

MS960 - Aprendizado de Máquinas
Aspectos Teóricos e Práticos

Prof. Dr. João Batista Florindo

**Projeto 3 - Análise de Componentes Principais
e Sistemas de Recomendação**

Bryan Alves do Prado	195171
Hugo Ricardo Ribeiro Matarozzi	155743
Isabella Mi Hyun Kim	170093

Instituto de Matemática, Estatística e Computação Científica (IMECC)
Universidade Estadual de Campinas (UNICAMP)
Brasil, Novembro 2021



Conteúdo

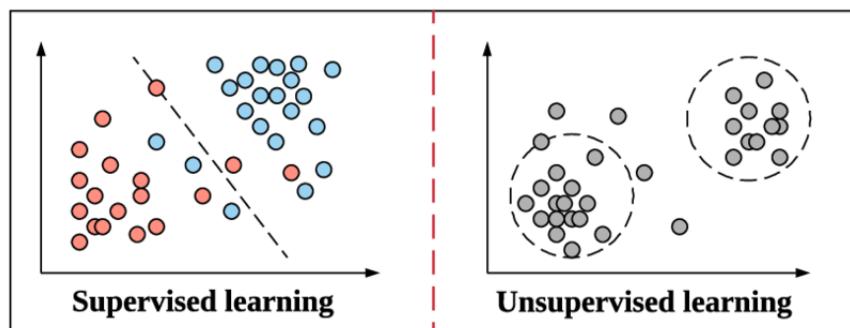
1	Introdução	2
2	Parte I - Análise de Componentes Principais	3
2.1	História e Ideia de Funcionamento:	3
2.2	Aprofundando a Teoria: Decomposição SVD	4
2.3	Algoritmo em Python para a Parte I	5
2.4	Primeiro Problema da Primeira Parte:	9
2.5	Segundo Problema da Primeira Parte:	10
2.6	Terceiro Problema da Primeira Parte:	12
2.7	Uma Visualização Interessante:	14
3	Parte II - Sistemas de Recomendação	16
3.1	História e Ideia de Funcionamento:	16
3.2	Aprofundando a Teoria: Algoritmo Para Recomendação	18
3.3	Algoritmo em Python Para a Parte II:	18
3.4	Primeiro e Segundo Problemas da Segunda Parte:	24
3.5	Busca Por Filmes:	32
3.6	Processo de Salvamento de Dados do Modelo:	35
4	Conclusão	36
5	Bibliografia:	36

1 Introdução

Nos relatórios anteriores foram considerados algoritmos de regressão e classificação, por funções hipótese lineares, não lineares e por meio de redes neurais. Ambos são exemplos de algoritmos voltados à aprendizagem supervisionada, onde os dados de teste contém o “valor verdadeiro”, *ground truth*, como par ordenado para treinamento do modelo.

Por mais que este seja o tipo de problema mais pesquisado e, consequentemente, com mais algoritmos desenvolvidos, há também o problema de aprendizado não-supervisionado, que será o considerado neste relatório.

O aprendizado não-supervisionado, por não ter disponíveis os valores a serem preditos nos dados de treino em forma de par ordenado, visam capturar estrutura nos dados ao invés de gerar previsões. Um exemplo simples de problema de aprendizado não-supervisionado é o de agrupamento (*clustering*), onde o objetivo é encontrar estruturas que permitam a separação dos dados em grupos. É relevante apontar que o número de grupos não é dado previamente, diferente de uma classificação multi-classes.



Exemplo de aprendizado supervisionado (classificação) e não-supervisionado (agrupamento).¹

As áreas onde os problemas resolvidos por aprendizado não-supervisionado surgem são muito diversas, muitas vezes no mercado: marketing, redes sociais, segurança pública, vendas e serviços online, clusters de computadores, astronomia, entre outros. Alguns exemplos de problemas e algoritmos são: agrupamento, compressão de dados, detecção de anomalias e sistemas de recomendação.

Neste relatório serão considerados o problema de compressão de dados, que tem como algoritmo a Análise de Componentes Principais (PCA); e o problema de sistemas de recomendação.

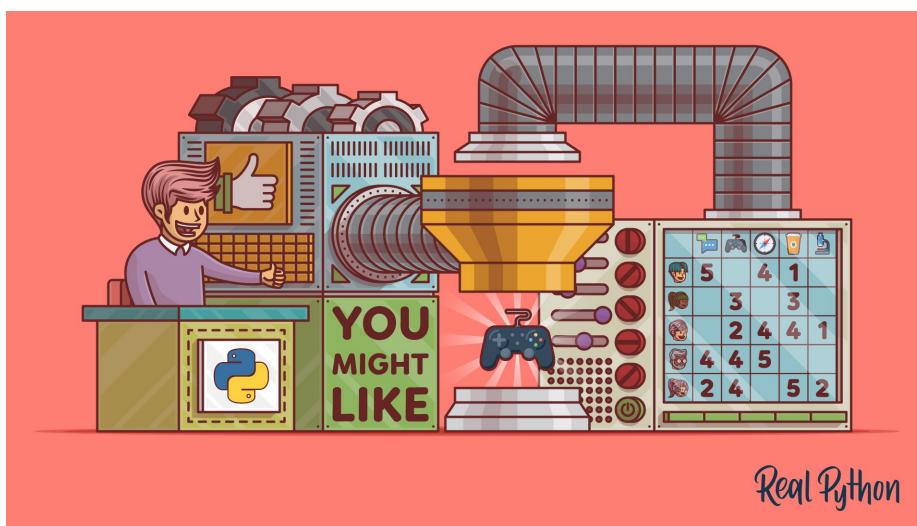


Ilustração de um sistema de recomendação.²

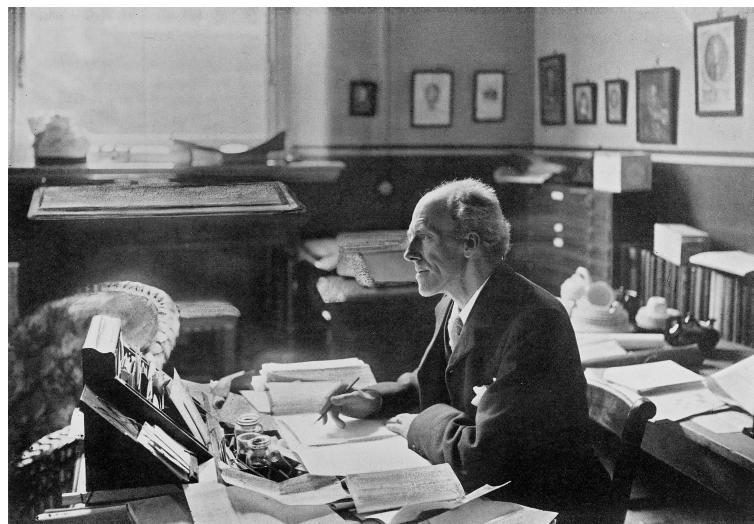
¹Imagem retirada de https://www.researchgate.net/figure/Examples-of-Supervised-Learning-Linear-Regression-and-Unsupervised-Learning_fig3.336642133

²Imagem retirada de <https://realpython.com/build-recommendation-engine-collaborative-filtering/>

2 Parte I - Análise de Componentes Principais

2.1 História e Ideia de Funcionamento:

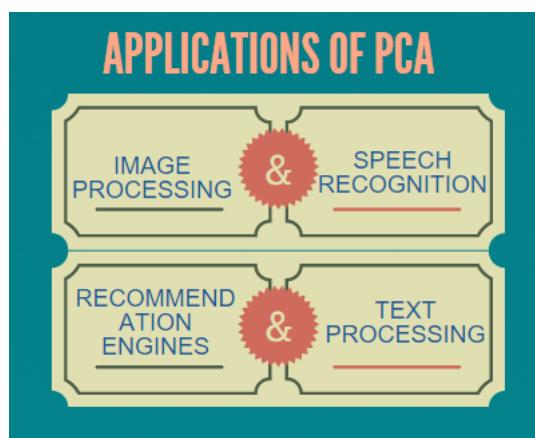
Inventado em 1901 pelo matemático e estatístico britânico Karl Pearson, o PCA (Principal Component Analysis) consiste em um procedimento matemático que utiliza uma transformação ortogonal, a ortogonalização de vetores, para a conversão de um conjunto de variáveis potencialmente correlacionadas em um conjunto de variáveis linearmente não correlacionadas, denominadas “componentes principais”, sendo que o número de componentes principais é sempre menor ou igual ao de variáveis originais. É possível se referir a este método como “Transformada de Karhunen-Loève discreta”, “Transformada de Hotelling” ou “Decomposição ortogonal própria”.



Karl Pearson, inventor do PCA, em 1910³

Sendo a mais simples das verdadeiras análises multivariadas por autovetores, o PCA é comumente e extensamente utilizado na análise exploratória de dados e para a construção de modelos preditivos.

Entre os tópicos possíveis de aplicação de tal procedimento podemos citar a neurociência, principalmente para a identificação de propriedades específicas de estímulos que aumentam a probabilidade de um neurônio gerar uma potencial ação; finanças quantitativas, diminuindo problemas complexos através da seleção de seus pontos “pilares”; a compressão de imagens, mantendo as características principais das mesmas; e por fim, o reconhecimento facial, aplicação que será abordada neste projeto.



Alguns exemplos de aplicação do PCA⁴

³Imagem retirada de https://commons.wikimedia.org/wiki/File:Karl_Pearson,_Biometrist,_at_his_desk,_1910_Wellcome_M0011713.jpg

⁴Imagen retirada de <https://www.r-bloggers.com/2016/02/principal-component-analysis-using-r/>

2.2 Aprofundando a Teoria: Decomposição SVD

Ao realizar a Análise de Componentes Principais, recorremos a uma rotina já implementada no `numpy.linalg` que realiza a decomposição SVD da matriz de covariância empírica $\text{cov}(X) = X^T X$, onde X é a matriz referente aos dados de nosso problema, já normalizada.

Aprofundamos um pouco mais a teoria em relação a essa decomposição SVD. O nome é um acrônimo para *Singular Value Decomposition*, ou seja, decomposição em valores singulares, e surge ao, dada matriz $A \in \mathbb{R}^{m \times n}$, buscar uma base ortonormal para o espaço gerado tanto pelas colunas quanto pelas linhas de A .

Seja $A \in \mathbb{R}^{m \times n}$ matriz de posto p , e u_1, u_2, \dots, u_p base ortonormal do espaço gerado pelas linhas de A . Podemos então denotar:

$$AV = U\Sigma$$

$$= (u_1 \ u_2 \ \dots \ u_p) \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_p \end{pmatrix}$$

Para considerarmos também o núcleo de A , adicionamos $m-p$ vetores u_i ortonormais e valores $\sigma_{p+1} = \sigma_{p+2} = \dots = \sigma_m = 0$ de forma a obter:

$$AV = U\Sigma$$

$$= (u_1 \ u_2 \ \dots \ u_m) \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \end{pmatrix}$$

com U matriz ortogonal cujas colunas são base de \mathbb{R}^m . Dessa maneira, $V \in \mathbb{R}^n$ é arbitrária e podemos tomar matriz cujas colunas v_i são ortonormais, i.e., V é ortogonal. Como consequência, deduzimos:

$$AV = U\Sigma$$

$$\Rightarrow A = AVV^T$$

$$= U\Sigma V^T$$

$$= (u_1 \ u_2 \ \dots \ u_m) \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_m^T \end{pmatrix}$$

o que constrói nossa decomposição SVD.

