Lista de Exercícios - PCA e classificação de imagens

Renato Assunção - DCC, UFMG

2018

Nesta lista, a parte inicial é apenas para sua orientação. Ela mostra um sistema de classificação de faces em fotos e você não precisa entregar nada acerca desta parte inicial. O que você precisa entregar é um exercício quase idêntico para classificar dígitos manuscritos usando a mesma técnica desta parte inicial. Veja a seção **Exercícios** no final deste documento.

1 Reconhecimento de faces com componentes principais

O arquivo `DadosYaleFaces.rar`, na página da disciplina, contem diretórios com arquivos de fotos de 15 indivíduos, um diretório para cada indivíduo. Dentro deles, temos 11 fotos de cada indivíduo. As fotos variam de acordo com aspectos tais como iluminação, expressão (sorrindo, sério, triste), pela presença de óculos ou não. A Figura **??** exibe uma amostra dessas fotos. Cada linha de fotos corresponde a um indivíduo. As imagens são normalizadas para alinhar os olhos e bocas. Eles aparecem mais ou menos no mesmo local na imagem.

Figura 1: Amostra de 5 fotos de 4 indivíduos.

Vamos fazer uma análise das fotos via componentes principais com o objetivo de reconhecer estes rostos. Imagine que você tenha uma base de dados com várias fotos de um conjunto de indivíduos. Você vai especificar um sistema de vigilância para uma companhia e apenas os 15 indivíduos destas fotos podem entrar num certo local. A checagem é feita automaticamente com uma nova foto tirada no momento da tentativa de entrada. Vamos usar o PCA para criar um sistema para classificar esta nova foto a uma

das 15 classes representadas pelos diferentes indivíduos. Assim,o problema é: Chega uma nova foto. Queremos encontrar o rosto mais parecido com a nova foto no banco de dados. Se o rosto mais parecido não estiver próximo o suficiente da nova foto, a entrada não é permitida.

O método das *autofaces* foi proposto por Turk and Pentland (1991a, 1991b). Ele é principalmente um método de redução de dimensionalidade, podendo representar muitos indivíduos com um conjunto relativamente pequeno de dados. A idéia é representar a foto de um rosto como uma soma de um rosto médio mais uma combinação linear de um pequeno número de pseudo-fotos, que são as autofaces. Estas autofaces são fotos embaçadas que capturam aspectos importantes da composição de uma face.

Podemos imaginar cada foto sendo aproximadamente obtida como representado na Figura **??**. As fotos à esquerda são aproximadamente iguais a uma mesma foto média (a primeira do lado direito) mais quatro autofaces, cada uma delas multiplicada por um peso w_{ij} que é específico do indivíduo. Diferentes indivíduos vão variar apenas nos 4 pesos w_{ij} que cada autoface recebe. As autofaces são fixas e as mesmas para todos os indivíduos considerados, bem como a face média, ue também é a mesma para todos.

