UNICAMP

Projeto Computacional 2 - Análise de Componentes Principais e Sistemas de Recomendação

Aprendizado de Máquinas: Aspectos Teóricos e Práticos

MS571 / MT571

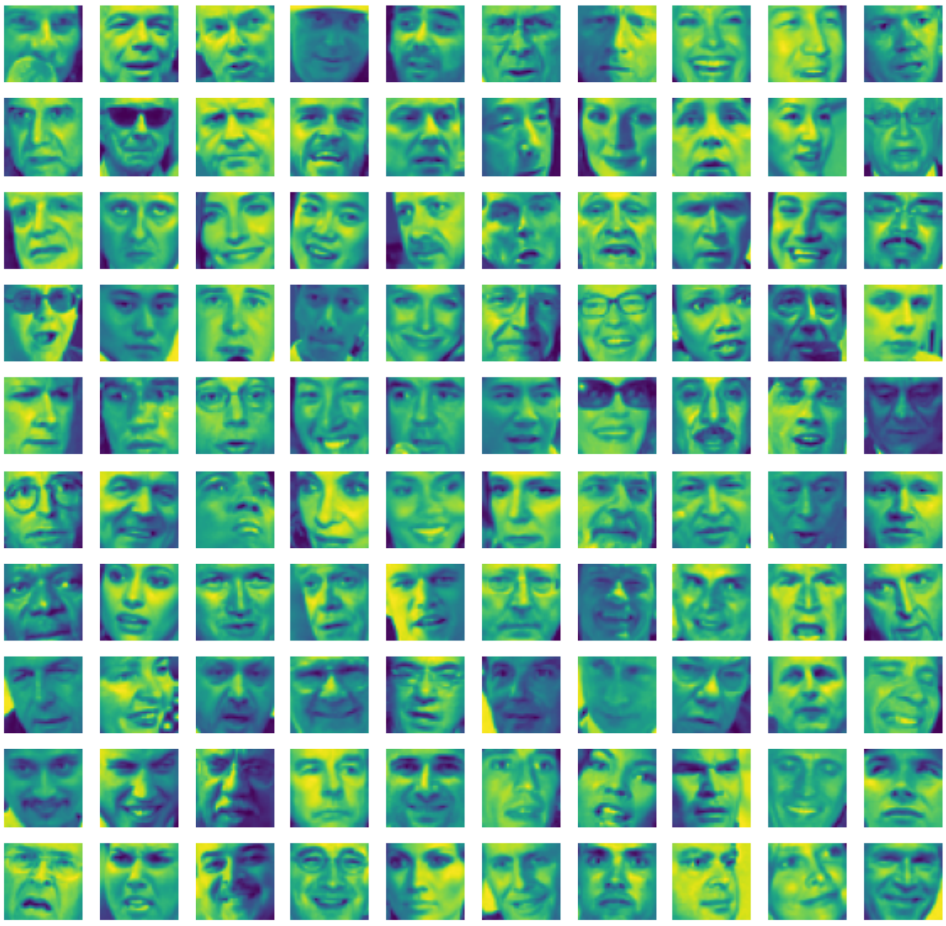
Amanda Rocha Faga - 212573

Caique Oliveira Alves da Silva - 168023

2° Sem/2022 - Campinas - SP

Parte I - Análise de Componentes Principais

Nessa primeira parte do projeto, nosso objetivo será resolver os exercícios propostos visando ter uma maior compreensão sobre o tópico de análise de componentes principais na aplicação de reconhecimento facial (eigenfaces). Ao final da Parte I, esperamos reconstruir imagens faciais diversas a partir de um pequeno grupo de treinamento, com boa precisão, visando reconhecer as imagens originais em uma rodada teste a partir das imagens reconstruídas finais. Veja a seguir 100 exemplos de imagens a serem reconhecidas, que neste caso serão usadas como nossa base de treinamento:



Para facilitar nossa implementação feita em Python, cada imagem com tamanho original 32×32 foi convertida em um vetor de 1024 componentes. Essas imagens foram carregadas a partir do arquivo *dados1.mat* através dos comandos: from scipy.io import loadmat ; dado1 = loadmat("dado1.mat") ; X = dado1["X"]. No total, possuímos 5000 imagens a serem classificadas, tendo uma matriz X contendo $m = 5000$ linhas e $n = 1024$ colunas. Além disso, foram usada as bibliotecas Pandas, Numpy, Scipy, Copy e Matplotlib.pyplot.