Para encontrarmos os vetores u_i e v_i e as componentes principais σ_i , buscamos resolver os problemas de autovalores e autovetores relacionados às matrizes $A^T A$ e AA^T . Como são matrizes reais e simétricas, seus autovalores são reais e temos:

$$A^T A = (V\Sigma^T U^T)(U\Sigma V^T)$$

$$= V\Sigma^T \Sigma V^T$$

$$= (v_1 \ v_2 \ \dots \ v_n) \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{pmatrix}$$

e

$$\begin{aligned}
AA^T &= (U\Sigma V^T)(V\Sigma^T U^T) \\
&= U\Sigma\Sigma^T U^T \\
&= \begin{pmatrix} u_1 & u_2 & \dots & u_m \end{pmatrix} \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_m^2 \end{pmatrix} \begin{pmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_m^T \end{pmatrix}
\end{aligned}$$

Para resolvê-los, basta considerar:

$$\begin{aligned}
A^T A v_i &= V \Sigma^T \Sigma V^T v_i \\
&= V \Sigma^T \Sigma e_i \\
&= V \sigma_i^2 e_i \\
&= \sigma_i^2 v_i
\end{aligned}$$

e analogamente para $AA^T u_i$, o que nos permite concluir que os autovetores de $A^T A$ e AA^T são v_i e u_i respectivamente, ambos referentes ao autovalor σ_i^2 . Sendo assim, resolvendo o problema de autovalores e autovetores encontramos a decomposição desejada. Usualmente tomamos o valor positivo $\sigma_i = |\sigma_i|$, de forma a termos todos os valores singulares positivos ou nulos.⁵

Ao ordenar as matrizes U , Σ e V de forma decrescente por valores singulares, obtemos a conexão com o PCA: as componentes principais são as colunas de V , pois são os autovetores de $A^T A$, a matriz de covariância, e os seus pesos, i.e., suas contribuições são dados pelo quadrado do valor singular σ_i .

2.3 Algoritmo em Python para a Parte I

Para a primeira parte do projeto, desenvolvemos um algoritmo que agrupa todas as funções que serão utilizadas para resolver os 3 problemas propostos (a, b e c), permitindo também ao usuário uma grande interação e definição de parâmetros para o código, gerando um programa que possibilita uma maior quantidade de análises e resoluções. A seguir, análises voltadas para cada função presente no programa:

```

1 | def Load(g):
2 |     dado1=loadmat(g)
3 |     X=dado1.get('X')
4 |     return X

```

Primeiramente, temos a função *Load*, responsável por carregar os dados de forma adequada e retornar os mesmos de maneira a serem utilizados futuramente no programa.

```

1 | def feature_normalize(X):
2 |     mu = np.mean(X, axis=0)
3 |     sigma = np.std(X, axis=0)
4 |
5 |     X_norm = (X-mu)/sigma
6 |
7 |     return X_norm, mu, sigma

```

Em seguida, temos a função *feature_normalize*, responsável por normalizar as colunas da matriz X de dados, permitindo assim uma melhor manipulação e utilização da mesma e possibilitando a implementação do algoritmo PCA.

```

1 | def PCA(X):
2 |     m,n=X.shape
3 |     Sigma=(1/m)*(X.T @ X)
4 |     U, S, V = svd(Sigma)
5 |     return U, S, V

```

A função *PCA* é responsável por realizar a decomposição *SVD* da matriz de covariância $X^T X$.

⁵Demonstração baseada no material de MIT OpenCourseWare, Linear Algebra, 2011. Acesso por: https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/positive-definite-matrices-and-applications/singular-value-decomposition/MIT18_06SCF11_Ses3.5sum.pdf

```

1 | def show_img(img_arr):
2 |     pixels = img_arr.reshape((32, 32))
3 |     plt.imshow(pixels.T, cmap='gray')
4 |     plt.xticks([])
5 |     plt.yticks([])

6 |
7 |
8 | def show_img_grid(data, n_imgs=100, grid_v=10, grid_h=10, img_size=15,
9 | randomic=True, inverse_order=False):
10 |     fig = plt.figure(figsize=(img_size, img_size))
11 |     fig.subplots_adjust(hspace=0, wspace=0)

12 |
13 |     for i in range(n_imgs):
14 |         ax = fig.add_subplot(grid_v, grid_h, i+1)
15 |         num = i
16 |         if inverse_order:
17 |             num = data.shape[1] - i - 1
18 |         if randomic:
19 |             num = randint(0, len(data) - 1)
20 |         show_img(data[num])

```

A seguir, temos as funções *show_img* e *show_img_grid*, responsáveis pelos plots das imagens do programa, sendo que, a primeira é responsável pelo plot direto e a segunda é responsável pela organização da imagem em um “grid” de forma a facilitar a visualização.

```

1 | def eigenfaces_video(V, lim, y):
2 |     y=y+"\eigen.gif"
3 |     fig, ax = plt.subplots(figsize=(10,10))
4 |     ax.set_ylim(0, 30)
5 |     line, = ax.plot(0, 0)

6 |
7 |     def animation_frame(i):
8 |         img_arr = np.zeros(1024)
9 |         for j in range(i):
10 |             img_arr = img_arr + V[j, :]
11 |             pixels = img_arr.reshape((32, 32))
12 |             plt.imshow(pixels.T, cmap='gray')
13 |             plt.xticks([])
14 |             plt.yticks([])
15 |             plt.title(f'Eigenface {i+1} - {i+1}\'')

16 |
17 |         ax.invert_yaxis()

18 |
19 |     animation = FuncAnimation(fig, func=animation_frame,
20 |     frames=np.arange(0, lim, 1), interval=10)
21 |     animation.save(y)

```

Com a função *eigenfaces_video*, conseguimos criar um gif que mostra a variação da soma a cada componente principal. Tal gif é armazenado em um diretório oferecido pelo usuário (Não é necessário se oferecer o nome do arquivo).

```

1 | def project_pca(U, img_array, n, show=False):
2 |     img_alt = U[:, :n] @ U[:, :n].T @ img_array.T
3 |     if show:
4 |         show_img(img_alt)
5 |
6 |     return img_alt

```

Tal função *project_pca* é responsável pela projeção da imagem sobre as n primeiras componentes principais, com isso, conseguimos comparar a imagem reconstruída com a original obtida pelos dados oferecidos.

```

1 | def sbs_img_plot(img_array_1, img_array_2):
2 |     fig = plt.figure(figsize=(25,25))
3 |     fig.subplots_adjust(hspace=0, wspace=0)

4 |
5 |     ax = fig.add_subplot(5, 5, 1)
6 |     show_img(img_array_1)

```

```

7     plt.title('Reconstruida')
8
9     ax = fig.add_subplot(5, 5, 2)
10    show_img(img_array_2)
11    plt.title('Original')

```

Utilizamos a função *sbs_img_plot* para plotarmos as imagens reconstruídas e originais lado a lado, permitindo assim ver como o algoritmo “tratou” as mesmas.

```

1 def proj_evolution_video(U, img_array, lim, step,j):
2     j=j+'\proj.gif'
3     fig = plt.figure(figsize=(15,15))
4     fig.subplots_adjust(hspace=10, wspace=0)
5     def animation_frame(i):
6         '''
7             Creates frame for animation
8         '''
9         ax = fig.add_subplot(1,2, 1)
10        show_img(project_pca(U, img_array, i))
11        plt.title(f'Reconstruida, {i} componentes principais',
12                  fontsize=20)
13
14
15        ax = fig.add_subplot(1,2, 2)
16        show_img(img_array)
17        plt.title('Original', fontsize=20)
18
19
20    animation = FuncAnimation(fig, func=animation_frame,
21    frames=np.arange(0,lim,step), interval=10)
22
23    Path("images").mkdir(parents=True, exist_ok=True)
24    animation.save(j)

```

Semelhante ao que fizemos anteriormente para as eigenfaces, a função *proj_evolution_video* cria e armazena um gif que mostra a evolução da imagem reconstruída em comparação com a imagem original.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4 from pathlib import Path
5 from scipy.io import loadmat
6 from random import randint
7 from numpy.linalg import svd

```

Acima, as bibliotecas e funções prontas que importamos para utilizar no programa.

```

1 g=input("\nEndereço da pasta onde se encontram os dados
2 (sem o nome do arquivo): ")
3 g=g+'\dados'
4 X=load(g)
5 X_norm ,mu,sigma=feature_normalize(X)
6 a=int(input('Gostaria de visualizar uma amostra aleatoria de imagens?
7 1 para sim e 2 para nao: '))
8 if a==1:
9     b=int(input('Qual o numero de imagens desejado? '))
10    show_img_grid(X, b, int(np.ceil(b**1/2))), int(np.ceil(b**1/2))),
11    15)
12 U,S,V=PCA(X_norm)
13 c=int(input('\nGostaria de ver as eigenfaces correspondentes aos
14 n primeiros componentes principais? 1 para sim e 2 para nao: '))
15 if c==1:
16     d=int(input('\nQual o valor de n? '))
17     show_img_grid(V, d, int(np.ceil(d**1/2))), int(np.ceil(d**1/2))),
18     15, randomic=False)
19     e=int(input('\nGostaria de ver as eigenfaces correspondentes aos n
20 ultimos componentes principais? 1 para sim e 2 para nao: '))
21     if e==1:

```

```

22     show_img_grid(V, d, int(np.ceil(d**1/2))),  

23     int(np.ceil(d**1/2)))  

24     , 15, randomic=False, inverse_order=True)  

25 f=int(input('\nGostaria de salvar o gif com as eigenfaces  

26 correspondentes aos n primeiros componentes principais somadas?  

27 1 para sim e 2 para nao: '))
28 if f==1:  

29     y=input('\nEndereco para se salvar o gif (Sem o nome do arquivo)  

30      : ')
31     eigenfaces_video(V, d,y)
32 z=int(input("\nGostaria de visualizar as imagens reconstruidas e  

33 originais lado a lado? 1 para sim e 2 para nao: "))
34 if z==1:  

35     h=int(input("\nQual seria o numero de imagens? "))
36     n=int(input("\nQual seria o numero de componentes principais? "))
37     for i in range(h):
38         num = randint(0, len(X) - 1)
39         sbs_img_plot(project_pca(U, X[num], n), X[num])
40 u=int(input("\nGostaria de salvar o gif com a evolucao da  

41 reconstrucao? 1 para sim e 2 para nao: "))
42 if u==1:  

43     k=int(input("\nQual a quantidade de componentes principais?  