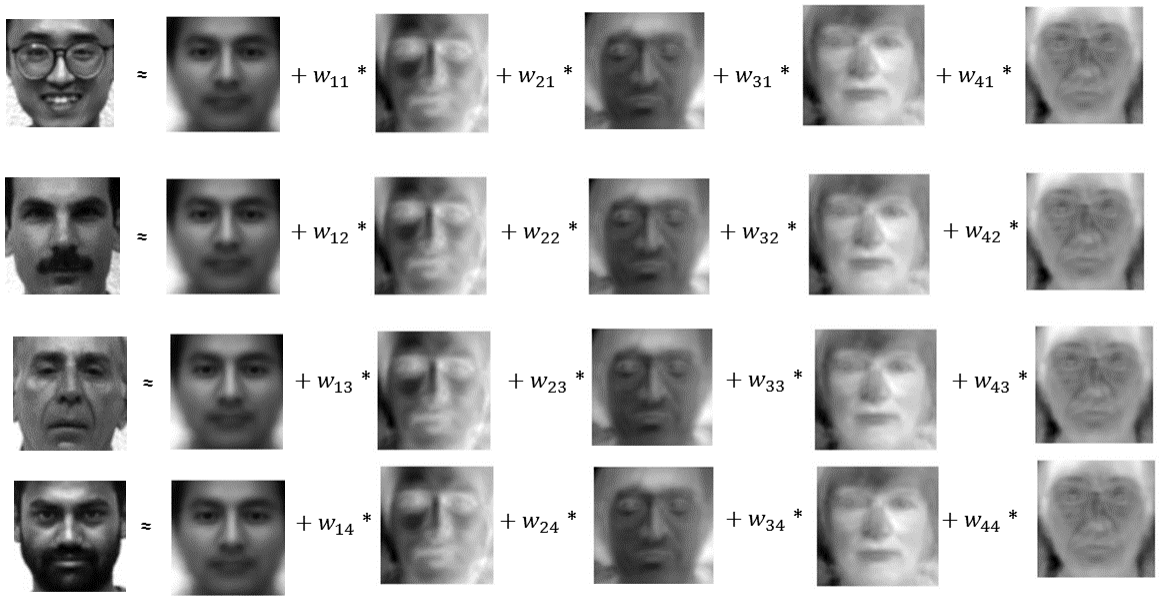

Figura 2: Uma foto (a esquerda) é aproximandamente a soma de uma foto média mais quatro autofaces multiplicada por pesos w_{ij} específicos do indivíduo. Diferentes indivíduos vão variar apenas nos 4 pesos w_{ij} que cada autoface recebe.

Neste exercício, você deve reproduzir a análise abaixo em `R`. Primeiramente, leia as as fotos no `R`. Vou usar o pacote `imager` que fornece algumas funcionalidades para processamento de imagens no `R`. Informações em `http://dahtah.github.io/imager/gettingstarted.html`. O pacote `imager` is baseado em CImg, uma biblioteca em C++ criada por David Tschumperle (CNRS). `imager` está no CRAN e pode ser instalada a partir da linha de comando (se conectado a web). Depois de instalar, carregue o pacote para a sessão de trabalho.

```
install.packages("imager")
library(imager)
```

A função `load.image` lê imagens em arquivos ou URLs. Formatos de imagens que têm suporte atualmente

são os seguintes: JPEG, PNG e BMP. Outros formatos pedem a instalação de `ImageMagick`. Vamos mudar o diretório de trabalho para aquele que contem as imagens. usando `setwd("dir_com_imagens")`. A leitura da foto `s5.jpg` no diretório `Faces3` é feita com o comando `load.image`. A seguir, visualize a foto e imprima uma informação básica sobre a foto:

```
im <- load.image('Faces3/s5.jpg')
plot(im)   # O eixo vertical corre na direcao contraria ao usual
im
# A saida eh o que vai abaixo:
# Image. Width: 100 pix Height: 100 pix Depth: 1 Colour channels: 3
```

O objeto `im` é um objeto do tipo imagem com 100×100 pixels (width e height). Depth indica quantos frames tem a imagem. Se depth > 1 então a imagem é um vídeo. `im` possui três canais de cores, o usual sistema RGB (red-green-blue channels).

O comando `is.array(im)` retorna T mostrando que, na prática, `im` é apenas um array 3-dim para permitir a leitura de fotos coloridas com os 3 canais rgb. Cada slice do array armazena uma das cores-canais fundamentais (3 canais, red-green-blue ou rgb channels com intensidades em cada pixel). Entretanto, os 3 slices são idênticospois nossas imagens não são coloridas. Elas são apenas pixels com diferentes intensidades de cinza. Vamos checar que os 3 slices do array 3-dim sao os mesmos verificando apenas uma pequena parte:

```
im[1:5, 1:5, 1] == im[1:5, 1:5, 2]
# Como os 3 slices do array sao identicos, reduzimos a uma matriz 2-dim
im_mat = im[, , 1]
is.matrix(im_mat)
dim(im_mat) # Agora temos uma matriz 100 x 100
plot(im_mat) # im_mat ja nao eh mais uma imagem
image(im_mat) # heat map da matriz im_mat.
# O comando image eh do R basico e faz um heat map de matriz numerica
# Histograma dos tons de cinza dos 100*100 pixels da matriz im_mat
hist(im_mat)
# O tom de cinza eh um real entre 0 e 1 (ao inves do inteiro de 0 a 255)
# Uma amostra aleatoria de 5 desses pixels.
sample(im_mat,5)
```