A partir dessa base de treinamento com 5000 imagens definidas na matriz X, podemos aplicar o método PCA (Análise de Componentes Principais), utilizando o seguinte algoritmo:

```
Sigma = (1/m)*X'*X;
U,S,V = svd(Sigma);
Ureduzida = U(:,1:k);
z = Ureduzida'*x;
```

Inicialmente, faremos uso de $k = 100$ componentes principais, e depois introduziremos a lógica indicada abaixo para encontrar um número ideal de k, que facilite futuramente nosso reconhecimento facial na rodada de testes.

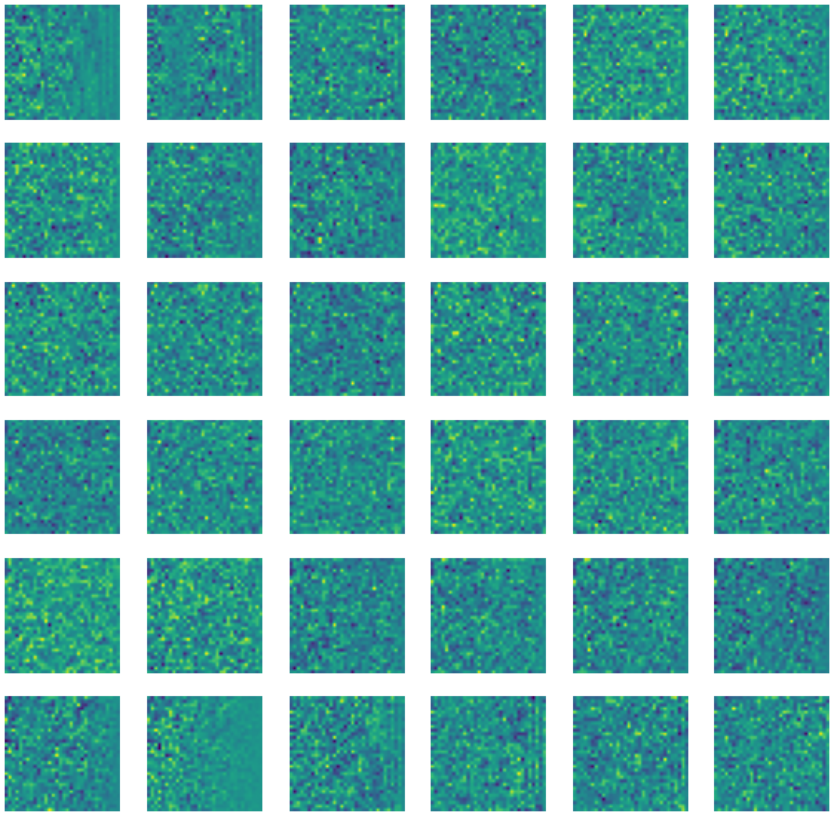
Testar PCA com $k = 1$

Calcular $U_{reduzida}, z^{(1)}, z^{(2)}, \cdots, z^{(m)}, x^{(1)}_{aproximado}, \cdots, x^{(m)}_{aproximado}$

Checar se $\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{aproximado}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$

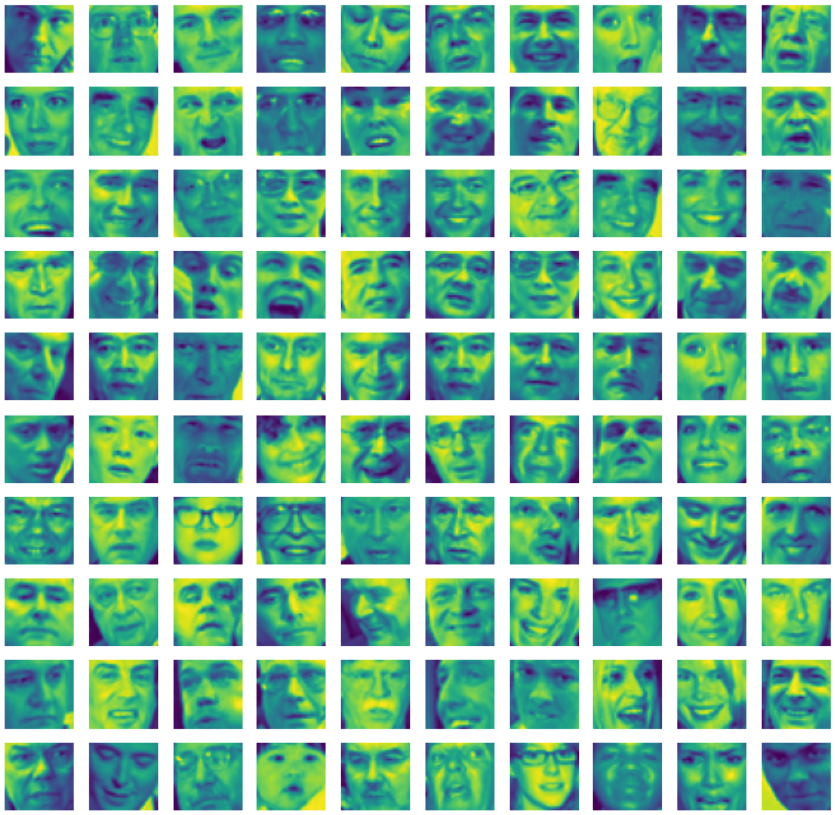
Repetir esse processo para $k = 2, 3, 4, \cdots$