44 (Multiplo de 5): "))
45     j=input('\nEndereco para se salvar o gif (Sem o nome do arquivo): ')
46     proj_evolution_video(U, X[randint(0, len(X) - 1)], k, 5,j)

```

Por fim, nossa “central de comando”, onde definimos os parâmetros que serão ofertados para cada função, os inputs que serão realizados pelo usuário e entre outros.

Assim, com tais funções aqui apresentadas, conseguimos desenvolver a proposta da primeira parte de forma plena e satisfatória, abarcando todos os pontos citados e adicionando alguns extras, como a possibilidade de visualizar um número qualquer de imagens, salvar gifs que facilitam na compreensão do funcionamento do algoritmo, entre outros.

A seguir, um exemplo de execução do código na IDE Spyder:



Exemplo na IDE Spyder

2.4 Primeiro Problema da Primeira Parte:

Para esse problema, começamos carregando os dados através da já citada função *Load*. Desta forma, obtemos uma matriz X com 5000 linhas e 1024 colunas, onde cada linha é uma imagem de dimensão 32×32 transformada em vetor.

Com essas prerrogativas em mente, desenvolvemos as funções *show_img* e *show_img_grid* que reconvertem os vetores para imagens e as organizam dentro de um “grid” que tem como dimensões “x” e “y” o inteiro obtido ao arredondarmos “para cima” a raiz quadrada da quantidade de imagens oferecida pelo usuário (de forma a ter um “grid” quadrado), o que foi possível utilizando a função *np.ceil* do Numpy. Com isso, para um exemplo de 100 imagens aleatórias, obtivemos o seguinte resultado:



Exemplo de resultado com 100 imagens aleatórias

Como podemos visualizar, a implementação atendeu a ideia proposta de forma satisfatória. Como fizemos um algoritmo interativo, é possível plotar outras quantidades como, por exemplo, de 4 imagens:



Exemplo de resultado com 4 imagens aleatórias

Em seguida, implementamos a função *feature_normalize*, utilizada para a normalização das colunas da matriz X , visando a aplicação da função *PCA*. Com êxito, obtemos as matrizes U , S e V utilizadas em outros trechos da primeira parte do projeto.

2.5 Segundo Problema da Primeira Parte:

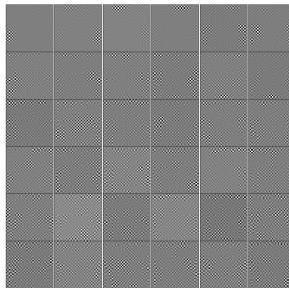
Para este segundo problema, utilizamos as mesmas funções `show_img` e `show_img_grid` para plotarmos as “eigenfaces” a partir da matriz V obtida a partir da função *PCA*. Para o caso exemplo proposto no problema (eigenfaces correspondentes aos 36 primeiros componentes principais), obtemos o seguinte resultado:



Exemplo de resultado com eigenfaces correspondentes aos 36 primeiros componentes principais

Como é possível visualizar, essas 36 eigenfaces representam os 36 autovetores de $X^T X$ com os maiores autovalores, ou seja, representam as características que dão maior variância entre os dados da matriz X . Elas demonstram as características mais relevantes na formação das imagens das faces, como seu formato, formato dos olhos, da boca, nariz, sombrancelhas, entre outros. Ou seja, são as características principais que compõem os rostos presentes na matriz X .

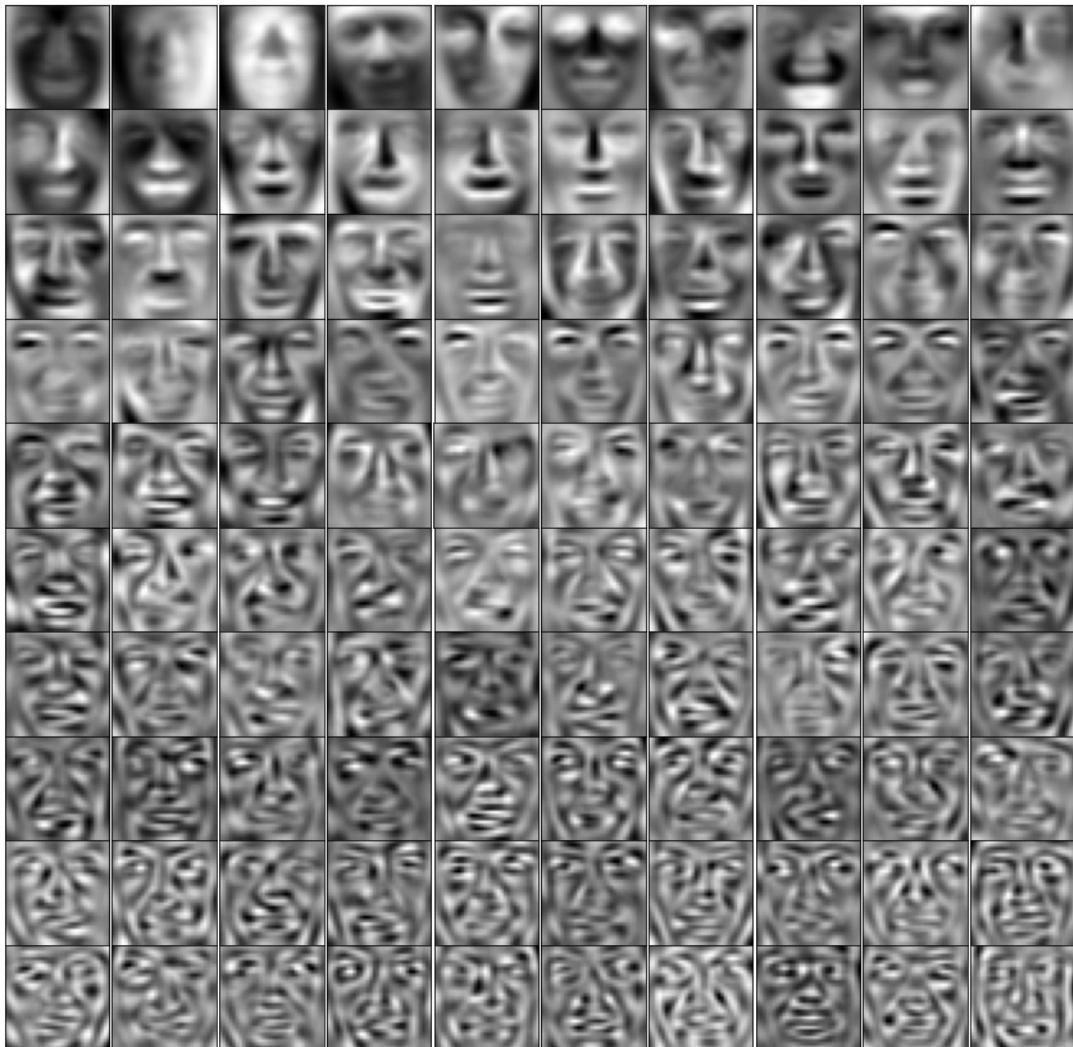
Em compensação, se tomarmos as eigenfaces correspondentes aos 36 últimos componentes principais, obtemos o seguinte resultado:



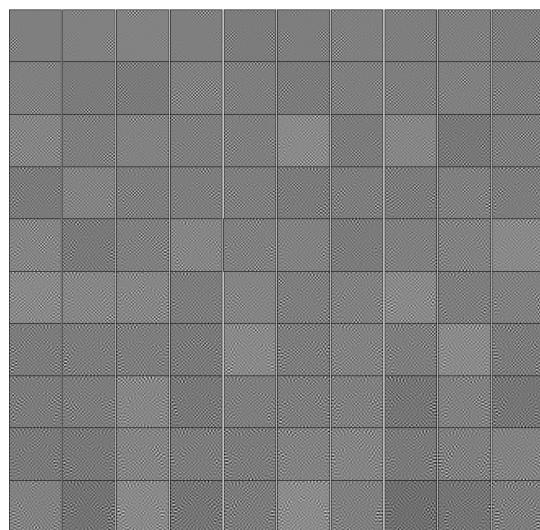
Exemplo de resultado com eigenfaces correspondentes aos 36 últimos componentes principais

Como é possível visualizar, são componentes que não representam características importantes ou relevantes de nossas imagens.

Temos também um exemplo com as eigenfaces correspondentes aos 100 primeiros componentes principais:



Exemplo de resultado com eigenfaces correspondentes aos 100 últimos componentes principais
E as eigenfaces correspondentes aos 100 últimos componentes principais:



Exemplo de resultado com eigenfaces correspondentes aos 100 últimos componentes principais

Importante ressaltar que desenvolvemos uma função, a *eigenfaces_video*, que permite visualizar um gif que mostra a variação na soma dos n primeiros componentes principais (de forma simplificada, claro, sem considerar pesos distintos e entre outros). Sua execução pode ser a partir do Jupyter Notebook ou do código original.

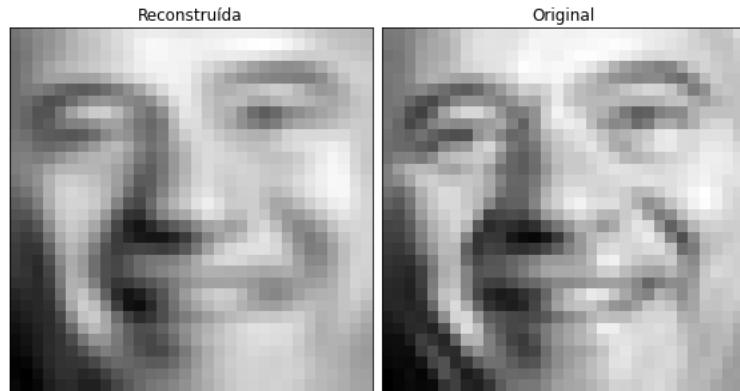
2.6 Terceiro Problema da Primeira Parte:

Para a resolução deste problema, construímos a função *project_pca*, responsável por projetar a imagem sobre as n primeiras componentes principais. Um exemplo de seu resultado de saída com $n = 100$ é o seguinte:



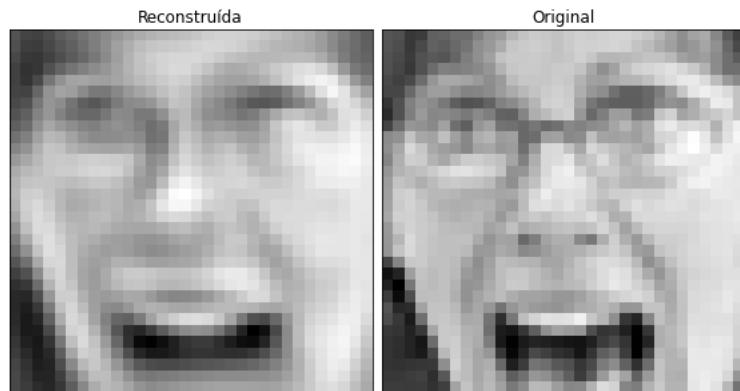
Exemplo de resultado com projeção de imagem sobre as 100 primeiras componentes principais

A partir disso, usamos a função *sbs_img_plot* para realizarmos o plot da imagem reconstruída e da original lado a lado. Para um caso com $n=100$, temos o seguinte exemplo:



Exemplo de resultado com projeção de imagem sobre as 100 primeiras componentes principais lado a lado com a imagem original

A reconstrução permite diferenciar faces, mas não de identificar, pelo menos visualmente a olho humano, sua maioria. Posições da cabeça e de partes do rosto fora da média geram distorções nas imagens, e detalhes como linhas de expressão, posições da íris, acessórios, entre outros, muitas vezes não estão claros na reconstrução. Um exemplo interessante para a questão da identificação ou não de acessórios é o seguinte:



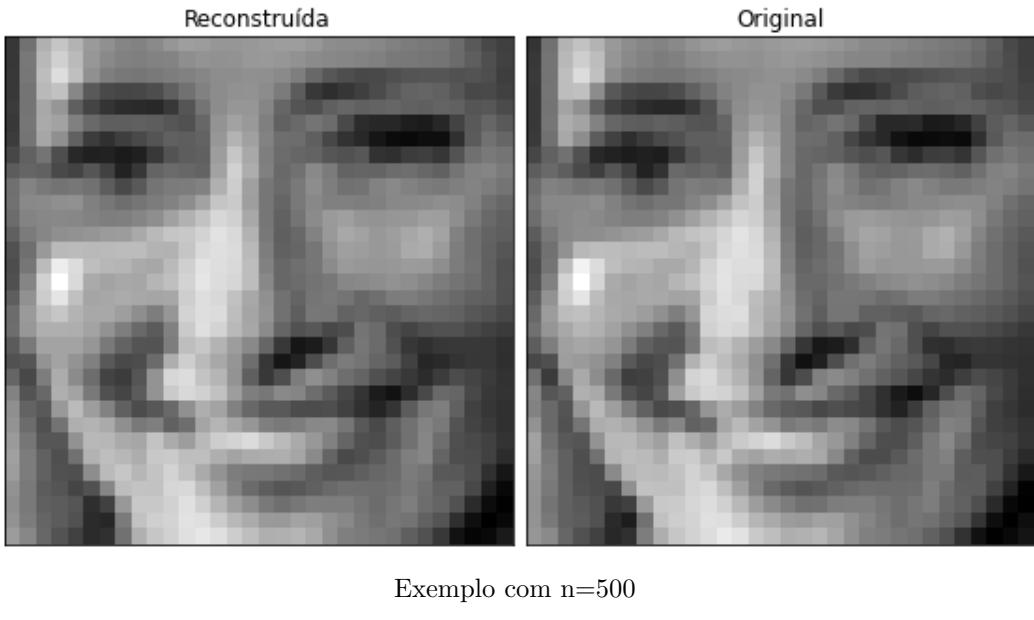
Exemplo de ausência de acessório na imagem reconstruída ($n=100$)

Como é possível observar, na imagem original é perceptível o uso de óculos por parte da pessoa na imagem, porém, tal elemento foi perdido na versão reconstruída.

Dessa maneira, aplicações interessantes para este tipo de reconstrução são em áreas onde é relevante identificar que certa imagem é de um humano, e seja necessário diferenciar uma pessoa de outra, mas não é relevante identificá-la. Um exemplo de situação é a de carros autônomos:

é importante identificar onde estão pedestres para que uma rota segura seja tomada, mas não é primordial identificar cada pessoa. É, aliás, importante que o processo seja ágil de forma a mitigar acidentes (menor tempo de reação), o que justifica uma possível diminuição de detalhes das faces em prol de uma maior agilidade na identificação e tomada de decisões. Outro exemplo onde a identificação da presença ou não de indivíduos é importante é em alarmes de segurança que são ativados quando identificam alguma presença humana, a agilidade nesse processo é algo também extremamente fundamental.

Também realizamos testes para diferentes valores de n , como por exemplo, $n = 500$:



Para esse exemplo conseguimos observar uma maior nitidez em relação aos exemplos anteriores quanto a detalhes; são visíveis linhas de expressões e entre outros.

Um exemplo com $n=2$:



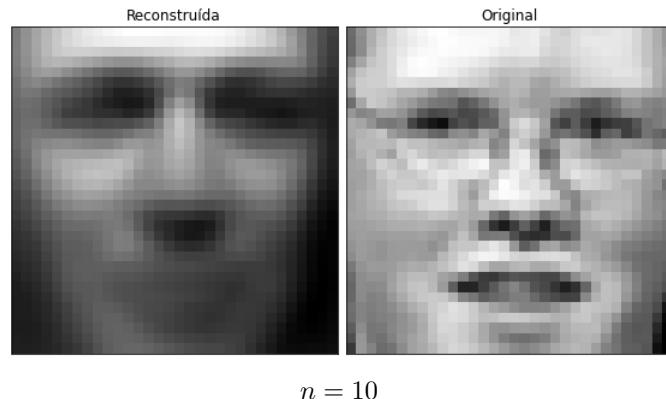
Conseguimos observar o contorno do rosto bem semelhante ao original, mas outras características são completamente ausentes.

Com isso, constatamos que quanto maior for o número de componentes principais, ou seja, quanto maior for n , mais próxima da imagem original a imagem reconstruída chegará, com os detalhes característicos progressivamente se tornando presentes ao aumentar de n . Isso foi percebido também com a implementação de nossa função `proj_evolution_video`, que de forma semelhante à função extra implementada para o problema anterior, cria um gif onde é possível visualizar progressivamente como a imagem reconstruída evolui ao aumentar o número de componentes principais. Tal função pode ser plenamente executada no Jupyter Notebook ou no código original.

2.7 Uma Visualização Interessante:

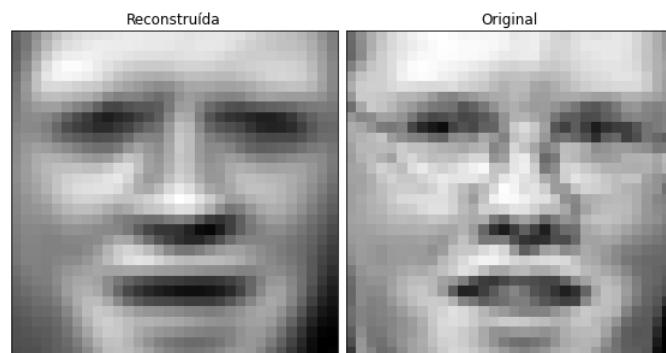
Como um extra, decidimos mostrar a evolução da qualidade da reconstrução conforme o número de componentes principais aumenta (De certa forma, o mesmo processo que ocorre no gif)

Sendo assim, primeiramente para $n = 10$:



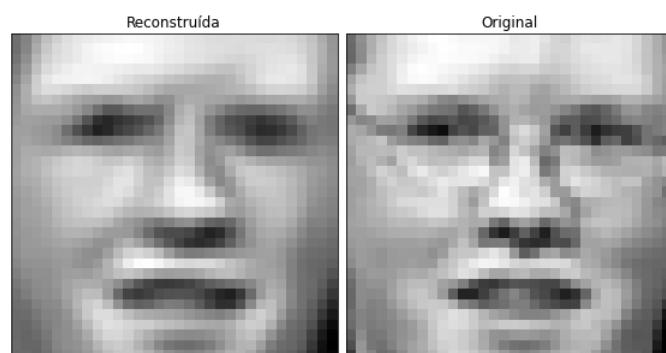
$n = 10$

Para $n = 50$:



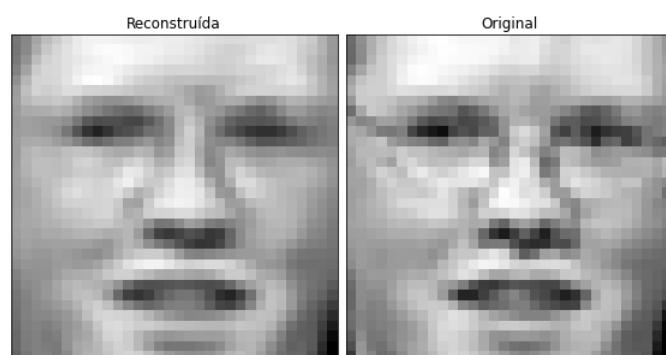
$n = 50$

Para $n = 100$:



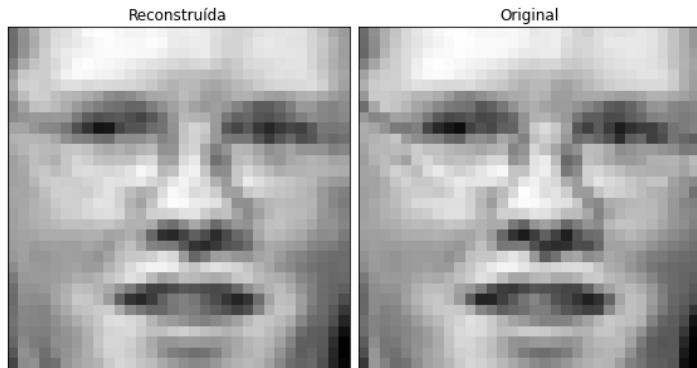
$n = 100$

Para $n = 200$:



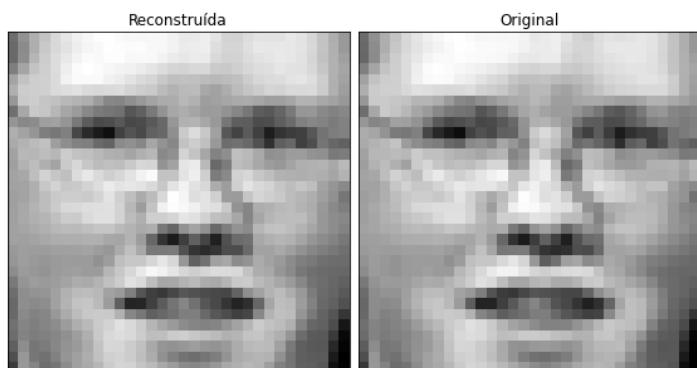
$n = 200$

Para $n = 400$:



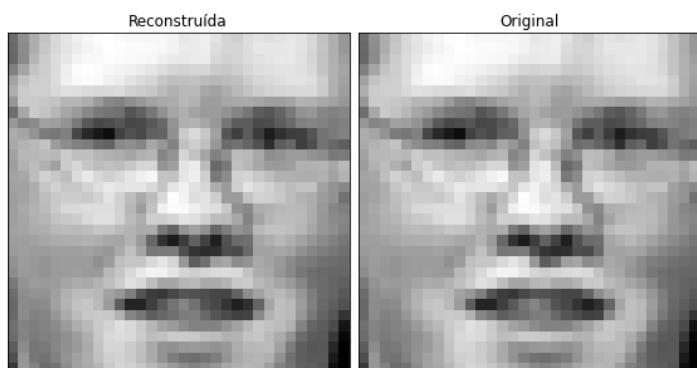
$n = 400$

Para $n = 800$:



$n = 800$

Para $n = 1600$:



$n = 1600$