Vamos agora ler todas as fotos e começar de fato a análise. Vamos declarar uma lista para receber as fotos de todos os individuos: `fotos = list()`. Esta é uma lista de tamanho 15 e cada elemento desta lista é uma outra lista que contem 11 fotos de um mesmo indivíduo. As fotos estão em 15 diretórios, um para cada indivíduo, e nomeados como `Faces1, Faces2`, ..., `Faces15`. Dentro de um diretório `Facesi` encontramos as 11 fotos, em formato `.bmp` e `.jpg`, nomeadas `s1.bmp`, `s2.bmp`, ..., `s11.bmp` e `s1.jpg, s2.jpg`, ..., `s11.jpg`. Cada uma dessas fotos correspondem a diferentes poses de um mesmo indivíduo. Vamos ver como fazer a leitura das fotos e armazenamento na lista `fotos`.

```
fotos = list()
for(i in 1:15){
  fotos[[i]]=list() # elemento i da lista individuos eh uma lista tambem
  for(j in 1:11){
    # Leitura dos arquivos de fotos do individuo i
    # Sao 11 fotos no diretorio Faces#i
    fotos[[i]][[j]] <- load.image(sprintf("Faces%i/s%i.jpg",i,j))
  }
}
plot(fotos[[9]][[7]]) # Exibe a foto 7 do individuo 9
```

Vamos ver algumas fotos aleatórias. Cada linha de fotos será um indivíduo distinto. Para que todas as fotos caibam na janela gráfica, vamos eliminar o espaço deixado (como default) nas marges dos gráficos. Para isto, vamos alterar os parâmetros gráficos. Mas antes, vamos guardar uma cópia dos parâmetros gráficos default para restaurá-los no final:

```
opar <- par()  # salve parametros graficos
## elimine os espacos brancos nas margens e prepare a janela
## grafica para receber 4*5=20 fotos
par(mfrow=c(4,5), mar=c(0,0,0,0))
## plot as 5 primeiras fotos dos individuos 1, 3, 8 e 10
for(i in c(1, 3, 8, 10)){
  for(j in 1:5) plot(fotos[[i]][[j]], axes=F)
}
## restaure as opcoes graficas default
par(opar)
```

Para a análise de componentes principais, vamos converter as fotos em matrizes e colocá-las numa lista `fotosmat`:

```
fotosmat = list()
for(i in 1:15){
  fotosmat[[i]]=list()
  for(j in 1:11){
    fotosmat[[i]][[j]] = fotos[[i]][[j]][ , , 1]
  }
}
# Checando se todas as 11 fotos do individuo 5 sao matrizes
sapply(fotosmat[[5]], is.matrix)
# todas as 11 matrizes-fotos sao de dimensao 100 x 100
sapply(fotosmat[[5]], dim)
```

Vamos empilhar as colunas de cada matriz formando uma única coluna. Com isto, perdemos a dimensão espacial dos pixels da imagem. A seguir, coletamos as colunas numa grande matriz. O comando `stack` faz isto mas ele funciona apenas com dataframes, não funciona com matrizes. Assim, transformamos a matriz em um dataframe e usamos o comando `stack` para empilhar. O comando `stack` retorna uma matriz com *duas* colunas: numa, ficam os valores das colunas empilhados; na outra, ficam os índices das colunas na matriz original. Procure entender usando com este exemplo simples: `stack(as.data.frame(matrix(1:6, ncol=2)))`. Assim, precisamos da *primeira* coluna da saida de `stack`. Montamos a matriz com os vetores empilhados.

```
mat_pixels = matrix(0,nrow=(100*100), ncol=11*15)
for(i in 1:15){
  for(j in 1:11){
    mat_pixels[,j+(i-1)*11] = stack(as.data.frame(fotosmat[[i]][[j]]))[,1]
  }
}
```