Por enquanto, com $k = 100$, podemos reformatar os 36 primeiros componentes PCA (autovetores de Σ /eigenfaces) principais em 36 imagens de dimensões 32×32. Observe:

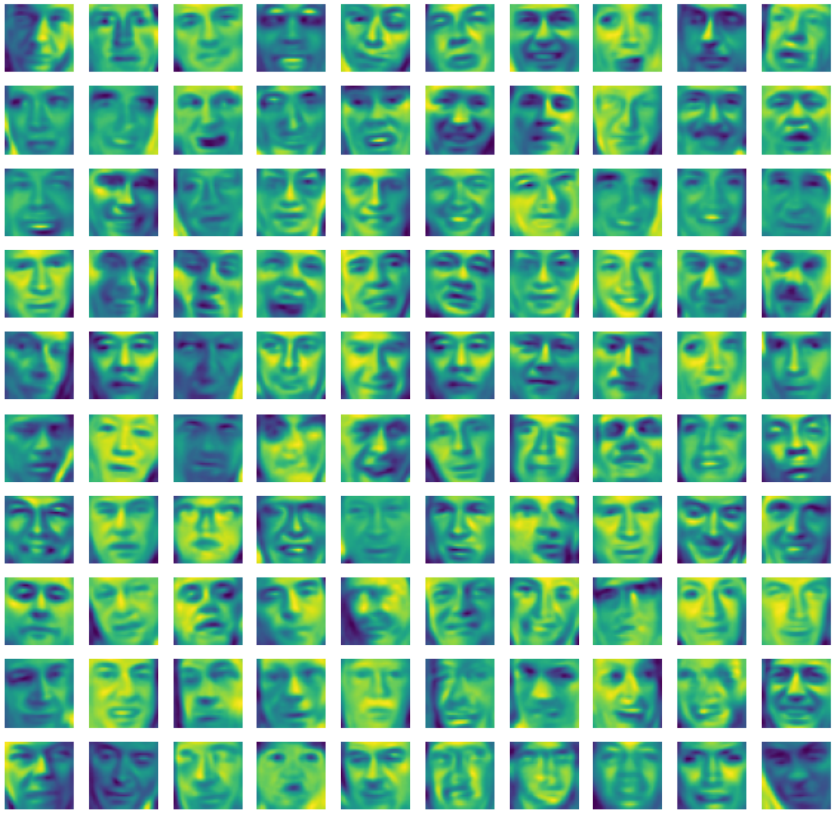


Apesar da imagem acima parecer não nos dizer muita coisa, é interessante ver como os autovetores de Σ influenciam em cada pixel das imagens, sendo uma espécie de peso no processo de reconstrução das faces feita através da matriz Z e da projeção da matriz X sobre esses componentes principais.