Assim, é interessante visualizarmos como ocorre progressivamente um ganho de detalhes, mas a partir de dado ponto, no nosso exemplo acima a partir de $n = 800$, o ganho pelo acréscimo do número de componentes principais usados não é tão grande.

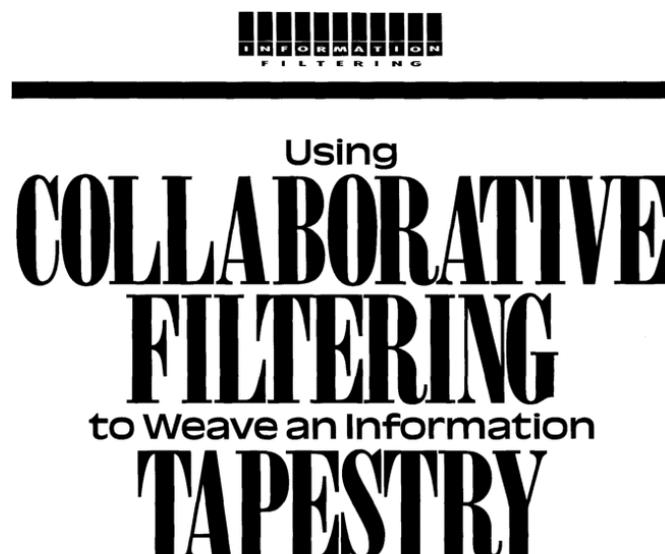
3 Parte II - Sistemas de Recomendação

3.1 História e Ideia de Funcionamento:

Podemos definir sistemas de recomendação como grandes filtros que auxiliam o usuário a tomar uma dada decisão com base na análise e interpretação de dados, suas relações, “pesos” para um dado usuário e entre outros. Com raízes em pesquisas de ciência cognitiva e recuperação/análise de informação, o primeiro sistema que podemos classificar como um exemplo primitivo voltado para recomendação é a “Usenet”, um sistema de comunicação criado na segunda metade da década de 1970 pela Universidade de Duke, na Carolina do Norte; tal sistema possibilitava o compartilhamento de textos de pesquisa entre seus usuários, onde de forma automática ocorria uma classificação dos mesmos em grupos e subgrupos que utilizavam como base o eixo temático em comum de alguns textos. Porém, é importante ressaltar que no exemplo da “Usenet” ainda não se ocorriam recomendações de forma automatizada para os usuários.

Podemos constatar que o primeiro exemplo desenvolvido que segue as diretrizes que utilizamos atualmente para classificar como um sistema de “recomendação” é o bibliotecário computacional Grundy, popular no final da década de 1970 nos EUA, o mesmo realizava anteriormente uma “entrevista” com o usuário, e com base nas informações obtidas, classificava o mesmo em grupos diferentes que tenderiam a gostar de um gênero específico de livro.

O primeiro modelo de algoritmo baseado em filtragem colaborativa, tipo imensamente importante em aplicações modernas, foi o sistema “Tapestry” desenvolvido pela Xerox PARC em 1992, que permitia aos usuários escreverem anotações e comentários nos documentos que os mesmos estavam lendo, e em seguida, utilizava tais relatos para classificar os documentos em um ranking de utilidade.



David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry

Tapestry is an experimental mail system developed at the Xerox Palo Alto Research Center. The motivation for Tapestry comes from the increasing use of electronic mail, which is resulting in users being inundated by a huge stream of incoming documents [2, 7, 12]. One way to handle large volumes of mail is to provide mailing lists, enabling users to subscribe only to those lists of interest to them. However, as illustrated in Figure 1, the set of documents of interest to a particular user rarely map neatly to existing lists. A better solution is for a user to specify a *filter* that scans all lists, selecting interesting documents no matter what list they are in. Several mail systems support filtering based on a document's contents [3, 5, 6, 8]. A basic tenet of the Tapestry work is that more effective filtering can be done by involving humans in the filtering process.

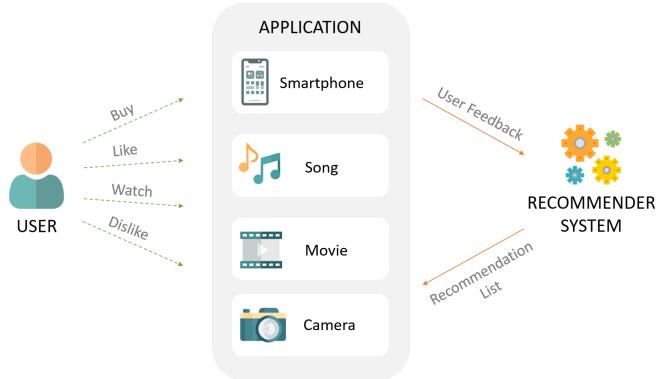
In addition to content-based filtering, the Tapestry system was designed and built to support *collaborative filtering*. Collaborative filtering simply means that people collaborate to help one another perform filtering by recording their reactions to documents they read. Such reactions may be that a document was particularly interesting (or particularly uninteresting). These reactions, more generally called *annotations*, can be accessed by others' filters. One application of annotations is in support of moderated newsgroups.

COMMUNICATIONS OF THE ACM / December 1992 / Vol. 35, No. 12 61

Imagen da Publicação Referente ao Sistema Tapestry em 1992⁶

⁶Imagen retirada de <https://dl.acm.org/doi/10.1145/138859.138867>

Com o avanço contínuo da tecnologia e das técnicas de implementação de sistemas de recomendação, e principalmente, com o avanço da internet e por consequência, do fluxo de informações, sua importância se tornou cada vez maior na sociedade contemporânea, temos seu uso em larga escala em sistemas de empresas, como forma de adquirir um feedback de funcionários, mas sua imensa presença está no meio digital. E-commerce como Amazon, Submarino, Best-Buy, Americanas se utilizam de dados do usuário para que possam gerar recomendações de produtos para o mesmo, e assim, potencializar o número de vendas, enquanto isso, plataformas de streaming, como Netflix, HBO Max, Amazon Prime, Disney+ oferecem recomendações com o intuito de aumentar o tempo de retenção de usuários, e dessa forma, se beneficiarem com a venda de merchandising de suas franquias a longo prazo, planos de assinatura e entre outros.



Exemplo de Funcionamento de um Sistema de Recomendação⁷

No entanto, sua maior aplicação, e possivelmente a mais polêmica, se dá em redes sociais como Facebook, Twitter, Instagram e etc. Com a ideia de recomendar conteúdo com base nas características de um usuário, como sua orientação ideológica, hobbies e entre outros, com o intuito de aumentar o tempo de retenção do mesmo e dessa maneira ofertar uma maior quantidade de propagandas, seu resultado prático lança luz sobre a discussão acerca da criação de bolhas no meio digital, basicamente o oferecimento de conteúdo centrado em apenas algumas características do usuário faz com que a sua “realidade” se molde a aquilo, causando um fenômeno de desconhecimento acerca de outras vivências e até mesmo um “descolamente” completo da compreensão e análise de fatos, como em casos onde o indivíduo é inundado de teorias da conspiração e pseudociências do mais diferentes tipos.

Um exemplo de como um algoritmo descalibrado, adotando parâmetros sem um filtro mínimo, pode gerar resultados “bizarros” é o caso de correlação entre transfobia e supremacistas visualizado no Tik Tok. Com um poderoso algoritmo capaz de fornecer vídeos com base no tempo de visualização de um usuário em um dado eixo temático, o Tik Tok foi ferramenta para a análise da propagação de conteúdos extremistas em redes digitais. Com a visualização contínua de materiais transfóbicos por um tempo relativamente baixo, se constatou que as recomendações do aplicativo passaram a incluir desde conteúdo que reforçavam a transfobia, até conteúdos com teor racista, homofóbicos e nazistas, ou seja, a rede social entrou em uma espiral enorme e infinita de recomendação de conteúdo extremista. Mais informações e detalhes acerca deste caso podem ser encontradas em <https://tinyurl.com/2p8htx8h>.



Exemplos de Redes Sociais⁸

⁷Imagem retirada de <https://www.mdpi.com/2076-3417/10/16/5510>

⁸Imagem retirada de <https://makeawebsitehub.com/social-media-sites/>

3.2 Aprofundando a Teoria: Algoritmo Para Recomendação

Para este projeto tivemos que realizar a implementação de um algoritmo de filtragem colaborativa a partir de uma matriz Y que armazena na linha i e coluna j a nota dada pelo usuário j para o filme i , e de uma matriz R onde temos $R(i, j) = 1$ se o usuário j deu alguma nota para o filme i e 0 caso o contrário. Sendo assim, seguimos o preceito de inicializar a matriz Θ e a matriz X de maneira aleatórias com termos pequenos, e com isso, minimizamos o custo J a partir de duas opções, gradiente descendente ou conjugado.

A matriz X e a matriz Θ foram atualizadas da seguinte maneira:

$$x_k^{(i)} = x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(i)} = \theta_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(j)} + \lambda \theta_k^{(i)} \right)$$

Importante ressaltar que para esse caso, $x_0 = 1$ não é válido, a matriz X inteira tem seus valores obtidos a partir do treinamento do modelo de forma contínua. E é perceptível que todos usuários colaboram para a mensuração da nota de determinado filme.

Assim, obtendo X e Θ finais; e para um usuário com parâmetro $\theta^{(j)}$ e um filme com atributos $x^{(i)}$, podemos prever a nota através do cálculo $(\theta^{(j)})^T x^{(i)}$.

3.3 Algoritmo em Python Para a Parte II:

Assim como na primeira parte, nosso código desenvolvido abarca todas as possibilidades que avançamos factíveis de implementar em um primeiro momento, gerando um modelo que permite uma grande interação por parte do usuário, tanto na definição de parâmetros, como na visualização e obtenção de resultados. A seguir, análises voltadas para cada função presente no programa:

```

1  def Load():
2      g=input("\nEndereço da pasta onde se encontram os dois arquivos com
3          os dados (sem o nome dos mesmos): ")
4      with open(g+"\dado3.txt") as f:
5          T=f.readlines()
6          T=[line.rstrip() for line in T]
7      T=np.array(T).flatten()
8      for a in T:
9          a=str(a)
10     g=g+"\dado2"
11     dado2=loadmat(g)
12     R=dado2.get('R')
13     Y=dado2.get('Y')
14     q=int(input("\nGostaria de Visualizar um plot com as distribuições
15 de notas para cada filme? 1 para sim e 2 para não: "))
16     if q==1:
17         plt.figure(figsize=(8,16))
18         plt.imshow(Y)
19         plt.xlabel("Users")
20         plt.ylabel("Movies")
21         aux_Y=np.zeros((Y.shape[0],Y.shape[1]))
22         mean=np.zeros((Y.shape[0],1))
23         for i in range(0,Y.shape[0]):
24             mean[i]=np.sum(Y[i,:])/np.count_nonzero(R[i,:])
25             aux_Y[i,R[i,:]==1] = Y[i,R[i,:]==1] - mean[i]
26         Ynorm=aux_Y
27         num_users=Y.shape[1]
28         num_movies=Y.shape[0]
29         num_features=int(input("\nQual você gostaria que fosse a dimensão
30         das colunas de theta? "))
31         theta=np.random.randn(num_users,num_features)
32         X=np.random.randn(num_movies,num_features)
33         params=np.append(X.flatten(),theta.flatten())
34         return R,Ynorm,Y,T,X,theta,mean,params,num_users,
35         num_movies,num_features

```

Começando pela função *Load*, responsável por importar os dados a partir de um diretório oferecido pelo usuário, a mesma prepara todos os parâmetros iniciais para inicialização de todo o processo. É nela que o usuário escolhe a dimensão das colunas de *Theta* e visualiza o plot com as distribuições de notas por filme.