Vamos separar uma foto de cada indivíduo para fazer sua classificação. Isto vai constituir o conjunto de teste, onde vamos avaliar o nosso método de classificação como se estas fotos separadas fossem novas fotos. Ficaremos com uma matriz com $150 = 15 \times 11 - 15$ colunas pois existem 15 indivíduos com 11 fotos cada um. Vamos escolher uma foto ao acaso de cada indivíduo. Comece fixando a semente de números aletórios:

```
set.seed(123)
## indice do numero da foto, dentro de cada individuo
ind = sample(15, 1:11, replace=T)

## pegando agora os indices das colunas de cada foto em mat_pixels
indcol = ind + ((1:15) - 1) *11

# separando as fotos para teste posterior. Elas estao numa matriz
# com 15 colunas ordenadas de acordo com o indice dos individuos.
mat_teste = mat_pixels[ , indcol]
dim(mat_teste)

## Retirando as colunas de teste da matriz onde vamos aplicar PCA:
mat_pixels = mat_pixels[,-indcol]
```

Precisamos centrar todas as fotos do conjunto de treinamento subtraindo de cada foto a foto média de todo o conjutnode fotos. Esta foto média é simplesmente a "fotoÂ´Â´ obtida tirando a média aritmética sobre o conjunto de fotos em cada pixel. Isto é, para um pixel localizado numa certa posição, tiramos a média de todos os valores observados naquela posição nas diferentes fotos do conjunto de treinamento. Em `R`, isto é muito simples:

```
mat_media = apply(mat_pixels, 1, mean)
mat_centrada = mat_pixels - mat_media
```

Vamos desempilhar esta foto média e visualizá-la. Para desempilhar, nós quebramos a coluna `mat_centrada` em 100 colunas de tamanho 100, gerando uma "imagemÂ´Â´ com a mesma quantidade de pixels que as fotos originais.

```
foto_media = as.cimg(mat_media, x=100, y=100)
par(mfrow=c(1,1))
plot(foto_media, axes=F)
```

Obtemos agora os componentes principais da matriz *transposta* `t(mat_centrada)` com 150 itens e 10000 atributos. A matriz $\mathbf{\Sigma}$ de covariância dos atributos (os pixels, neste caso) é uma matriz de dimensão 10000×10000. Devemos portanto obter 10000 autovetores e autovalores. Vamos usar a função `princomp` do `R`.

```
pca_pixels = princomp(t(mat_centrada))
```

```
Error in princomp.default(t(pca_pixels = princomp(mat_centrada))) :
  'princomp' can only be used with more units than variables
```

A matriz `mat_centrada` possui mais atributos-colunas ($p = 10000$) que itens-linhas ($n = 150$). Nesta situação, `princomp` não funciona. O comando sabe que a matriz de covariâancia empírica de dimensão $p \times p$ baseada numa matriz de dados de dimensão $n \times p$ possui posto igual ou menor ao mínimo entre n e p. Isto significa que a matriz de covariância da matriz de dads 150×10000 é de dimnsão 10000×10000 e com posto150. Isto implica que existem $10000 - 150$ autovalores exatamente nulos e o algoritmo de `princomp` não vai rodar. Uma saída simples é usar outro comando, `prcomp`, que usa a decomposicao do valor singular (SVD) e não se incomoda com as dimensões da matriz de dados. Uma vantagem adicional é que o algoritmo SVD paraencontrar autovaloes e autovetores é mais estável numericamente e deveria ser preferido mesmo quando a matriz tem mais linhas-itens que colunas-atributos.

```
pca_pixels = prcomp(t(mat_centrada))
summary(pca_pixels)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7
Standard deviation     8.5911 7.9668 6.8476 4.96998 4.10745 3.83001 3.27323
Proportion of Variance 0.2041 0.1755 0.1297 0.06831 0.04666 0.04057 0.02963
Cumulative Proportion  0.2041 0.3796 0.5093 0.57759 0.62425 0.66481 0.69444
                           PC8    PC9    PC10    PC11    PC12    PC13    PC14
Standard deviation     3.00237 2.8962 2.52647 2.35629 2.23969 2.14522 2.06362
Proportion of Variance 0.02493 0.0232 0.01765 0.01535 0.01387 0.01273 0.01178
Cumulative Proportion  0.71937 0.7426 0.76022 0.77557 0.78944 0.80217 0.81394
                          PC15    PC16    PC17    PC18    PC19    PC20    PC21
Standard deviation     1.84285 1.83284 1.74649 1.66961 1.63947 1.58556 1.48814
Proportion of Variance 0.00939 0.00929 0.00844 0.00771 0.00743 0.00695 0.00612
Cumulative Proportion  0.82334 0.83263 0.84106 0.84877 0.85620 0.86315 0.86928
...
```