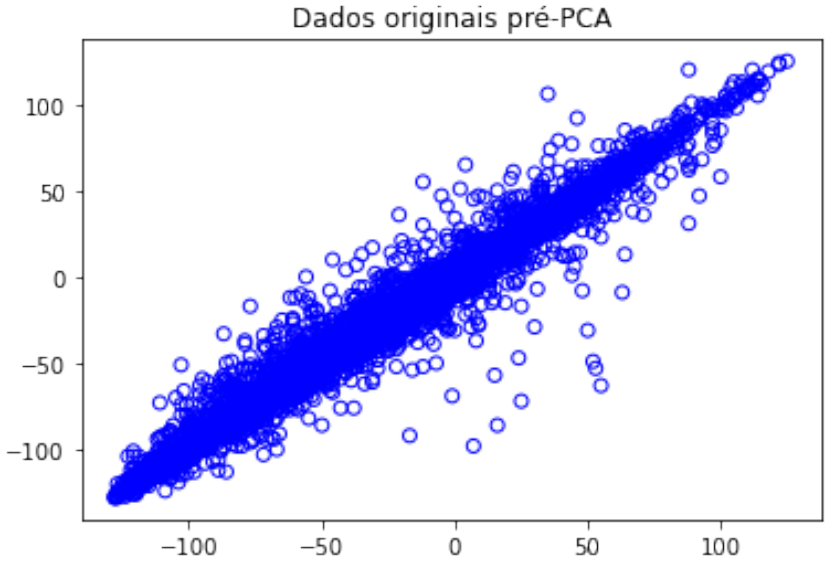
Agora, devemos projetar a matriz X sobre os 100 primeiros componentes principais, reconstruindo o dado original sobre esta projeção. A seguir, podemos observar 100 imagens aleatórias originais, e compará-las com suas reconstruções obtidas após o PCA:

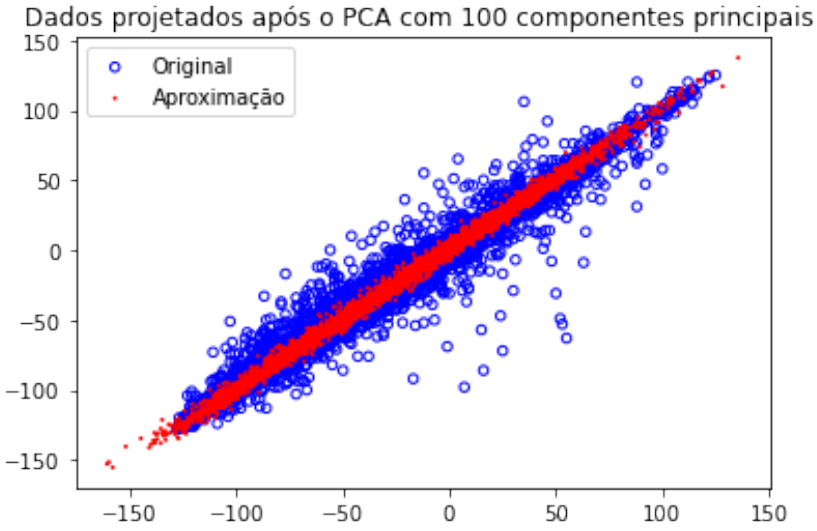


Imagens Originais pré-PCA

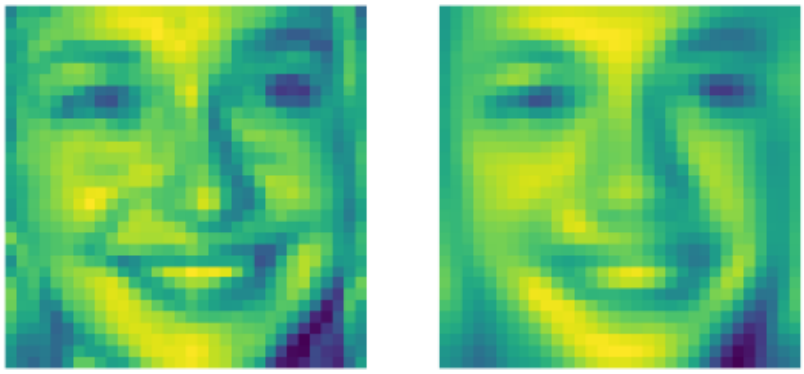


Imagens Reconstruídas após o PCA (K=100)