```

1 def Verification(T,Y,R):
2     a=int(input("\nGostaria de verificar se um filme esta na lista de
3     avaliados? 1 para sim e 2 para nao: "))
4     if a==1:
5         u=1
6         while u==1:
7             b=input("\nNome do filme que se quer buscar: ")
8             e=cp.deepcopy(b).lower()
9             for i in range(len(T)):
10                 d=cp.deepcopy(T[i]).lower()
11                 if d.find(e)!=-1:
12                     print("\nSim, o filme %s, lancado em %s,
13                     esta na lista
14                     com uma nota media de %.1f/5"%(b, d[-5:-1],
15                     np.sum(Y[i,:]*R[i,:])/np.sum(R[i,:])))
16                     u=int(input("\nGostaria de uma nova consulta? 1 para
17                     sim e 2 para nao: "))
18                     break
19                 elif i==len(T)-1:
20                     print("\nNao, o filme %s nao esta na lista"%b)
21                     u=int(input("\nGostaria de uma nova consulta? 1
22                     para sim e 2 para nao: "))
23                     break
24             b=int(input("\nGostaria de verificar todos os filmes que sairam em
25             um dado ano? 1 para
26             sim e 2 para nao: "))
27             if b==1:
28                 u=1
29                 k=[]
30                 while u==1:
31                     b=int(input("\nAno que se quer verificar: "))
32                     for i in range(len(T)):
33                         d=cp.deepcopy(T[i])
34                         if d[-5]=='1':
35                             if b==int(d[-5:-1]):
36                                 if T[i][1]==": ":
37                                     k.append(d[2:-7])
38                                 elif T[i][2]==": ":
39                                     k.append(d[3:-7])
40                                 elif T[i][3]==": ":
41                                     k.append(d[4:-7])
42                                 elif T[i][4]==": ":
43                                     k.append(d[5:-7])
44                         if k!=[] and i==len(T)-1:
45                             print("\nNo ano %d sairam os seguintes filmes:"%(b))
46                             print("\n")
47                             for j in k:
48                                 print("-----")
49                                 print(j)
50                                 print("-----")
51                             u=int(input("\nGostaria de uma nova consulta? 1
52                             para sim e 2 para nao: "))
53                             k=[]
54                             break
55                         elif i==len(T)-1 and k==[]:
56                             print("\nNenhum filme da lista saiu em %d"%(b))
57                             u=int(input("\nGostaria de uma nova consulta? 1 para
58                             sim e 2 para nao: "))
59                             k=[]
60                             break
61             return

```

A função *Verification* permite ao usuário consultar se um dado filme específico se encontra ou não na lista. É possível também realizar a buscar por um dado ano e receber como resultado todos os filmes que saíram no mesmo.

```

1 | def CostFunction(params ,Ynorm ,R ,num_users ,num_movies ,num_features ,beta ,
2 | lamb):
3 |     X=params [:num_movies*num_features].reshape(num_movies ,num_features)
4 |     Theta = params [num_movies*num_features :].reshape(num_users ,
5 |         num_features)
6 |     predictions = X @ Theta.T
7 |     j=(predictions - Ynorm)
8 |     J = 1/2 * np.sum((j**2) * R)
9 |     X_grad = j*R @ Theta
10 |    Theta_grad = (j*R).T @ X
11 |    grad = np.append(X_grad.flatten() ,Theta_grad.flatten())
12 |    if beta==1:
13 |        reg_X = lamb/2 * np.sum(Theta**2)
14 |        reg_Theta = lamb/2 *np.sum(X**2)
15 |        reg_J = J + reg_X + reg_Theta
16 |        reg_X_grad = X_grad + lamb*X
17 |        reg_Theta_grad = Theta_grad + lamb*Theta
18 |        reg_grad = np.append(reg_X_grad.flatten() ,
19 |            reg_Theta_grad.flatten())
20 |        J_history.append(reg_J)
21 |        return reg_J,reg_grad
22 |    else:
23 |        J_history.append(J)
24 |        return J , grad

```

A função *CostFunction* é nossa implementação do cálculo de custo, e como é possível observar, abrange tanto as versões regularizadas quanto as versões não regularizadas do gradiente e do custo, basta o usuário optar pela de sua preferência.

```

1 | def Gradient(params ,Ynorm ,R ,num_users ,num_movies ,num_features):
2 |     alpha=int(input("\nNumero de iteracoes desejado: "))
3 |     c=int(input("\nGostaria de fazer a minimizacao atraves de
4 |         gradiente descendente ou gradiente conjugado?
5 |         1 para descendente e 2 para conjugado: "))
6 |     beta=int(input("\nGostaria que o custo e o gradiente obtidos fossem
7 |         regularizados? 1 para sim e 2 para nao: "))
8 |     if beta==1:
9 |         lamb=float(input("\nQual seria o valor de lambda? "))
10 |    else:
11 |        lamb=0
12 |    if c==2:
13 |        initial_time=time.time()
14 |        result=minimize(CostFunction ,params ,args=(Ynorm ,R ,num_users
15 |            ,num_movies ,num_features ,beta ,lamb) , method='CG' , jac=True ,
16 |            tol=None , callback=None , options={'maxiter':alpha , 'disp':True ,
17 |                'gtol':1e-4})
18 |        end_time=time.time()
19 |        h=int(input("\nGostaria de ver os resultados obtidos pela
20 |            minimizacao atraves do gradiente conjugado:?
21 |            1 para sim e 2 para nao: "))
22 |        if h==1:
23 |            print("\n")
24 |            print(result)
25 |        return result.x , end_time-initial_time ,(alpha ,lamb ,0
26 |            ,end_time-initial_time)
27 |    if c==1:
28 |        p=float(input("\nQual seria o valor da taxa de aprendizado? "))
29 |        initial_time=time.time()
30 |        result=GradientDescent(params ,Ynorm ,R ,num_users ,num_movies
31 |            ,num_features ,p ,alpha ,beta ,lamb)
32 |        end_time=time.time()
33 |        return result , end_time-initial_time ,(alpha ,lamb ,p ,
34 |            end_time-initial_time)

```

Em seguida, temos a função *Gradient*, ela permite ao usuário escolher qual método de minimização utilizar, gradiente conjugado ou descendente, em seguida, recebe do usuário os parâmetros para a realização do processo. É nessa etapa que também é registrado o tempo total do treinamento.

```

1 def GradientDescent(params ,Ynorm ,R,num_users ,num_movies ,num_features ,
2 alpha ,num_inter ,beta ,lamb):
3     X=params [:num_movies*num_features].reshape(num_movies ,num_features)
4     Theta = params [num_movies*num_features: ].reshape(num_users ,
5     num_features)
6     for i in range(num_inter):
7         params = np.append(X.flatten() ,Theta.flatten())
8         J,grad=CostFunction(params ,Y,R,num_users ,num_movies ,num_features ,
9         beta ,lamb)
10        X_grad=grad [:num_movies*num_features].reshape(num_movies ,
11        num_features)
12        Theta_grad=grad [num_movies*num_features: ].reshape(num_users ,
13        num_features)
14        X=X-(alpha * X_grad)
15        Theta=Theta-(alpha * Theta_grad)
16        params=np.append(X.flatten() ,Theta.flatten())
17    return params

```

A função *GradientDescent* é a implementação padrão do algoritmo de gradiente descendente para o caso do usuário optar pelo uso da mesma como técnica de minimização.

```

1 def Result(params ,total_time ,J_history ,mean ,num_features ,num_movies ,
2 num_users ,T):
3     X=params [:num_movies*num_features].reshape(num_movies ,
4     num_features)
5     theta=params [num_movies*num_features: ].reshape(num_users ,
6     num_features)
7     result=X @ theta.T
8     prediction=result [:,0] [:,np.newaxis]+mean
9     df = pd.DataFrame(np.hstack((prediction,T[:,np.newaxis])))
10    k=int(input("\nGostaria de buscar a nota predita pelo modelo de algum
11    filme em específico? 1 para sim e 2 para não: "))
12    if k==1:
13        u=1
14        while u==1:
15            b=input("\nNome do filme que se quer buscar: ")
16            e=cp.deepcopy(b).lower()
17            for i in range(len(df[1])):
18                d=cp.deepcopy(df[1][i]).lower()
19                if d.find(e)!=-1:
20                    print("\nO filme %s, lançado em %s, teve sua nota
21                    prevista de %.1f/5, enquanto através de média
22                    simples, sua nota foi %.1f/5%(b, d[-5:-1],
23                    float(df[0][i]), mean[i,0]))"
24                    u=int(input("\nGostaria de uma nova consulta?
25                    1 para sim e 2 para não: "))
26                    break
27                elif i==len(T)-1:
28                    print("\nInfelizmente o filme %s não está na lista"
29                    "%b")
30                    u=int(input("\nGostaria de uma nova consulta?
31                    1 para sim e 2 para não: "))
32                    break
33    df.sort_values(by=[0] ,ascending=False ,inplace=True)
34    df.reset_index(drop=True,inplace=True)
35    t=int(input("\nGostaria de visualizar um 'top' dos melhores e piores
36    filmes? 1 para sim e 2 para não: "))
37    save=[]
38    if t==1:
39        g=int(input("\nQual seria o 'top' que se quer visualizar? "))
40        print("\nMelhores recomendações para você:\n")
41        for i in range(g):
42            print("Nota predita",round(float(df[0][i]),1)," para o filme
43            ",df[1][i])

```

```

44     save.append((round(float(df[0][i]),1),df[1][i]))
45     a=list(reversed(df[0]))
46     b=list(reversed(df[1]))
47     print("\n")
48     print("Piores recomendacoes para voce:\n")
49     for j in range(g):
50         print("Nota predita",round(float(a[j]),1)," para o filme",
51               b[j])
52         save.append((round(float(a[j]),1),b[j]))
53     print("\nO tempo total de execucao do modelo para predicao das
54     notas foi de %.1f segundos"%total_time)
55     return df,prediction,save

```

A função *Result*, como o nome sugere, é responsável por apresentar ao usuário os resultados obtidos pelo modelo, para tal, o mesmo pode pesquisar por um filme específico e averiguar a nota predita em comparação com a nota obtida através de média simples; é possível também visualizar um “top n”, com n sendo um número escolhido pelo usuário, dos filmes com maiores e menores notas preditas.