Veja a variância explicada por cada um dos componentes. Os 10 primeiros PCs explicam 76% da variação total. Acrescentar mais 10, ficando com 20 PCs, leva a 86% A saída do comando `prcomp` mostra apenas os 150 primeiros componentes pois os outros $10000 - 150$ componentes principais têm autovalor igual a zero. Os 150 primeiros autovetores, de dimensão 10000 cada um, formam as colunas da matriz `pca_pixels$rot`.

```
dim(pca_pixels\$rot)

## Grafico scree com os 10 primeiros autovalores (ou
## variancias dos 10 1os PCAs)
plot(pca_pixels)

## Como sao muitos autovalores, defaultdo plot usa apenas os 10 1os
## Para ver TODOS os 150, podems extrai-los do objeto sdev
autovalores = (pca_pixels$sdev)^2

## Barplot das variancias acumuladas indicando a escolha de poucos
## PCAs devem representar bem os 10000 atributos (pixels)
barplot(cumsum(autovalores))

## Vamos normalizar o eixo vertical dividindo pela soma total
aux = cumsum(autovalores)/sum(autovalores)
barplot(aux)

## Procurando ver a parte inicial do grafico com mais detalhes
barplot(aux[1:30], ylim=c(0,1))

## Vamos usar os 20 primeiros autovetores.
## Eles sao vetores de dimensao 10000 = 100 * 100

autovetores = pca_pixels$rot[ , 1:20]
```

Vamos criar agora as autofaces. Vamos desempilhar os autovetores quebrando a coluna de cada um deles em 100 colunas de tamanho 100 cada uma. Com isto, cada autovetor dá origem a uma "imagemÂ´Â´ com a mesma quantidade de pixels que as fotos originais. Vamos chamar estas pseudo-fotos de *autofaces*

```
## Criando as autofaces
```

```
auto_face=list()

for(i in 1:20){
  auto_face[[i]] = as.cimg(pca_pixels\$rot[,i], x=100, y=100)
  #100x100, grayscale image
}

## Vendo estas 20 autofaces
par(mfrow=c(4,5), mar=c(0,0,0,0))
for(i in 1:20) plot(auto_face[[i]], axes=F)
```

Vamos visualizar uma face em `mat_teste` e sua aproximação usando as k primeiras autofaces. Para isto precisamos escrever uma foto como aproximadamente uma soma da foto média mais uma combinação linear das primeiras k autofaces (todos como vetores). Em seguida, desempilhamos os vetores e mostramos as faces.

Por exemplo, usando a face na coluna 5 de `mat_teste`. Vamos obter os coeficientes b_k da combinação linear

$$mbox{foto} \approx \text{fotomedia} + b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \ldots + b_k\mathbf{v}_k$$

Sabemos que b_j é o produto interno dos vetores formados pela foto centrada com o autovetor $\mathbf{v}_k$. Então

```
coef = t(autovetores) %*% (mat_teste[, 5] - mat_media)
dim(coef)
# [1] 20  1
```

A matriz `coef` é uma *matriz* 20 x 1 com os coeficientes $(b_1, b_2, \ldots, b_{20})$. Este vetor-coluna é a representação da foto no espaço gerado pelos 20 primeiros autovetores. Nesta base de autovetores, o vetor `coef` representa a foto aproximadamente. Os comandos abaixo geram uma aproximacao da foto 5 em `mat_teste` usando 2, 3, 4 até 20 autovetores resultando na Figura **??**.

Figura 3: Um indivíduo e sucessivas aproximações usando 2, 3, até 20 autovetores.

```
foto_teste5 = list()
```

```
foto_teste5[[1]] = as.cimg(mat_teste[,5], x=100, y=100)
for(i in 2:20){
  aprox_vetor = mat_media + autovetores[, 1:i] %*% coef[1:i,1]
  foto_teste5[[i]] = as.cimg(as.numeric(aprox_vetor), x=100, y=100)
}

## guarde uma copia dos parametros graficos default
opar <- par()
## elimine os espacos brancos nas margens e prepare para 4*5=20 fotos
par(mfrow=c(4,5), mar=c(0,0,0,0))
## plot as fotos
for(i in 1:20) plot(foto_teste5[[i]], axes=F)
## restaure as opcoes graficas default
par(opar)
```