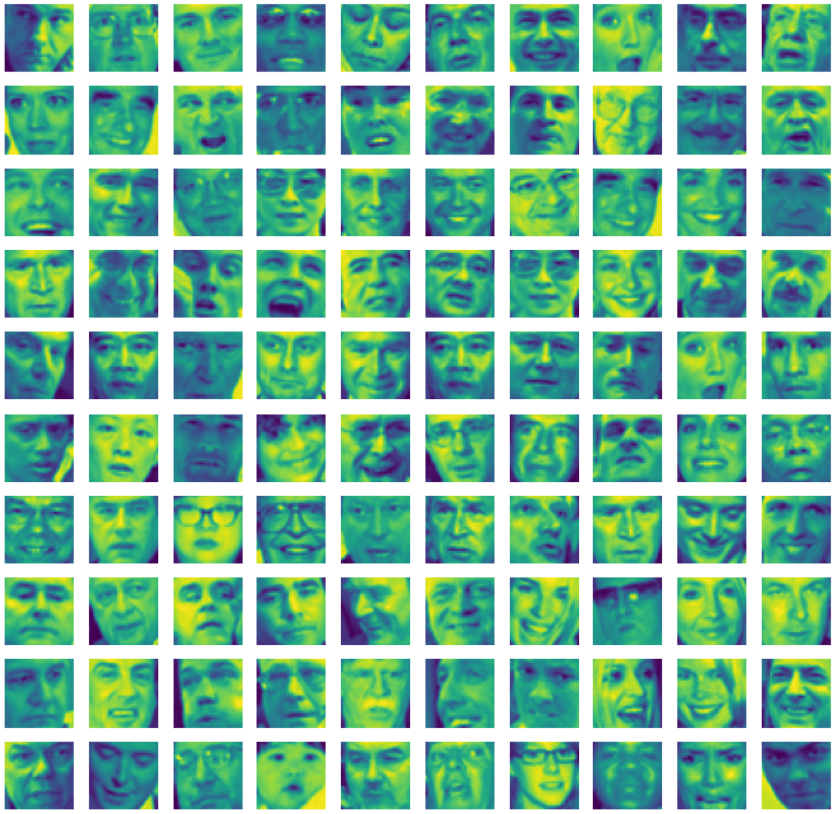




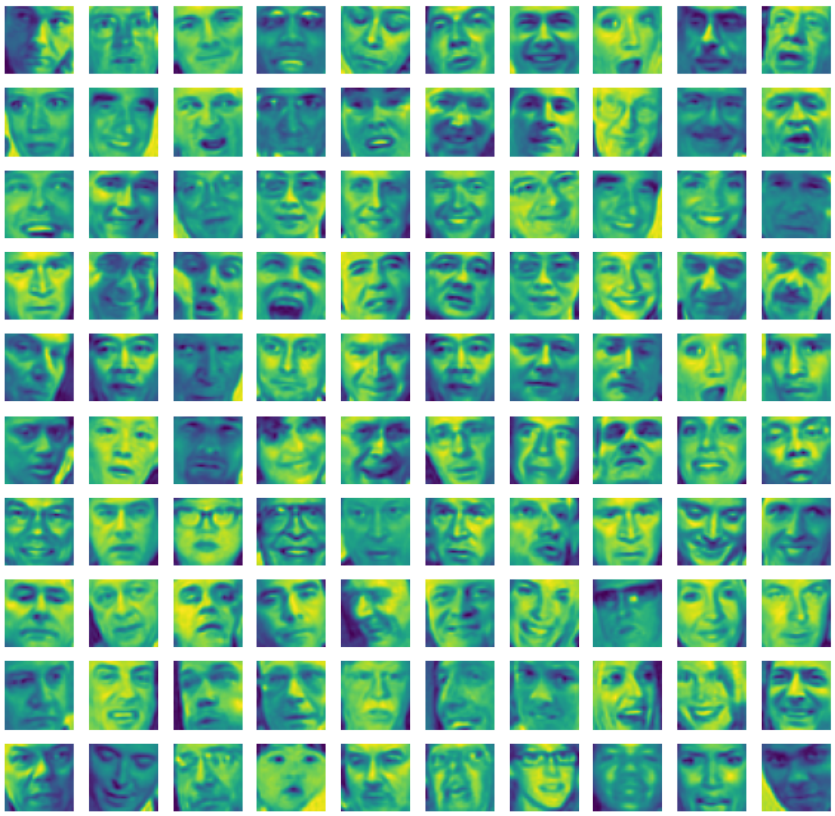
Repare no exemplo a seguir como podemos reconhecer facilmente a mesma pessoa comparando duas imagens equivalentes pré e pós-PCA:



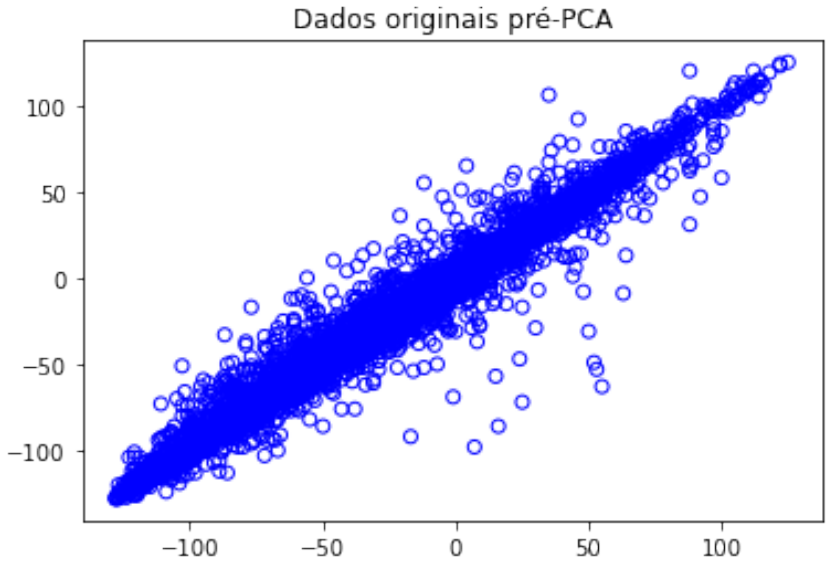
Isso ocorre devido ao baixo Erro Quadrático Médio ($\approx 5,66\%$) e à alta Variância Retida ($\approx 94,34\%$) no processo feito com 100 componentes principais. Entretanto, se desejarmos um erro ainda menor, podemos variar o número de componentes principais até obtermos o erro quadrático médio desejado. Variando o número de componentes principais a partir de K=300, procuramos um valor que nos fornecesse um erro $\leq 1\%$, e obtivemos os seguintes resultados:

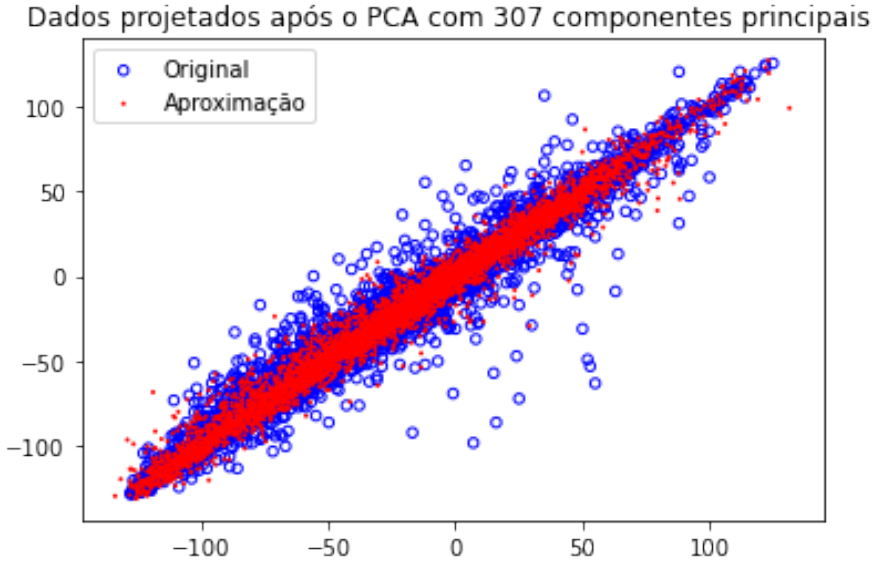


Imagens Originais pré-PCA



Imagens Reconstruídas após o PCA (K=307)





Observe que com K=307, um Erro quadrático médio de $\approx 0,997\%$ e uma variância retida de $\approx 99,1\%$, obtivemos faces reconstruídas ainda mais parecidas com as originais, quando comparadas às imagens reconstruídas com K=100.

Conclusão

Partindo dos resultados obtidos acima, podemos concluir que o PCA é um excelente método para reconstruir imagens a partir de uma base de fotos visualmente parecidas, ao ponto de conseguirmos reconhecer uma imagem original a partir do dado reconstruído, utilizando um número minimamente aceitável de componentes principais de acordo com o Erro Quadrático Médio que desejamos atingir, e vice-versa.

É claro que, nosso objetivo sempre será obter um erro baixo ($\leq$ 1, 5 ou 10%) para que seja possível visualizar claramente uma imagem reconstruída e que ela se pareça ao máximo com a original. Por isso mesmo, a Análise de Componentes Principais se mostrou um método extremamente eficiente especialmente no campo do reconhecimento facial.

