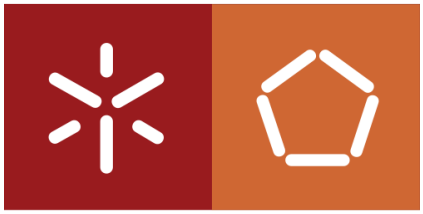


Universidade do Minho
Escola de Engenharia

Métricas em Machine Learning

Face Recognition , Grupo E

Guilherme Nunes, A80524
Guilherme Andrade, A80426
Gabriela Martins, A81987
José Costa, A82136
Rui Costa, A79947
Rui Ribeiro, A80207

Janeiro, 2020

Conteúdo

1 Introdução

Ao longo deste relatório, o grupo apresentará o desenvolvimento de um sistema de reconhecimento facial tendo em conta os conhecimentos adquiridos nas aulas. Dentro de todas as escolhas possíveis, o grupo decidiu-se pelo trabalho de reconhecimento facial, não só por representar um desafio para colocar as capacidades que se ganhou nas aulas, como também por ser um enunciado mais entusiasmante.

Para tornar o trabalho mais interessante, o grupo decidiu usar as imagens dos diferentes elementos para compor o *dataset*, de modo a obtermos um conjunto de dados de treino composto.

Neste relatório, é feita a análise do *dataset* assim como a especificação e explicação de todas as metodologias usadas de modo a manipular e interpretar estes dados, para obter os resultados pretendidos.

1.1 Estrutura do relatório

- No capítulo *Dataset* explica-se o *dataset* aplicado no caso de estudo e a sua preparação para o desenvolvimento do sistema;
- No capítulo Técnicas refere-se qual o método utilizado no reconhecimento facial e uma sucinta explicação da sua implementação e resultados;
- O relatório é concluído na Conclusão com observações relevantes.

2 *Dataset*

O projeto exigia uma base de dados de fotografias. Foi proposto ao grupo, como abordagem ao trabalho, a utilização de um *dataset* de própria autoria. O grupo decidiu que seria interessante criar a base de dados e procedeu à seleção de cinquenta fotografias diferentes em que cada dez seriam de um só indivíduo. Demonstra-se, de seguida, uma imagem de cada um destes subconjuntos de dez fotografias, para ilustrar graficamente a constituição das amostras utilizadas.

((a)) Imagem exemplo de Guilherme Andrade

((b)) Imagem exemplo de Guilherme Nunes

((c)) Imagem exemplo de Rui Ribeiro

((d)) Imagem exemplo de Rui Costa

((e)) Imagem exemplo de Gabriela Martins

A base de dados é, então, constituída por dez fotos de cinco pessoas diferentes. Em cada um destes subconjuntos, procurou-se usar expressões faciais diferentes para tornar as imagens de um mesmo indivíduo mais diferenciadas entre si. Houve o cuidado de manter alguma uniformização do *background* de cada fotografia, pois o algoritmo utilizado na tarefa pode apresentar um grau de sensibilidade considerável em relação a grandes variadades deste aspecto (ruído).

Este *dataset* necessitou de algum preprocessamento antes de ser legível para o programa. Para que seja possível comparar duas entradas, cada uma desta teria que ter o mesmo número de atributos, correspondentes entre si. Cada subconjunto de fotografias teve fontes diferentes, o que implicou, fotos com tamanhos diferentes, isto é, o número de *pixeis* são diferentes entre imagens de indivíduos. O primeiro passo constituiu um *resize* de todas as fotografias para um tamanho definido pelos autores. O valor definido foi 255x255. Foi criado um *script* em *python* que iterava por cada uma das imagens, fazia as operações de formatação e armazenava as novas instâncias noutra pasta de *output*. Essa pasta contêm cinquenta fotografias de dimensão 255x255 em formato RGB.

É de notar que houve a possibilidade de decidir entre *grey scale* e RGB, mas foi decidido que RGB seria uma opção mais interessante, puramente do ponto de vista científico.