A primeira imagem é a imagem real. As seguintes fornecem as aproximações usando sucessivamente 2, 3, até 20 autofaces. Parece que usar 20 autofaces talvez seja excessivo. Visualmente, não existe uma diferença relevante entre a aproximação obtida usando apenas os 12 primeiros autovetores ou aquela com 20 autovetores. Apesar disto, vamos repetir esta aproximação usando 20 autofaces para todas as fotos-colunas em `mat_teste`, salvando os coeficientes numa matriz `coef` de dimensão 20×15:

```
coef = t(autovetores) %*% (mat_teste - mat_media)
dim(coef)
# [1] 20 15
```

Como classificar cada uma destas 15 fotos em uma das categorias disponíveis, as categorias sendo os 15 indivíduos? Qual será a taxa de acerto deste sistema de classificação? Vamos primeiro obter a representação de cada uma das 150 fotos de treino nos 20 PCAs, exatamente como fizemos com a fotos de teste

```
coef_treino = t(autovetores) %*% (mat_pixels - mat_media)
## coef_treino eh matriz 20 x 150
```

Podemos ver como as 150 fotos do treino ficam dispostas no plano determinado apenas pelas duas primeiras componentes principais. Supostamente, os dois primeiros componentes principais não deve ser suficiente para discriminar bem os indivíduos.

```
par(mfrow=c(1,1))
colface = rainbow(15)[rep(1:15,rep(10,15))]
plot(coef_treino[1,], coef_treino[2,], pch=21, bg=colface)
```

Vamos obter uma representação *média* de cada um dos 15 indivíduos. Vamos tirar a média dos coeficientes das 10 fotos de cada indivíduo.

```
coefmedio = matrix(0, ncol=15, nrow=20)
for(i in 1:15){
  coefmedio[,i] = apply(coef_treino[ ,(1+(i-1)*10) : (i*10)], 1, mean)
}
## Esta eh uma matriz 20 x 15

## Vamos ver como os 15 perfis medios do treino no plano
```

```
## determinado apenas pelas duas primeiras componentes principais

par(mfrow=c(1,1))
colface = rainbow(15)
plot(coefmedio[1,], coefmedio[2,], pch=21, bg=rainbow(15))
```

Cada foto de `mat_teste`, na representação dos 20 primeiros PCAs, é uma coluna da matriz `coef` que tem dimensão 20×15. A matriz `coefmedio` também é de dimensão 20×15. Para cada foto (isto é, cada coluna de `coef`), vamos encontrar a coluna de `coefmedio` que é a mais próxima. A ordem desta coluna mais próxima é o indivíduo mais próximo.

```
indproximo = numeric()
for(j in 1:15){
  indproximo[j] = which.min( apply((coefmedio - coef[,j])^2, 2, mean) )
}

indproximo
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
## Voila!! Classificacao perfeita!!!
```

O vetor `indproximo` indica que a primeira foto em `mat_teste` (coluna 1) foi classificada ao primeiro indivíduo, o que é a decisão correta. Ele também indica que a segunda foto em `mat_teste` (coluna 2) foi classificada ao segundo indivíduo, o que também é correto. Olhando o resto do vetor,fica claro que a classificação foi a correta para todas as 10 fotos do conjunto de teste.

Em conclusão, uma versão aproximada de uma foto em escala de cinza de um rosto humano pode ser obtida como uma combinação linear de umas poucas autofaces (*eigenfaces*, em inglês):

$$\text{foto} = \text{média geral} + c_1\mathbf{v}_1 + \ldots c_k\mathbf{v}_k$$

Os autovetores ou PCAs $\mathbf{v}_1, \ldots, \mathbf{v}_k$ da matriz de covariância $\mathbf{\Sigma}$ da distribuição conjunta dos pixels formam as autofaces. É impressionante que apenas algumas poucas, as primeiras k, autofaces ou PCAs sejam suficientes para obter uma boa semelhança dos rostos da maioria das pessoas. As autofaces se parecem com um rosto humano médio, sem muitos traços distintivos.