Sobretudo, pudemos notar que esse método é capaz de minimizar o erro de projeção dos dados, preservando o máximo da variância original. Deste modo, ele também acaba se tornando um bom redutor de dimensionalidade, sendo um método recomendado para a compressão de dados em caso de necessidade reduzir o consumo de memória e acelerar algoritmos de aprendizado. Neste caso, aplica-se o PCA mapeando $x^{(i)} \rightarrow z^{(i)}$ sobre o conjunto de treinamento e calculando $U_{reduzida}$ e eventuais parâmetros de normalização sobre o conjunto de treino, e aplicando-os sobre o conjunto de teste.

Parte II - Sistemas de Recomendação

A segunda parte deste projeto consistiu no desenvolvimento de um sistema de recomendação de filmes baseado em filtragem colaborativa, que a partir de um banco de dados, faz a predição das notas de cada filme para cada um dos usuários, levando em consideração as notas dadas por eles nos filmes que assistiram e avaliaram.

(a) A implementação foi realizada em Python, com o auxílio das bibliotecas Numpy, Pandas, Matplotlib e Scipy. O banco de dados utilizado possui 943 usuários e 1682 filmes, e foi extraído do arquivo *dados2.mat*, por meio da função *loadmat* do *scipy.io*. Este arquivo contém duas matrizes: a matriz Y, que armazena na linha i e coluna j a nota de 0 a 5 dada pelo usuário j para o filme i, e a matriz R, que armazena na posição R(i,j) o dígito 1 se o usuário j deu alguma nota para o filme i e o dígito 0 caso contrário. Também foi carregado o arquivo *dados3.txt*, que carrega o nome e o ano de lançamento de cada filme.

O algoritmo de Machine Learning implementado se chama Low Rank Matrix Factorization, e consiste nas seguintes etapas:

Primeiramente, inicializou-se - com valores aleatórios e pequenos - duas matrizes X e Θ, contendo respectivamente os vetores de 100 atributos x para cada filme i, e os vetores de 100 parâmetros θ para cada usuário j.

Em seguida, com o objetivo de evitar que os usuários que não avaliaram um filme causem distorções na nota predita para ele, realizou-se o processo de normalização pela média da matriz Y, que consiste em:
1) definir um vetor Y_mean que armazena na linha i a nota média do filme i, dada pela divisão entre a soma das notas desse filme e o número de usuários que efetivamente o avaliaram.
2) remover o vetor Y_mean de cada coluna de Y.

Então foi definida a função de custo do problema, dada pela fórmula abaixo:

$$J(x^{(1)},\cdots,x^{(n_m)},\theta^{(1)},\cdots,\theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Como não foi utilizada regularização, o termo regularizador λ é igual a zero na expressão acima, de modo que ela se torna simplesmente:

$$J(x^{(1)},\cdots,x^{(n_m)},\theta^{(1)},\cdots,\theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2$$

Nota-se que o argumento da função de custo é um vetor com todos os atributos e parâmetros. No código elaborado, este vetor foi criado a partir das matrizes X e Θ, por meio do comando a seguir: *initial_params = np.append(X.flatten(),Theta.flatten())*.

Com base na função de custo do problema, foram definidas duas funções no código, sendo elas:

1) *cost_function*: calcula o custo e atualiza os parâmetros e atributos a cada iteração

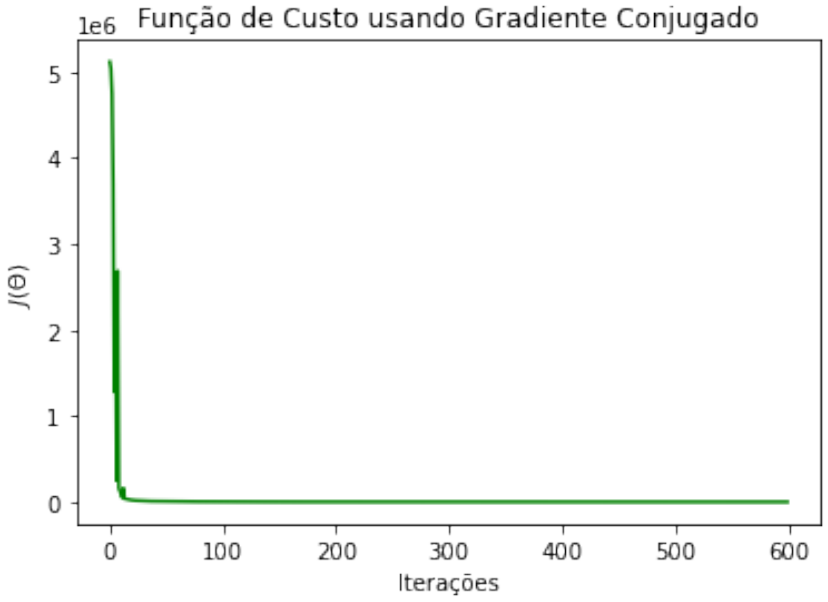
2) *cost_history*: executa uma iteração da *cost_function* e armazena o vetor *J_history*, contendo o seu histórico do custo.

