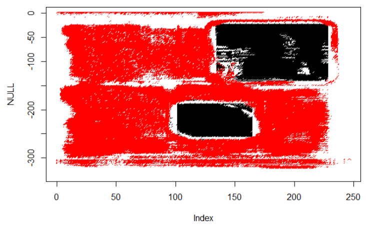
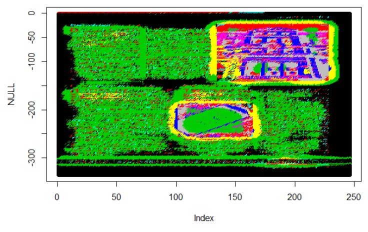


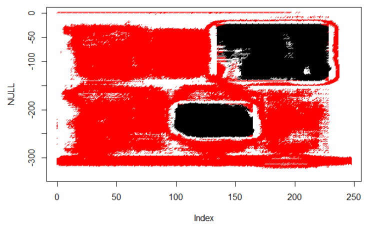
Figure 9: KNN clustering with k=100.

4.4 K Means Using K-center as Initialization

K means algorithm uses random initial points for prototypes, and for this reason may not achieve the optimal result, and our results from K-means were not promising, so we try to improve the result by using the K-center clustering as initial points for K-means. We made the greedy algorithm to choose up to 11 points and made these points the initialization for k-means. Since the error reduction from our k-means analysis was not significant after k=11. Because of computational constraints,and since k-venter is an unsupervised learning algorithm, We ran the algorithm only on our test data and after assigning the prototypes to class with largest number of members from response, the result was improves as expected , and the error rate was 18.8%. The result is shown in figure 10.



(a) Unsupervised k-means



(b) After assigning prototypes

Figure 10: K-means using K-center as initialization with k=11

5 Discussion

In this analysis we tried to obtain the best model for our classification problem. The results from k-nearest neighbors with k=100 and k-mean with k-center as initialization were our best models, and performed better than our previous best model which was the QDA from cross validation and logistic regression both with error rates of approximately 22%. As it can be seen from the figures, the picture areas are better detected with these two models comparing to k-means with random initializations.

We performed the PCA on the data but did not include it in report because the results were not better than the k-means using original data which is not surprising as our feature space does not have a dimensionality problem.