O método de autofaces foi proposto por Turk and Pentland (1991a, 1991b). Ele é principalmente um método de redução de dimensionalidade, podendo representar muitos indivíduos com um conjunto relativamente pequeno de dados. De fato, no nosso problema, temos uma base de 10 fotos de 15 indivíduos, cada foto armazenada como uma matriz com 10^4 elementos. Assim, temos, ao todo, $10 \times 15 \times 10^4 = O(10^6)$ bytes para armazenar. Vamos imaginar uma base maior com n indivíduos implicando em $O(n10^5)$. Ao chegar uma nova foto precisamos compará-la com as $10n$ fotos por um procedimento ingênuo (naive). Com as autofaces, guardamos apenas 20 pseudo-fotos, as autofaces (possivelmente, apenas 12 seriam suficientes), e os coeficientes médios de cada indivíduo. Veja que o número de autofaces não varia muito com n. Mesmo que n seja muito grande, teremos apenas um número pequeno de autofaces. Supondo que sejam 20 autofaces, isto significa guardar $(20 \times 10^4) + (20 \times n)$. Por exemplo, com $n = 1000$ e 10 fotos para cada um e com 20 autofaces teríamos que guardar e manipular 2.2×10^5 enquanto que uma comparação ingênua requer armazenar 10^8, ou 1000 vezes mais espaço.

No entanto, o método de autofaces pode ter um desempenho muito ruim se existir muita diferença entre as imagens na base de treinamento e as novas imagens (Moon and Phillips, 2001). Uma imagem de um indivíduo sob iluminação frontal pode ter coeficientes muito diferentes do mesmo indivíduo, na mesma pose, sob iluminação lateral intensa. Assim, as novas fotos devem ter representantes similares na base de treino.

Autofaces é uma técnica antiga e já existem muitas variações e melhorias, bem como outra abordagens completamente diferentes para o mesmo problema. Para quem quiser se aprofundar sobre as muitas

outras técnicas envolvidas com o reconhecimento de faces, visite o website `http://www.face-rec.org/general-info/`.

---

2 Exercício: Reconhecimento de dígitos

Este exercício é praticamente a mesma coisa que foi feita acima para as fotos. Ele foi extraído da página web do livro página do livro *The Elements of Statistical Learning*, por Hastie, Tibshirani e Friedman. Este excelente (e avançado) livro está disponível para download gratuito e legal na página `http://statweb.stanford.edu/~tibs/ElemStatLearn/`.

O objetivo é construir um algoritmo em `R` para a classificação de dígitos escritos á mão. Os dados são uma parte da base *US Postal Service Database* e correspondem à digitalização de números de CEP escritos a mão em correspondências enviadas pelo correio americano. Estes dados estão na página do livro, onde é chamado de `ZIP code` (é o último da lista de datasets).

O conjunto de dados refere-se a dados numéricos obtidos a partir da digitalização de dígitos escritos à mão a partir dos envelopes pelo Serviço Postal dos EUA. Imagens em preto e branco foram normalizadas em termos de seu tamanho de forma a caber em uma caixa de pixels 20×20, preservando a sua razão de aspecto (aspect ratio). As imagens resultantes contêm níveis de cinza como um resultado da técnica de anti-aliasing usada pelo algoritmo de normalização. As imagens foram centradas em uma imagem 28×28 calculando o centro de massa dos pixels e traduzindo a imagem de modo a posicionar este ponto no centro da matriz 28×28. O resultado final são imagens 28×28 em tons de cinza.