Após a tarefa de *resize*, cada fotografia seria representada por uma matriz n-dimensional (255,255,3). O facto de haver uma dimensão extra é uma propriedade do RGB na definição de intensidades para os *pixels*. Procedeu-se a alterarão do formato de representação de cada foto. Estas passaram a ser representadas por um *array* de dimensão 195075, isto é, um *reshape* da matriz com todos os seus constituintes (255x255x3 = 195075). A base de dados é representada por uma matriz de duas dimensões com cinquenta (50) linhas, representativas de cada fotografia,

e cento e noventa e cinco mil e setenta e cinco (195075) colunas, representativas do valor de cada *pixel* de cada instância.

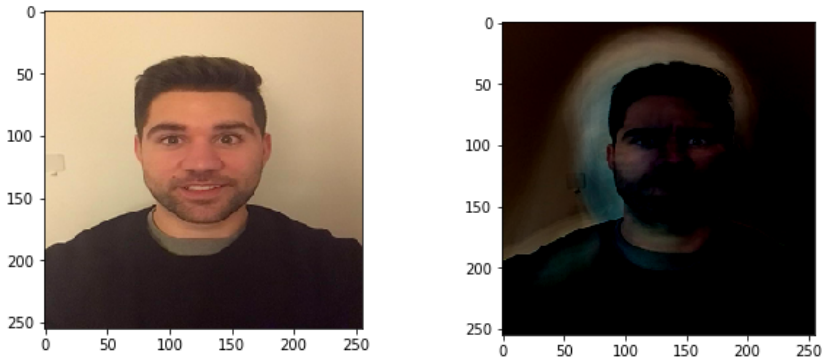
Para efeitos de teste, dividiu-se a base de dados num *train_set*, constituído por oito fotografias escolhidas sequencialmente de cada subconjunto associado a um indivíduo, e num *test_set*, constituído pelas duas restantes fotografias de cada subconjunto. Para efeitos de experiência, mais tarde, foram incluídas e testadas fotografias de outros indivíduos não presentes no *dataset* inicialmente.

3 Técnicas

Para a realização do reconhecimento facial, utilizou-se *PCA*(Análise de Componentes Principais). É um procedimento matemático que utiliza uma transformação ortogonal (ortogonalização de vetores) para converter um conjunto de observações de variáveis possivelmente correlacionadas num conjunto de valores de variáveis linearmente não correlacionadas chamadas de componentes principais. O número de componentes principais é sempre menor ou igual ao número de variáveis originais.

O conjunto de variáveis para este caso são os conjuntos de pixeis relativos às imagem. Como explicado na secção 2, os nossos dados de treino são constituídos por uma matriz $N * M$, em que **N** representa o número de imagens utilizadas para treino e **M** o número de pixeis de uma imagem *RGB*, neste caso 195075, todas estes pixeis são considerados como variáveis.

Primeiramente centrou-se todas as imagens de treino calculando a média de cada coluna da matriz *train_set* guardando os valores num *array*. Tendo a média relativa a cada coluna do *train_set*, subtrai-se ao *train_set* original obtendo assim um novo conjunto de dados centrados. O resultado desta operação é apresentado na imagem 1



((a)) Imagem inicial ((b)) Imagem centrada

Figura 1: Imagens centradas

Após termos as imagens centradas procede-se o cálculo dos vetores próprios associados aos maiores valores próprios deste conjunto de dados. Para isto utilizou-se uma biblioteca de *python*, respetivamente *Numpy*, que contém vários métodos para este problema. O método utilizado foi o SVD(singular values decomposition) que através da matriz de treino devolve os valores singulares e os vetores próprios associados por ordem decrescente. Após obtermos os valores singulares calcula-se os valores próprios elevando os valores singulares ao quadrado. O resultado é apresentado na figura 2.