6 Appendix 1: R code

```
rm(list=ls())
#install.packages("png")
#install.packages("jpeg")

library(png)
library(jpeg)

setwd("/Users/BSS/Documents/5. 2018.1 Spring/Stat 557/project1/good")

file_no = c(1,2,3,4,5,14,15,16,20,21) #,23,27,28,29,33,34,35,36,42,55,58,65,67,68,69)
i=1
j=1
for(i in 1:10){
  no = file_no[i]
  x = readJPEG(paste("image (",no,").jpg",sep=""))
  y = readPNG(paste(no,"_m.png",sep=""))

  xxx = x[,,1]
  yy = matrix(0,dim(y)[1]*dim(y)[2])
  yy[which(y[,,1]==1)] = 1
  yy[which(y[,,3]==1)] = 2
  yyy = matrix(yy,dim(y)[1],dim(y)[2])

  nam_x <- paste("mx", i, sep = "")
  nam_y <- paste("my", i, sep = "")
  assign(nam_x, xxx)
  assign(nam_y, yyy)

  nam_x <- paste("vx", i, sep = "")
  nam_y <- paste("vy", i, sep = "")
  assign(nam_x, matrix(xxx,dim(x)[1]*dim(x)[2])) #stack each column
  assign(nam_y, matrix(yyy,dim(y)[1]*dim(y)[2])) #stack each column

  rm(x,y,yy,xxx,yyy)
}
hist(vx1)
hist(vy1)
#save.image(file="image.Rd")
load("image.Rd")

###############################################################
# check the image
plot.img <- function(samp.x, s=10, col=F){
  #s=5  # definition of image
  #samp.x = nmx_2[[1]]                #image x
  samp.bw = (samp.x-min(samp.x))/(max(samp.x)-min(samp.x)) #normalize
  samp.co = samp.x

  ys = floor(dim(samp.x)[1]/s)
  xs = floor(dim(samp.x)[2]/s)
  plot.default(NULL,xlim=c(1,xs),ylim=c(-ys,-1))
  for(i in 1:xs){
    for(j in 1:ys){
      if(col==F) points(i,-j,col=grey(samp.bw[j*s,i*s]),pch=19)  # image x
      if(col==T) points(i,-j,col=samp.co[j*s,i*s],pch=19)  # image x

    }
  }
}

plot.range <- function(x,y){
  plot.default(x=x,y=y,pch=19,cex=0.8,col=y+1,xlab="Gray value of each pixel",ylab="Predicted class"
  legend(0, 3, legend=c("Background","Picture","Text"),
         col=c("black","red","green"), pch=19, cex=0.8,
         horiz=T,bty="n")
}

plot.scatter <- function(x,y){
  xx=x[,2];xy=x[,3]
  plot(xx, xy, pch=19,cex=0.8,col=y+1, xlab="Variance of 21x21 pixel matrix",ylab="Average of 21x21 p
  legend(min(xx),max(xy), 1, legend=c("Background","Picture","Text"),
         col=c("black","red","green"), pch=19, cex=0.8,
         horiz=T,bty="n")
}

##############################################################
# shrink the size

nmx = list()
nmy = list()
vmx = list()
vmy = list()
for(i in 1:10){
  s=10

  mx = eval(parse(text=paste("mx",i,sep="")))
  my = eval(parse(text=paste("my",i,sep="")))

  dimv = dim(mx)[1]
  dimh = dim(mx)[2]
  ndimv = floor(dimv/s)
  ndimh = floor(dimh/s)

  lv = s*(1:ndimv)
  lh = s*(1:ndimh)

  xxx = mx[lv,lh]
  yyy = my[lv,lh]

  nmx[[i]] = xxx
  nmy[[i]] = yyy

  vmx[[i]] = matrix(xxx,ndimv*ndimh)
  vmy[[i]] = matrix(yyy,ndimv*ndimh)
  rm(mx,my,lv,lh,xxx,yyy)
}
nmy
jpeg(file = "notshrunk.jpg")
plot.img(nmy[[4]],s=10)
dev.off()

##############################################################
# create variables

r = 10 #radius

nmx_1 = list()
nmx_2 = list()
nmx_3 = list()

vmx_1 = list()
vmx_2 = list()
vmx_3 = list()
for(i in 1:10){
  mx = nmx[[i]]

  dimv = dim(mx)[1]
  dimh = dim(mx)[2]

  x1 = matrix(NA,dimv,dimh)
  x2 = matrix(NA,dimv,dimh)
  x3 = matrix(NA,dimv,dimh)

  for(v in 1:dimv){
    for(h in 1:dimh){
      minv = max(1,v-r)
      maxv = min(dimv,v+r)
      minh = max(1,h-r)
      maxh = min(dimh,h+r)

      x1[v,h] = var(c(mx[minv:maxv,minh:maxh]))
      x2[v,h] = mean(mx[minv:maxv,minh:maxh])
    }
  }
  x3 = x2-mx

  nmx_1[[i]] = x1
  nmx_2[[i]] = x2
  nmx_3[[i]] = x3

  vmx_1[[i]] = matrix(x1,dimv*dimh)