O método de otimização utilizado para minimizar esta função de custo foi o gradiente conjugado, por meio da função *scipy.optimize.minimize*, configurada com os seguintes hiper-parâmetros:

1) Taxa de aprendizado $\alpha = 0.0001$

2) Número máximo de iterações = 400

Após o término da otimização, foi gerado o gráfico abaixo, que exibe o valor do custo em relação ao número de iterações:



A partir da saída da função *scipy.optimize.minimize*, obteve-se o vetor *final_params*,

com os novos valores calculados para os parâmetros e atributos. Com os dados deste vetor, atualizou-se as matrizes X e Θ e definiu-se a matriz de predição *predictions*, dada por $X\Theta^\top$, que armazena na linha i e coluna j a nota predita para o filme i com relação ao usuário j.

Por fim, somou-se a cada coluna da matriz *predictions* o vetor *Y_ mean* que havia sido subtraído durante a etapa de normalização pela média, finalizando assim a implementação do algoritmo. Dessa forma, os filmes recomendados para o usuário j serão aqueles correspondentes às linhas com os maiores valores na coluna j da matriz *predictions*.

(b) Para calcular a nota média para o filme i, calculou-se a soma das notas preditas para ele com relação a todos os usuários e dividiu-se o resultado pelo número de usuários. Foi criado um vetor com as notas médias denominado *predictions_ mean*, que armazena na linha i a nota média do filme i.

Como estas notas não estão no intervalo de 0 a 5, que é o escopo das notas originais, criou-se também um vetor denominado *notas*, que traz as notas médias normalizadas por *scaling* e multiplicadas por 5, estando assim no intervalo desejado. Em seguida, utilizou-se funções da biblioteca Pandas para retornar as notas médias com 1 casa decimal e os nomes dos dez filmes com notas médias mais altas. O resultado está apresentado na tabela abaixo:

Tabela 1: Filmes com as maiores notas médias

Posição	Filme	Nota média	Nota média normalizada
1	They Made Me a Criminal (1939	5.3	5.0
2	Aiqing wansui (1994)	5.3	5.0
3	Prefontaine (1997) (1994)	5.3	5.0
4	Entertaining Angels: The Dorothy Day Story (1996)	5.2	5.1
5	Great Day in Harlem, A (1994)	5.1	4.8
6	Someone Else's America (1995)	5.1	4.8
7	Santa with Muscles (1996)	4.1	4.8
8	Star Kid (1997)	5.0	4.7
9	Marlene Dietrich: Shadow and Light (1996)	4.9	4.6
10	Pather Panchali (1955)	4.8	4.5

Conclusão

Com base nos resultados obtidos, alguns pontos merecem ser destacados:

Com relação à minimização da função de custo, a análise do gráfico do custo em função do número de iterações indica que o método do gradiente conjugado se mostrou de fato uma técnica poderosa para este tipo de problema, visto que o custo convergiu em um número relativamente baixo de iterações.

Além disso, nota-se que as notas médias preditas para os filmes não retornaram um valor entre 0 e 5, entretanto, isto não é um problema para efeitos de recomendação, visto que neste caso apenas a ordem das notas preditas é necessária, independente do seu intervalo de valores.

Por fim, foi possível concluir que a filtragem colaborativa é um algoritmo que pode ser implementado de forma fácil e rápida, principalmente em comparação com algoritmos de filtragem por conteúdo, que no caso deste projeto, necessitariam, por exemplo, de especialistas definindo os atributos e seus valores para cada filme do banco de dados, o que seria uma tarefa inviável, devido à quantidade de filmes, e pouco efetiva, visto que haveria, inevitavelmente, um elevado nível de subjetividade envolvida.

Referências Bibliográficas

[1] Materiais (slides, notas de aula, exemplos de códigos) disponibilizados pelo professor João Batista Florindo.

[2] FLORINDO, João Batista. **Machine Learning**. YouTube. Disponível em: <https://www.youtube.com/playlist?list=PLGwGFVrptiyRmFoDWxruNGgTu2cSPnTLX>. Acesso em: 04 de dezembro de 2022.