Obteve-se 45 valores próprios diferentes de 0 e 195075 vetores próprios referentes a cada atributo em que cada linha contém 40 valores representativos das 40 imagens. Tendo calculado os valores e vetores próprios associados à matriz de treino, verifica-se se é possível efetuar uma redução de dimensão, por forma a preservar 90% da informação. Para saber o impacto que um valor próprio tem no conjunto de dados, dividi-se o mesmo valor pela soma de todos os valores próprios. Caso o valor não tenha o impacto pretendido, neste caso é de 90% da informação, então adiciona-se o 2 valor próprio(não esquecer que o *svd* já devolve os valores próprios por ordem descrente) e assim sucessivamente. O resultado desta operação é representado pela figura 3.

Através do método do cotovelo apenas deviam ser utilizados entre 5 a 10 valores próprios, mas decidiu-se utilizar os primeiros 29 valores, para conseguir representar pelo menos 90% da

```
print(vet_prop.shape)
print(val_prop.shape)
print(val_prop)
```

```
(195075, 40)
(40,)
[4.39719360e+02 2.73301485e+02 1.67127836e+02 1.50827492e+02
 1.08584481e+02 8.74990245e+01 8.13350750e+01 7.88017325e+01
 6.96392089e+01 6.80102864e+01 6.38640392e+01 6.00128964e+01
 5.84543831e+01 5.65968659e+01 5.20384612e+01 5.08951764e+01
 4.94098130e+01 4.70112421e+01 4.67929785e+01 4.31935154e+01
 4.16217952e+01 4.12081644e+01 3.91289184e+01 3.82947162e+01
 3.67929531e+01 3.54184425e+01 3.47959042e+01 3.43464798e+01
 3.32689115e+01 3.25904435e+01 3.04820763e+01 2.96649998e+01
 2.86908276e+01 2.86588537e+01 2.77260222e+01 2.49233430e+01
 2.33344293e+01 2.16797265e+01 1.48445360e+01 7.67079707e-14]
```

Figura 2: Valores e vetores próprios

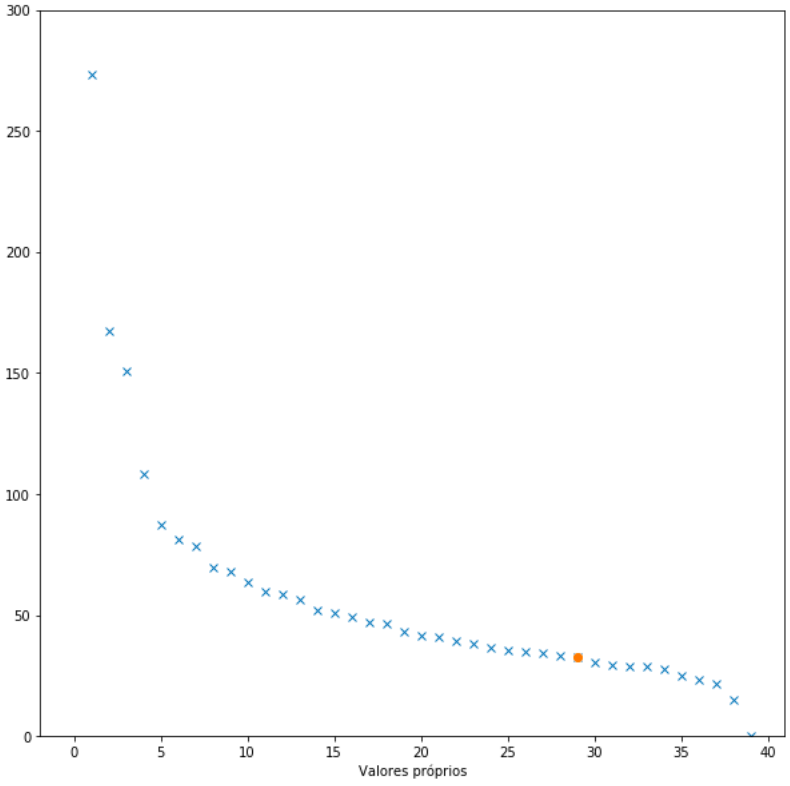


Figura 3: Análise dos valores próprios

informação. Na maior parte dos casos, exceder assim tanto o número de valores comparado com o método do cotovelo pode levar a um *overfitting* dos dados, mas neste caso, o *overfit* é muito pequeno, que se irá comprovar posteriormente.