  vmx_2[[i]] = matrix(x2,dimv*dimh)
  vmx_3[[i]] = matrix(x3,dimv*dimh)
  print(i)
}
#save(nmx_1,nmx_2,nmx_3, file="nmx.Rd")
#save(vmx_1,vmx_2,vmx_3, file="vmx.Rd")
```

```
load("nmx.Rd")
load("vmx.Rd")

plot.default(1,1,col=0)
plot.img(nmy[[4]],s=10)
plot.img(nmy[[4]],s=3,col=T)

##############################################################
# analysis
i=1
vmy_1 = list(); vmy_2 = list(); vmy_3 = list()
for(i in 1:10){
  my = vmy[[i]]
  n = length(my)

  my_1 = rep(0,n)
  my_2 = rep(0,n)
  my_3 = rep(0,n)
  my_1[which(my==0)] = 1
  my_2[which(my==1)] = 1
  my_3[which(my==2)] = 1

  vmy_1[[i]] = my_1
  vmy_2[[i]] = my_2
  vmy_3[[i]] = my_3
  rm(my_1,my_2,my_3)
}
table(my)
#-------------------------------------------------------------
train.x =  NULL
train.x.1 =  NULL
train.x.2 =  NULL
train.x.3 =  NULL
train.y.1 =  NULL
train.y.2 =  NULL
train.y.3 =  NULL
ind = NULL
for(i in c(1,2,3,5,6,7,8,9)){
  train.x=c(train.x,vmx[[i]])
  train.x.1=c(train.x.1,vmx_1[[i]])
  train.x.2=c(train.x.2,vmx_2[[i]])
  train.x.3=c(train.x.3,vmx_3[[i]])
  train.y.1=c(train.y.1,vmy_1[[i]])
  train.y.2=c(train.y.2,vmy_2[[i]])
  train.y.3=c(train.y.3,vmy_3[[i]])
  ind = c(ind,rep(i,length(vmx[[i]])))
}

#-------------------------------------------------------------
# 1-1) Binary Logistic Regression with one covariate
dat1 = data.frame(y = factor(train.y.1), x = train.x)
dat2 = data.frame(y = factor(train.y.2), x = train.x)
dat3 = data.frame(y = factor(train.y.3), x = train.x)
```

```
fitglm_1=glm(y~x, data=dat1, family='binomial')
fitglm_2=glm(y~x, data=dat2, family='binomial')
fitglm_3=glm(y~x, data=dat3, family='binomial')
summary(fitglm_1)

k = 4       # test image
newdat = data.frame(x = vmx[[k]])
yglm_1=predict(fitglm_1, newdata=newdat, type='response')
yglm_2=predict(fitglm_2, newdata=newdat, type='response')
yglm_3=predict(fitglm_3, newdata=newdat, type='response')


yglm = apply(cbind(yglm_1,yglm_2,yglm_3),1,which.max)-1
table(yglm)
table(nmy[[k]])
my1 = matrix(yglm,dim(nmx[[k]])[1],dim(nmx[[k]])[2])
error1 = sum(yglm!=nmy[[k]])/length(nmy[[k]])

jpeg(file="4-1result.jpg")
plot.img(my,s=1,col=T)
dev.off()

plot.range(as.numeric(newdat$x),yglm)
#--------------------------------------------------------------
# 1-2) Binary Logistic Regression with more covariates
dat1 = data.frame(y = factor(train.y.1), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 = train.x.
dat2 = data.frame(y = factor(train.y.2), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 = train.x.
dat3 = data.frame(y = factor(train.y.3), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 = train.x.

fitglm2_1=glm(y~x0+x1+x2, data=dat1, family='binomial')
fitglm2_2=glm(y~x0+x1+x2, data=dat2, family='binomial')
fitglm2_3=glm(y~x0+x1+x2, data=dat3, family='binomial')
summary(fitglm2_3)

newdat = data.frame(x0 = vmx[[k]], x1 = vmx_1[[k]], x2 = vmx_2[[k]], x3 = vmx_3[[k]])
yglm2_1=predict(fitglm2_1, newdata=newdat, type='response')
yglm2_2=predict(fitglm2_2, newdata=newdat, type='response')
yglm2_3=predict(fitglm2_3, newdata=newdat, type='response')

yglm2 = apply(cbind(yglm2_1,yglm2_2,yglm2_3),1,which.max)-1
table(yglm2)
table(nmy[[k]])
my2 = matrix(yglm2,dim(nmx[[k]])[1],dim(nmx[[k]])[2])
error2 = sum(yglm2!=nmy[[k]])/length(nmy[[k]])

jpeg(file="4-2result.jpg")
plot.img(my2,s=1, col=T)
dev.off()
jpeg(file="4-2true.jpg")
plot.img(nmy[[k]], s=1, col=T)
dev.off()

jpeg(file="4-2multcovlog.jpg")
plot.scatter(newdat[,1:2],yglm)
```

```
dev.off()