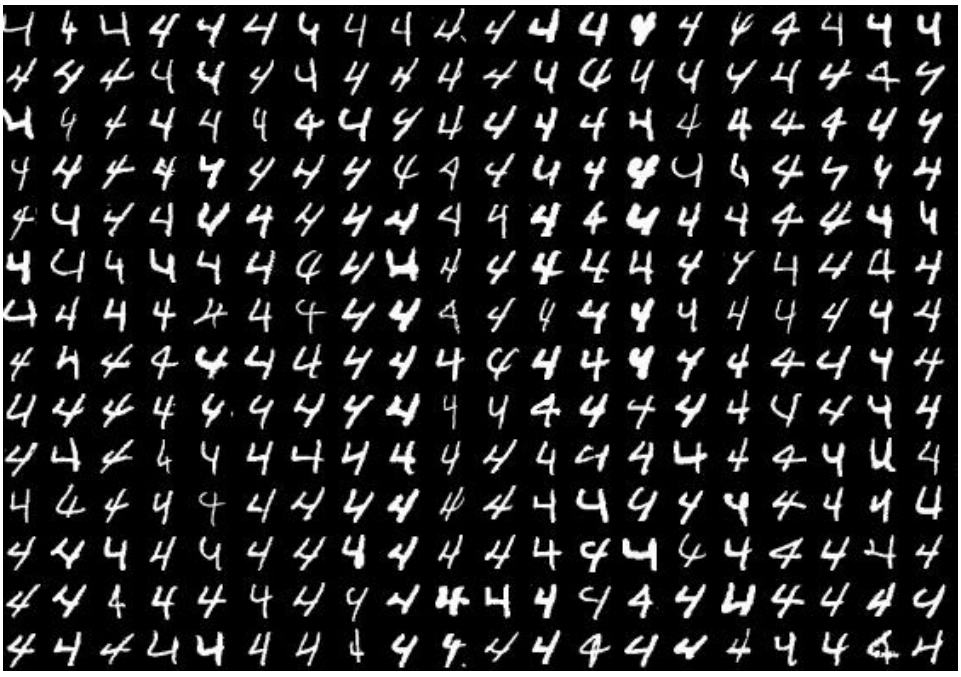

Figura 4: Imagens dos dígitos 4 da base USPS.

A Figura **??** mostra os dígitos 4 da base de dados. O objetivo do exercício é inteiramente análogo ao de reconhecimento de faces. Queremos um método de classificação de novas imagens de dígitos manuscritos. Assim, você deverá:

- Usando um conjunto de treinamento, criar uma regra de classificação de novas imagens de dígitos.

k	precisão média	revocação média
5	??	??
6	??	??
$\vdots$	$\vdots$	$\vdots$
20	??	??

Tabela 1: Precisão e revocação do método de classificação de dígitos como função do número k de autovetores.

Use os primeiros k autovetores da matriz de covariância entre os pixels para fazer esta regra de classificação. Você deve fazer seus cálculos com $k = 5, 10, 15, 20$.

- Usando apenas a amostra de TESTE, crie uma tabela de contingência 10×10 de confusão C. Nesta matriz C as linhas representam a classe verdadeira do dígito (de 0 a 9) e a coluna a classe em que ele foi alocado. Na entrada C_{ij} você deve colocar o número de itens (ou imagens) que caíram naquela categoria cruzada. Crie esta tabela com os quatro valores distintos de $k = 5, 10, 15, 20$.
- Calcule a proporção total das imagens da amostra de teste que caem na diagonal principal. Esta é uma medida global de classificação correta do método. Para qual valor de k esta proporção foi máxima?
- Preencha uma tabela como a que está abaixo:

  Precisão média é a média aritmética da precisão das 10 classes e definida como:

$$pm = \frac{1}{10} \sum_{i=0}^{9} \frac{C_{ii}}{C_{i+}}$$

  com C_{i+} sendo a soma da linha i na matriz de confusão. Revocação média é a média aritmética da revocação das 10 classes e definida como:

$$rm = \frac{1}{10} \sum_{i=0}^{9} \frac{C_{ii}}{C_{+i}}$$

  com C_{+i} sendo a soma da coluna i na matriz de confusão. Mais detalhes sobre precisão (precision) e revocação (recall) podem ser vistos no verbete *Precision and recall* na wikipedia. Ver também `http://www.text-analytics101.com/2014/10/computing-precision-and-recall-for.html`.

References

- M. Turk, A. Pentland, (1991a), Eigenfaces for Recognition, *Journal of Cognitive Neurosicence*, Vol. 3, No. 1, 1991, pp. 71–86.
- M.A. Turk, A.P. Pentland, (1991b), Face Recognition Using Eigenfaces, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3–6 June 1991, Maui, Hawaii, USA, pp. 586–591.
- A. Pentland, B. Moghaddam, T. Starner, (1994), View-Based and Modular Eigenspaces for Face Recognition, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 21–23 June 1994, Seattle, Washington, USA, pp. 84–91.
- H. Moon, P.J. Phillips, Computational and Performance aspects of PCA-based Face Recognition Algorithms, *Perception*, Vol. 30, 2001, pp. 303–321.