Sabendo quais os valores próprios a utilizar, guardou-se os vetores próprios associados e começou-se o cálculo das projecções internas. Sabe-se que para representar um determinado ponto $\mathbf{x}$ numa projeção $\mathbf{W}$, faz-se o seguinte cálculo.

$$proj(x)_{<W>} = (x.W) * W$$

A primeira multiplicação é o produto interno entre o ponto e o vetor, a segunda é a multiplicação do resultado anterior por W, assumindo que este tem uma norma = 1, neste caso isso é verdade, porque o método utilizado, *svd* já normaliza estes vetores. Também se realça que o pretendido aqui é apenas a primeira multiplicação, é esse resultado que representa o valor do ponto na projeção e é este raciocínio que aplicamos no nosso caso mas com mais projeções.

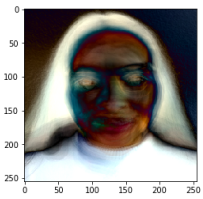
Como se pode observar na figura 4, primeiramente obtêm-se os vetores próprios associados aos **K** valores mais representativos do *dataset* e após isso calcula-se a projeção interna dos dados de treino pelos vetores próprios(29 vetores), obtendo assim os 29 pontos representativos de cada projeção por cada imagem de treino(40 imagens).

```
W = vet_prop[0:,:k]
projecao_interna = np.dot(train_set,W)
print(projecao_interna.shape)
```

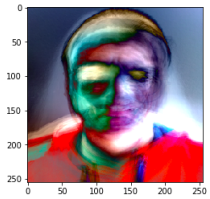
```
(40, 29)
```

Figura 4: Projeção interna

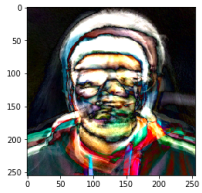
Alguns dos resultados dos vetores(*eigenfaces*) podem ser observados através das seguintes imagens:



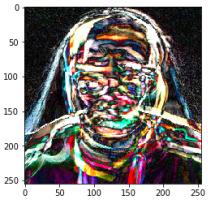
((a)) Vetor próprio 1



((b)) Vetor próprio 2



((c)) Vetor próprio 3



((d)) Vetor próprio 4

Tendo as projecções preparadas, começa-se a operar sobre *test_set*. Primeiramente centramos as imagens de teste com as médias calculadas anteriormente para as imagens de treino. Após isso, calculam-se as projeções, calculando o produto interno dos dados de teste com os **K** vetores. Todo este processo é semelhante ao descrito anteriormente, e o resultado são 29 valores por cada uma das 20 imagens de teste referentes às projeções.

Para finalizar, apenas se precisa de uma boa métrica para calcular a distância entre as projecões de cada imagem de teste com cada imagem de treino para que se consiga obter a distância mínima e a partir daí saber qual a pessoa na imagem. Utilizaram-se três métricas para calcular as distâncias, respetivamente, a distância euclidiana, normal e de *mahalanobis*. A partir desta parte todos os resultados apresentados são resultantes da utilização da distância euclidiana e numa parte final faz-se uma sucinta comparação entre métricas e resultados.