#--------------------------------------------------------------
# 3) LDA
library(MASS)
fitlda1=lda(factor(y)~x0+x1+x2, data=dat1)
fitlda2=lda(factor(y)~x0+x1+x2, data=dat2)
fitlda3=lda(factor(y)~x0+x1+x2, data=dat3)
summary(fitlda1)

ylda1=predict(fitlda1, newdata=newdat)
ylda2=predict(fitlda2, newdata=newdat)
ylda3=predict(fitlda3, newdata=newdat)

ylda = apply(cbind(ylda1$posterior[,2],ylda2$posterior[,2],ylda3$posterior[,2]),1,which.max)-1
table(ylda)
table(nmy[[k]])
my = matrix(ylda,dim(nmx[[k]])[1],dim(nmx[[k]])[2])
error = sum(ylda!=nmy[[k]])/length(nmy[[k]])

plot.img(my,s=3, col=T)

#--------------------------------------------------------------
# 4) QDA
library(MASS)
fitqda1=qda(factor(y)~x0+x1+x2, data=dat1)
fitqda2=qda(factor(y)~x0+x1+x2, data=dat2)
fitqda3=qda(factor(y)~x0+x1+x2, data=dat3)
summary(fitqda)

yqda1=predict(fitqda1, newdata=newdat)
yqda2=predict(fitqda2, newdata=newdat)
yqda3=predict(fitqda3, newdata=newdat)

yqda = apply(cbind(yqda1$posterior[,2],yqda2$posterior[,2],yqda3$posterior[,2]),1,which.max)-1
table(yqda)
table(nmy[[k]])
my = matrix(yqda,dim(nmx[[k]])[1],dim(nmx[[k]])[2])
error = sum(yqda!=nmy[[k]])/length(nmy[[k]])

plot.img(my,s=3, col=T)

#--------------------------------------------------------------
# 5) GMME
library(lme4)
dat1 = data.frame(id = ind, y = factor(train.y.1), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 
dat2 = data.frame(id = ind, y = factor(train.y.2), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 
dat3 = data.frame(id = ind, y = factor(train.y.3), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 

fitglmer_1=glmer(y~x0+x1+x2+(1|id), data=dat1, family='binomial')
fitglmer_2=glmer(y~x0+x1+x2+(1|id), data=dat2, family='binomial')
fitglmer_3=glmer(y~x0+x1+x2+(1|id), data=dat3, family='binomial')
summary(fitglm_1)
save(fitglmer_1,fitglmer_2,fitglmer_3, file="fitglmer")
```

```
newdat = data.frame(id = rep(1,length(vmx[[k]])), x0 = vmx[[k]], x1 = vmx_1[[k]], x2 = vmx_2[[k]])#,
yglmer_1=predict(fitglmer_1, newdata=newdat, type='response')
yglmer_2=predict(fitglmer_2, newdata=newdat, type='response')
yglmer_3=predict(fitglmer_3, newdata=newdat, type='response')

yglmer = apply(cbind(yglmer_1,yglmer_2,yglmer_3),1,which.max)-1
table(yglmer)
table(nmy[[k]])
my = matrix(yglmer,dim(nmx[[k]])[1],dim(nmx[[k]])[2])
error = sum(yglmer!=nmy[[k]])/length(nmy[[k]])

plot.img(my,s=3, col=T)
#######################################
train.x=NULL
train.x.1=NULL
train.x.2=NULL
train.x.3=NULL
train.yy= NULL
train.xx=NULL

for(i in c(1,2,3,5,6,7,8,9)){
  train.x=c(train.x,vmx[[i]])
  train.x.1=c(train.x.1,vmx_1[[i]])
  train.x.2=c(train.x.2,vmx_2[[i]])
  train.x.3=c(train.x.3,vmx_3[[i]])}


for(i in c(1,2,3,5,6,7,8,9)){
  train.yy=c(train.yy,vmy[[i]])}



test =  data.frame(x0 = vmx[[4]], x1 = vmx_1[[4]], x2 = vmx_2[[4]], x3 = vmx_3[[4]])
t = data.frame(y = factor(vmy[[4]]))
test=cbind(t,test)

train = data.frame(y = factor(train.yy), x0 = train.x, x1 = train.x.1, x2 = train.x.2, x3 = train.x.
## Analysis k-means
set.seed(22)
x.train<-train[,-1]
error=rep(0,19)
my=list()
for(m in 2:20){
km<-kmeans(x.train, m,nstart = 20)
v=as.data.frame(cbind(y=as.numeric(train.yy),x=km$cluster))
t=as.data.frame(table(v),head=TRUE)# counting classes in each prototype
t$y=as.numeric(t$y)
x.test=test[,-1]
class=matrix(rep(0,m),ncol = 1)
for (k in 0:(m-1)){  ## asigning class labels to prototypes
  j=3*k+1
  c=as.data.frame(rbind(t[j,],t[j+1,],t[j+2,]))
  class[k+1]= c[which.max(c[,3]),1]-1
}
class
centers=km$centers
x.test=test[,-1]
y.hat<-NULL
dist<-NULL  ### calculating distance of test features from centers(prototypes)
for (i in 1:nrow(x.test)){
  dist<-NULL
  for (j in 1:m){
    dist[j]=dist(rbind(centers[j,],x.test[i,]),method = "euclidean")
  }
  y.hat[i]<- class[which.min(dist)]
}
my[[m]] = matrix(y.hat,dim(nmx[[4]])[1],dim(nmx[[4]])[2])
error[m] = sum(y.hat!=nmy[[4]])/length(nmy[[4]])
}
#save(my,file="result.Rdata")
#save(error,file="errorkm.Rdata")
load("result.Rdata")
load("errorkm.Rdata")
plot.img(my[[11]],s=1, col=T)

##### KNN
library(class)
y.test<- test[,1]
y.train<- train[,1]
error.knn=rep(0,11)
knn.table<-list()
my.knn<-list()
tunning.param<-c(1,3,5,10,15,17,20,25,35,50,100)
set.seed (22)
for( i in 1:11){
    knn.fit =knn ( x.train,x.test, y.train ,k=tunning.param[i])
    knn.table[[i]]<-table (knn.fit , y.test)
    c<-as.data.frame(table(knn.fit , y.test))
    error.knn[i]<-sum(c$Freq[which(c$y.test!=c$knn.fit)]/sum(c$Freq))
    my.knn[[i]]=matrix(knn.fit,dim(nmx[[4]])[1],dim(nmx[[4]])[2])
}
#knn.table[11]
#save(my.knn,file="resultknn.Rdata")
#save(error.knn,file="errorknn.Rdata")
#my.knn2 = matrix(my.knn2,d[1],d[2])
#plot.img(my.knn2,s=1, col=T)
load("resultknn.Rdata")
load("errorknn.Rdata")
#-------------------------------
###########Principal component#########
prin.comp<- prcomp(x.test, scale=TRUE)
pc<-as.matrix(cbind(prin.comp$x[,1],prin.comp$x[,2]))
ukm<-kmeans(pc,10, nstart = 5)
plot(pc, col =( ukm$cluster +1) , main ="K- Means Clustering
Results with K=10", xlab ="", ylab ="", pch =20, cex =2)
ukm1<-kmeans(x.test,10,nstart=20)
clust1 = matrix(ukm1$cluster,dim(nmx[[4]])[1],dim(nmx[[4]])[2])
plot.img(clust1,s=1, col=T)
```

```
#######__________________K-Center
set.seed(22)
center=list(NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA)
a<-sample(1:nrow(x.test),1)
center[[1]]<-x.test[a,]
data<-x.test
dist=NA
class<-NA
for( i in 1:nrow(data)){
dist[i]<-dist(rbind(center[[1]],data[i,]),method = "euclidean")
}
for( j in 2:11){
  center[[j]]<-data[which.max(dist),]
  for( k in 1:nrow(data)){
    d<-NA
    for(n in 1:j){
      d[n]<-dist(rbind(center[[n]],data[k,]),method = "euclidean")}
    dist[k]<-min(d)
    class[k]<- which.min(d)
  }
}
save(center,file="k-center.Rdata")
save(class,file="k-center-class.Rdata")
#####K-mean with K-center initialization
save(x.test,file="x.test.Rdata")
centers=as.numeric(center[[1]])
save(y.test,file="y.test.Rdata")
for (i in 2:11){
  centers<- rbind(centers,center[[i]])
}

km.kcenter<-kmeans(x.test, centers = centers,nstart = 1)
my.kc = matrix(classs,dim(nmx[[4]])[1],dim(nmx[[4]])[2])
plot.img(my.kc,s=1, col=T)

v=as.data.frame(cbind(y=as.numeric(y.test),x=km.kcenter$cluster))
t=as.data.frame(table(v),head=TRUE)# counting classes in each prototype
t$y=as.numeric(t$y)
class.kmean.kcenter=matrix(rep(0,length(y.test)),ncol = 1)
for (k in 0:10){  ## asigning class labels to prototypes
  j=3*k+1
  c=as.data.frame(rbind(t[j,],t[j+1,],t[j+2,]))
  class.kmean.kcenter[which(km.kcenter$cluster==(k+1))]= c[which.max(c[,3]),1]-1
}
my.kc.1 = matrix(class.kmean.kcenter,dim(nmx[[4]])[1],dim(nmx[[4]])[2])
plot.img(my.kc.1,s=1, col=T)
error = sum(class.kmean.kcenter!=y.test)/length(y.test)
error
```

References

[1] Gonzalez, R. S., and Paul Wintz. "Digital image processing." (1977).

[2] A. M. Vilkin, I. V. Safonov, M. A. Egorova. Algorithm for segmentation of documents based on texture features // Pattern Recognition and Image Analysis March 2013, Volume 23, Issue 1, pp 153-159