As distâncias finais são uma matriz de 20 linhas por 40 colunas, em que cada linha corresponde a uma imagem e as respetivas colunas representam as distâncias entre a respetiva imagem e todas as imagens de treino(40 imagens). Tendo estas distâncias calcula-se qual o índice que representa a distância mínima por cada linha. Antes da explicação do índice, refere-se que o *dataset* está sequencialmente estruturado, ou seja, antes na inicialização do mesmo, temos 10 imagens por cada elemento do grupo, e durante a preparação do *dataset*, as imagens de treino são importadas

sequencialmente, sendo que as primeiras 8 imagens são referentes ao 1 elemento, as próximas 8 ao 2 elemento e assim sucessivamente. Disto isto, o índice calculado serve para saber qual a pessoa que representa a distância mínima a uma respetiva imagem. Imagine-se um suposto índice de 18, então neste caso 8+8+2 = 18, sendo que este índice está entre o 3 elemento.

Uma possível representação de uma projeção das imagens em *2D* é a seguinte:

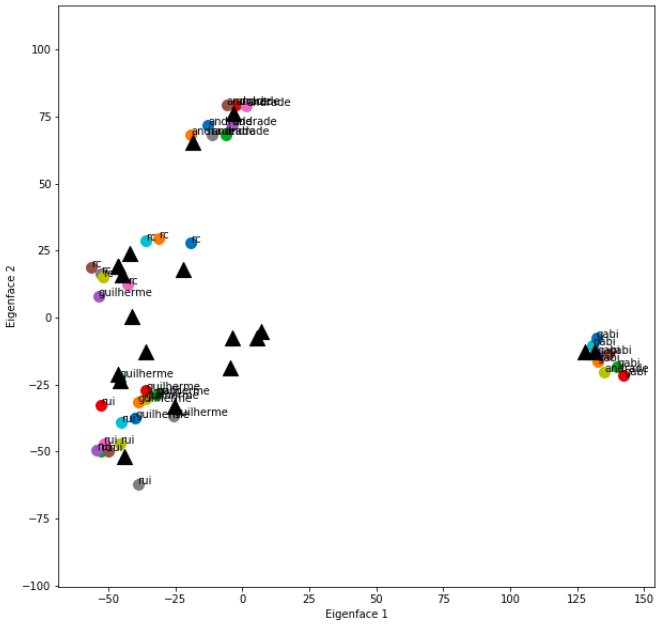


Figura 5: Projeções com os dois vetores próprios mais representativos

Conseguiu-se um modelo com 100% de precisão, este modelo não falhou para nenhuma das imagens de teste. Dado estes resultados, decidiu-se complicar um pouco. Tirou-se novas fotos de um mesmo elemento com roupas e posição diferentes, tal como um novo elemento, cujo nenhuma imagem de treino o representa.

((a)) Mesmo elemento com roupas diferentes

((b)) Novo elemento

Para o mesmo elemento, mas com posições e roupas diferentes obteve-se uma precisão ótima, nenhuma das previsões para estas imagens falhou, concluindo que o modelo não está a fazer *overfitting* dos dados, não havendo necessidade de alterar a quantidade de valores próprios, mas como era de esperar, para o novo elemento falhou, afirmando ser um dos elementos do grupo. Após isto, só foi preciso definir um limite para a distância, para caso alguma das distâncias calculadas ser superior aquele valor, então a respetiva imagem não representava ninguém do *dataset*, ou seja,

por outras palavras é necessário definir um *treshold*.

Tentou-se definir este *treshold* por tentativa e erro mas após analisar os dados relativamente às distâncias como podemos observar na imagem 6, conseguiu-se concluir que distâncias superiores a 80 normalmente devem-se ao facto de a imagem a ser avaliada não estar no *dataset*, por isso definimos o *treshold* como 80.

```
dist,d_min = euclidian_metric()
print(d_min)

#Temos que arranjar um valor para defenir um treshold, uma distancia máxima, caso passe não é nenhum dos elementos e qu
histogram_example = plt.hist(d_min, bins=100)
plt.show()
plt.boxplot(d_min)
```

```
[21.5156734  32.98058142 14.38763603 20.47230013 10.87637651 17.00965029
 52.90612606 42.46234649 36.66970797 38.46710004 71.79978286 58.63315263
 69.73848181 69.65004912 83.70621042 83.70621042 55.29256126 83.70621042
 49.39323118 50.15592356]
```

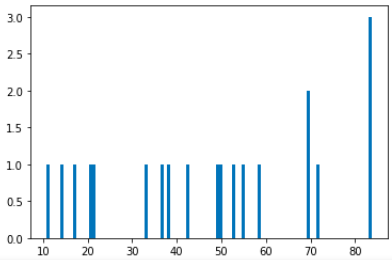


Figura 6: Distâncias euclidianas

Tendo todos os passos concluídos apenas, 1 imagem em 20 nos falhou, relativamente ao novo elemento. Segundo o nosso modelo a imagem 7(a) é muito semelhante a uma do elemento Guilherme 7(b), como podemos observar pelas seguinte imagem:

((a)) Imagem cuja previsão dá mal

((b)) Guilherme

Tentou-se observar se o resultado seria diferente com outras métricas mas obtemos o mesmo resultado em comparação com a normal. A distância de *mahalanobis* não foi uma boa métrica devido ao tipo de dados apresentado. Obteve-se distâncias a *NaN* em *python* que representam a operação de fazer a raiz de número negativos durante o cálculo de *mahalanobis*. Isto deve-se ao facto de haver uma grande diferença entre o número de amostras e atributos, mais concretamente, o número de amostras é muito mais pequeno que o número de atributos, o que causa problemas no cálculo da matriz de covariância dos dados, sendo esta singular e logo, não invertível, causando a presença dos valores negativos em raízes.

4 Conclusão

Durante a realização deste trabalho, o grupo aperfeiçoou mais um técnica poderosa de Machine Learning em ambientes não supervisionados, respetivamente, *PCA*. É uma técnica muito relevante, principalmente nos tempos que decorrem, pois os *datastes* são constituídos normalmente por centenas de atributos. É um bom algoritmo de redução de dimensão que nos permite ter as variáveis mais significativas do conjunto de dados. Para além disso ajuda na redução do *overfitting* e tem uma boa *performance*, neste caso, fomos capazes de implementar um sistema de reconhecimento facial para 5 indivíduos em menos de 1 segundo. Concluímos que, apesar de estarmos a remover informação sobre o dataset, conseguimos um bom desempenho e devido ao tempo não conseguimos, mas gostávamos de ter tentado utilizar um *dataset* maior, com mais variações entre imagens para podermos ver o resultado e se possível até aplicar redes neuronais para o processo de aprendizagem.

5 Bibliografia

Referências

[1] Dietmar Jannach, F.,Markus zanker, S., Alexander Felfering, Gerhard Friedrich, Recommender Systems, Cambrige.

[2] Yogesh Tayal, Pramod Kumar Pandey, D. B. V. Singh pp. 45–54. https://pdfs.semanticscholar.org/036a/13eaf042035c981bbec048b6d9ec21bd573b.pdf

[3] M.Turk, A. Pentland, EigenFaces for Recognition pp. 71–86. http://www.face-rec.org/algorithms/pca/jcn.pdf

[4] Cory Maklin, Towards Data Science, https://towardsdatascience.com/singular-value-decomposition-example-in-python-dab2507d85a0 Last accessed 8 Jan 2020

[5] https://towardsdatascience.com/tidying-up-with-pca-an-introduction-to-principal-components-analysis-f876599af383. Last accessed 5 Jan 2020

[6] https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8 Last accessed 7 Jan 2020

[7] https://intoli.com/blog/pca-and-svd/?fbclid=IwAR0atrrZwokCwR3Cg5EMFo5zDX0WSlS1p_W_cnVgWVoRGVf3TKX
Last accessed 2 Jan 2020

[8] https://medium.com/@jonathan_hui/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491?fbclid=IwAR187YuHO3-QuiZ6w7Qbd3Ld2zteP7ekM-CTz4hxONBRd9ggUM4JrnzDS0I Last accessed 29 Dez 2019