```

1 def Save(save,alpha,l,J_history,params,num_users,num_movies,
2 num_features):
3     X=params[:num_movies*num_features].reshape(num_movies,
4         num_features)
5     theta=params[num_movies*num_features:)].reshape(num_users,
6         num_features)
7     g=int(input("Gostaria de salvar as principais informacoes deste
8     programa em um arquivo txt e a matriz X e theta em arquivos CSV?
9     1 para sim e 2 para nao: "))
10    if g==1:
11        o=(input("Endereco para salvar os arquivoa (sem o nome do mesmo)
12          : "))
13        aux=cp.deepcopy(o)
14        o=r"%s/Resultado_Recomendacao.txt"%aux
15        v=r"%s/Matriz_X.csv"%aux
16        h=r"%s/Matriz_theta.csv"%aux
17        f=open(o,'w+')
18        f.write("Principais parametros e resultados do modelo:")
19        f.write("\nPara a dimensao de colunas dos vetores theta, foi
20        utilizado um valor de %d"%l)
21        if alpha[2]==0:
22            f.write("\nO metodo de minimizacao utilizado foi o de
23            gradiente conjugado")
24        else:
25            f.write("\nO metodo de minimizacao utilizado foi o de
26            gradiente descendente com uma taxa de aprendizado igual a
27            %.5f"%alpha[2])
28        f.write("\nO numero total de iteracoes para o treino do modelo
29        foi de %d"%len(J_history))
30        if alpha[1]==0:
31            f.write("\nNao houve regularizacao")
32        else:
33            f.write("\nHouve regularizacao e o valor de lambda
34            utilizado foi %.5f"%alpha[1])
35        f.write("\nO custo inicial foi de
36            %.4f, e terminou com %.4f"%(J_history[0],J_history[-1]))
37        f.write("\nO tempo total de execucao do modelo foi de
38            %.3f segundos"%alpha[3])
39        f.write("\n")
40        if save!=[]:
41            f.write("\nAs melhores recomendacoes para o usuario foram
42            as seguintes:")
43            for i in range(0,int(len(save)/2)):
44                f.write("\nFilme %s com nota %.1f"%(save[i][1],
45                    save[i][0]))
46                f.write("\n")
47                f.write("\nAs piores recomendacoes para o usuario foram
48            as seguintes:")

```

```

49         for j in range((int(len(save)/2))+1,len(save)):
50             f.write("\nFilme %s com nota %.1f"%(save[j][1],
51                 save[j][0]))
52         f.close()
53         with open(v, 'w', encoding='UTF8', newline='') as f:
54             writer=csv.writer(f)
55             writer.writerows(X)
56         with open(h, 'w', encoding='UTF8', newline='') as f:
57             writer=csv.writer(f)
58             writer.writerows(theta)
59         print("\nSeus arquivo foram salvos no diretorio %s :)"%aux)

```

Na função *Save*, oferecemos ao usuário a possibilidade de salvar os principais resultados do modelo em um arquivo de texto, e também salvamos as matrizes *X* e *Theta* obtidas no final.

```

1 import numpy as np
2 import copy as cp
3 import time
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import csv
7 from scipy.io import loadmat
8 from scipy.optimize import minimize

```

Acima estão as bibliotecas e funções prontas que importamos para a utilização no programa.

```

1 J_history=[]
2 R,Ynorm,Y,T,X,theta,mean,params,num_users,num_movies,num_features=Load()
3 Verification(T,Y,R)
4 params,total_time,alpha=Gradient(params,Ynorm,R,num_users,
5 num_movies,num_features)
6 df,prediction,save=Result(params,total_time,J_history,mean,num_features,
7 num_movies,num_users,T)
8 Save(save,alpha,theta.shape[1],J_history,params,num_users,num_movies
9 ,num_features)

```

E por fim, nossa “central de comando”, que para este código, serviu apenas como um local para acionamento das funções desenvolvidas. É também o local onde inicializamos *J_History* de forma global para que conseguíssemos guardar o custo do gradiente conjugado e do gradiente descendente iteração a iteração.

Com isso, concluímos que a implementação do algoritmo para o problema tratado foi satisfeatória, permitindo uma grande interação por parte do usuário e oferecendo os resultados de forma adequada e completa. A seguir, um exemplo de como o resultado é apresentado no arquivo de texto para um caso onde foi utilizado gradiente conjugado:

Principais parâmetros e resultados do modelo:

Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100
O método de minimização utilizado foi o de gradiente conjugado
O número total de iterações para o treino do modelo foi de 575
Houve regularização e o valor de lambda utilizado foi 10.00000
O custo inicial foi de 6344326.7822, e terminou com 34027.4697
O tempo total de execução do modelo foi de 49.341 segundos

As melhores recomendações para o usuário foram as seguintes:

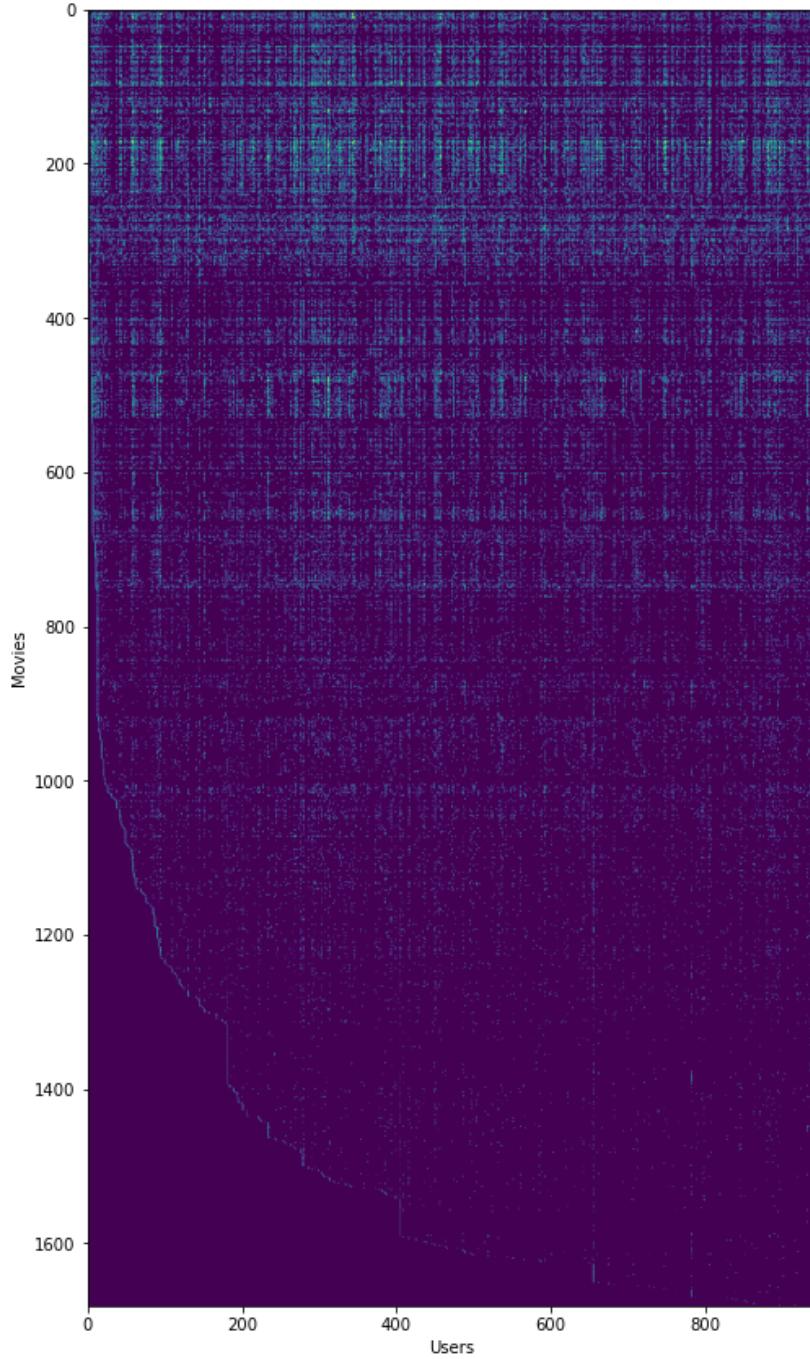
Filme 285 Secrets & Lies (1996) com nota 5.4
Filme 169 Wrong Trousers, The (1993) com nota 5.1
Filme 100 Fargo (1996) com nota 5.1
Filme 408 Close Shave, A (1995) com nota 5.0
Filme 50 Star Wars (1977) com nota 5.0

E as piores recomendações para o usuário foram as seguintes:

Filme 830 Power 98 (1995) com nota 1.0
Filme 1308 Babyfever (1994) com nota 1.0
Filme 1561 Tigrero: A Film That Was Never Made (1994) com nota 1.0
Filme 1577 Death in the Garden (Mort en ce jardin, La) (1956) com nota 1.0
Filme 1364 Bird of Prey (1996) com nota 1.0

3.4 Primeiro e Segundo Problemas da Segunda Parte:

Para a resolução deste problema, vamos iniciar analisando os dados que iremos utilizar, mais especificamente, a matriz Y onde a posição (i, j) representa a nota dada pelo usuário j para o filme i . Assim, o plot da matriz Y é o seguinte:



Plot da Matriz Y

Como é possível constatar, ocorre uma clara desigualdade na distribuição dos dados em relação a quantidade de votos para cada filme; é visível que temos uma incidência grande de votos para os aproximadamente 500 primeiros filmes, porém, quanto mais vamos aumentando i , e consequentemente, acessando outros filmes com essas coordenadas i , menor é a quantidade de votos por filme. Tal ocorrência pode gerar distorções no resultado final, dificultando o processo de treinamento do modelo, como veremos futuramente.

Com o fato anterior em mente, tomamos como padrão para a etapa seguinte a dimensão j de Θ como 100, assim, realizamos quatro exemplos para comparar e analisar os resultados obtidos: um exemplo com gradiente conjugado com regularização, um com gradiente conjugado sem regularização, outro com gradiente descendente com regularização, e por fim, um com gradiente descendente sem regularização.

Para um caso com gradiente conjugado regularizado, com 800 iterações máximas e $\lambda = 10$, tivemos a seguinte entrada:

```
Endereço da pasta onde se encontram os dois arquivos com os dados  
(sem o nome dos mesmos): C:\Users\USUARIO\Desktop\projeto 3

Gostaria de Visualizar um plot com as distribuições de notas para cada filme?  
1 para sim e 2 para não: 2

Qual você gostaria que fosse a dimensão de theta? 100

Gostaria de verificar se um filme está na lista de avaliados?  
1 para sim e 2 para não: 2

Gostaria de verificar todos os filmes que saíram em um dado ano?  
1 para sim e 2 para não: 2

Número de iterações desejado: 800

Gostaria de fazer a minimização através de gradiente descendente ou  
gradiente conjugado?  
1 para descendente e 2 para conjugado: 2

Gostaria que o custo e o gradiente obtidos fossem regularizados?  
1 para sim e 2 para não: 1

Qual seria o valor de lambda? 10
Optimization terminated successfully.
    Current function value: 34027.469714
    Iterations: 469
    Function evaluations: 696
    Gradient evaluations: 696

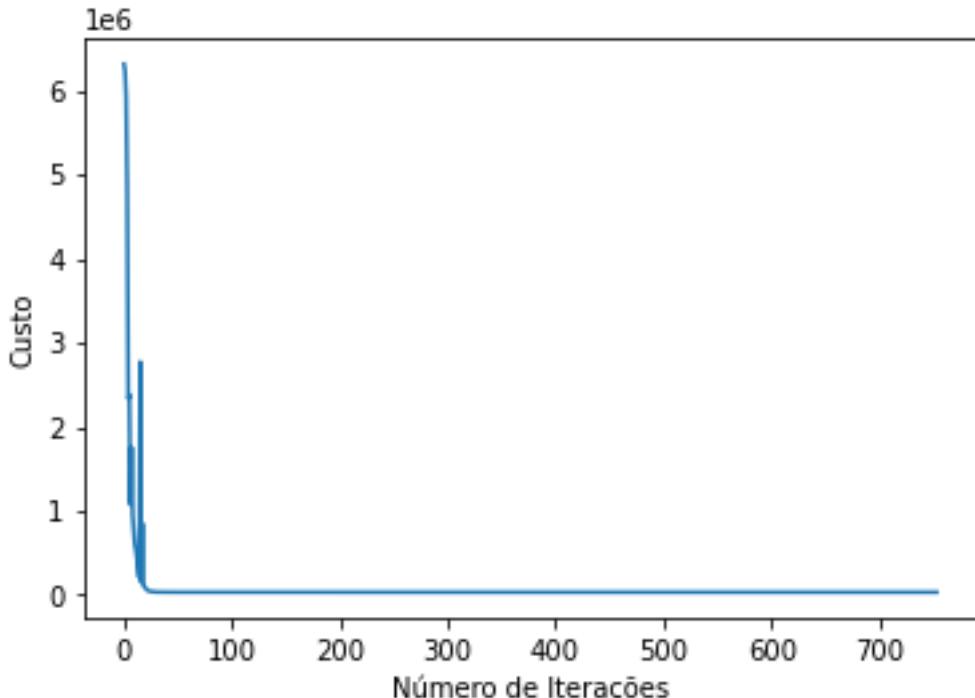
Gostaria de ver os resultados obtidos pela minimização através do  
gradiente conjugado?:  
1 para sim e 2 para não: 1

fun: 34027.46971365673
jac: array([ 1.40272170e-05, -1.87586826e-06,  8.17912430e-06, ...,
           -1.17172077e-06,   3.16259174e-06, -4.47738146e-07])
message: 'Optimization terminated successfully.'
nfev: 696
nit: 469
njev: 696
status: 0
success: True
x: array([-0.02364712, -0.18763051, -0.06114786, ...,  0.09955425,
          0.10761953,  0.06524088])
```

Como dados de saída no arquivo de texto, tivemos o seguinte:

```
Principais parâmetros e resultados do modelo:  
Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100  
O método de minimização utilizado foi o de gradiente conjugado  
O número total de iterações para o treino do modelo foi de 696  
Houve regularização e o valor de lambda utilizado foi 10.00000  
O custo inicial foi de 6328675.9075, e terminou com 34027.4697  
O tempo total de execução do modelo foi de 62.042 segundos
```

O gráfico de custo por iteração para este exemplo foi o seguinte:



Curva de Custos do Gradiente Conjugado Regularizado

O top 10 de melhores e piores filmes obtidos foi o seguinte:

<p>As melhores recomendações para o usuário foram as seguintes: Filme 285 Secrets & Lies (1996) com nota 5.4 Filme 169 Wrong Trousers, The (1993) com nota 5.1 Filme 100 Fargo (1996) com nota 5.1 Filme 408 Close Shave, A (1995) com nota 5.0 Filme 50 Star Wars (1977) com nota 5.0 Filme 302 L.A. Confidential (1997) com nota 5.0 Filme 1536 Aiqing wansui (1994) com nota 5.0 Filme 814 Great Day in Harlem, A (1994) com nota 5.0 Filme 1201 Marlene Dietrich: Shadow and Light (1996) com nota 5.0 Filme 1467 Saint of Fort Washington, The (1993) com nota 5.0</p>
<p>E as piores recomendações para o usuário foram as seguintes: Filme 1359 Boys in Venice (1996) com nota 1.0 Filme 1564 To Cross the Rubicon (1991) com nota 1.0 Filme 1582 T-Men (1947) com nota 1.0 Filme 1571 Touki Bouki (Journey of the Hyena) (1973) com nota 1.0 Filme 858 Amityville: Dollhouse (1996) com nota 1.0 Filme 1569 Vie est belle, La (Life is Rosey) (1987) com nota 1.0 Filme 1583 Invitation, The (Zaproszenie) (1986) com nota 1.0 Filme 1587 Terror in a Texas Town (1958) com nota 1.0 Filme 1408 Gordy (1995) com nota 1.0 Filme 1678 Mat' i syn (1997) com nota 1.0</p>

Assim, conseguimos constatar que o modelo foi efetivo em sua função de recomendar os dez melhores e dez piores filmes, tal processo ocorreu também em um tempo relativamente rápido e com uma queda brusca de custo a cada iteração. Importante ressaltarmos que algumas notas, como a do primeiro, segundo e terceiro colocado no top 10, ultrapassaram a margem de notas que ficam entre 1 e 5, isso pode ter ocorrido por distorções nos dados (Excesso de um e falta do outro por exemplo), tamanho não tão adequado da dimensão j de Θ e entre outros.

Para um caso de gradiente conjugado sem regularização e com 800 iterações máximas, tivemos a seguinte entrada:

```
Endereço da pasta onde se encontram os dois arquivos com os dados  
(sem o nome dos mesmos): C:\Users\USUARIO\Desktop\projeto 3

Gostaria de Visualizar um plot com as distribuições de notas para cada filme?  
1 para sim e 2 para não: 2

Qual você gostaria que fosse a dimensão de theta? 100

Gostaria de verificar se um filme está na lista de avaliados?  
1 para sim e 2 para não: 2

Gostaria de verificar todos os filmes que saíram em um dado ano?  
1 para sim e 2 para não: 2

Número de iterações desejado: 800

Gostaria de fazer a minimização através de gradiente descendente ou  
gradiente conjugado?  
1 para descendente e 2 para conjugado: 2

Gostaria que o custo e o gradiente obtidos fossem regularizados?  
1 para sim e 2 para não: 2
Warning: Maximum number of iterations has been exceeded.
    Current function value: 7.349663
    Iterations: 800
    Function evaluations: 1181
    Gradient evaluations: 1181

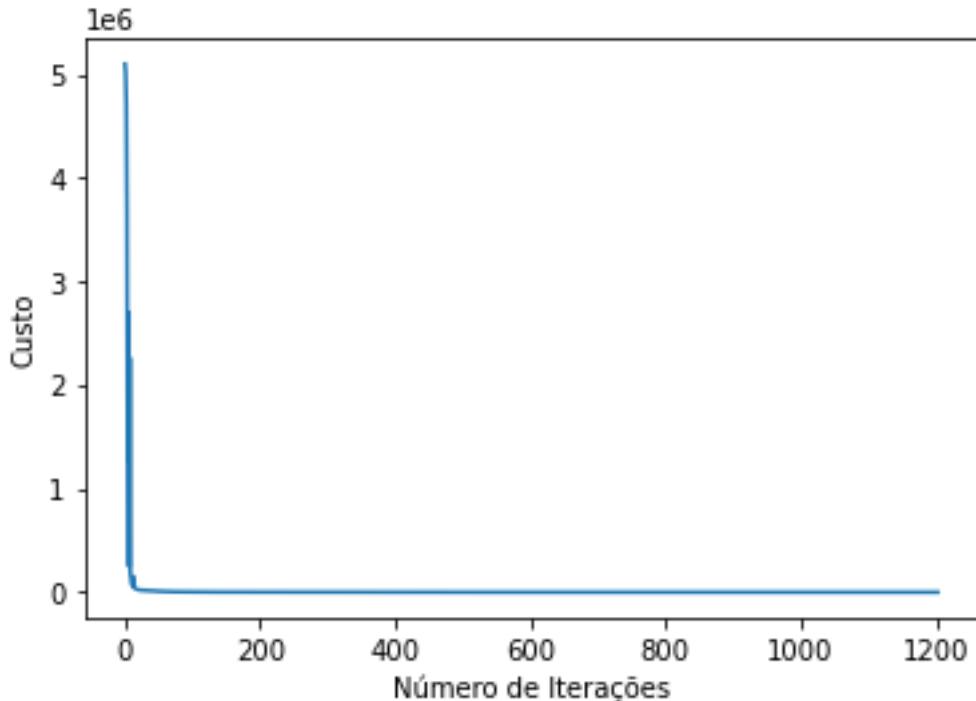
Gostaria de ver os resultados obtidos pela minimização através do  
gradiente conjugado?  
1 para sim e 2 para não: 1

fun: 7.349662960975326
jac: array([-0.00397095,  0.00277309,  0.00603297, ..., -0.00166512,
          0.01299088, -0.01277364])
message: 'Maximum number of iterations has been exceeded.'
nfev: 1181
nit: 800
njev: 1181
status: 1
success: False
x: array([ 0.54231784, -0.34282592,  0.03080289, ...,  0.6219037 ,
         -1.19168984,  0.54618493])
```

Como dados de saída no arquivo de texto, tivemos o seguinte:

```
Principais parâmetros e resultados do modelo:
Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100
O método de minimização utilizado foi o de gradiente conjugado
O número total de iterações para o treino do modelo foi de 1181
Não houve regularização
O custo inicial foi de 5090170.7703, e terminou com 7.3497
O tempo total de execução do modelo foi de 98.684 segundos
```

O gráfico de custo por iteração para este exemplo foi o seguinte:



O top 10 de melhores e piores filmes foi os seguintes:

As melhores recomendações para o usuário foram as seguintes:
Filme 801 Air Up There, The (1994) com nota 10.0
Filme 575 City Slickers II: The Legend of Curly's Gold (1994) com nota 9.9
Filme 1472 Visitors, The (Visiteurs, Les) (1993) com nota 9.9
Filme 1122 They Made Me a Criminal (1939) com nota 9.8
Filme 1142 When We Were Kings (1996) com nota 9.7
Filme 663 Being There (1979) com nota 9.7
Filme 1010 Basquiat (1996) com nota 9.7
Filme 429 Day the Earth Stood Still, The (1951) com nota 9.7
Filme 1131 Safe (1995) com nota 9.6
Filme 902 Big Lebowski, The (1998) com nota 9.5

E as piores recomendações para o usuário foram as seguintes:
Filme 426 Transformers: The Movie, The (1986) com nota -0.0
Filme 496 It's a Wonderful Life (1946) com nota -0.0
Filme 411 Nutty Professor, The (1996) com nota -0.0
Filme 1349 Mille bolle blu (1993) com nota -0.0
Filme 1118 Up in Smoke (1978) com nota -0.0
Filme 1429 Sliding Doors (1998) com nota -0.0
Filme 734 Made in America (1993) com nota -0.0
Filme 1584 Symphonie pastorale, La (1946) com nota -0.0
Filme 1203 Top Hat (1935) com nota -0.0
Filme 579 Fatal Instinct (1993) com nota -0.0

Como podemos observar, o algoritmo foi eficiente em sua tarefa de classificação, reduzindo bruscamente o custo em comparação ao modelo anterior, porém, tal processo levou consideravelmente mais tempo para ocorrer. É perceptível a divergência dos filmes no top 10, tanto dos piores quanto dos melhores, em relação a escala de nota indo de um a cinco, é facilmente constatável também que tais notas discrepantes ocorrem em filmes que possuem pouquíssimos votos na matriz Y , mostrando que a falta de dados interfere na resolução do modelo.

Para um caso de gradiente descendente regularizado, 800 iterações, $\lambda = 10$ e taxa de aprendizado igual a 0.001, tivemos a seguinte entrada:

```
Endereço da pasta onde se encontram os dois arquivos com os dados  
(sem o nome dos mesmos): C:\Users\USUARIO\Desktop\projeto 3

Gostaria de Visualizar um plot com as distribuições de notas para cada filme?  
1 para sim e 2 para não: 2

Qual você gostaria que fosse a dimensão de theta? 100

Gostaria de verificar se um filme está na lista de avaliados?  
1 para sim e 2 para não: 2

Gostaria de verificar todos os filmes que saíram em um dado ano?  
1 para sim e 2 para não: 2

Número de iterações desejado: 800

Gostaria de fazer a minimização através de gradiente descendente ou  
gradiente conjugado?  
1 para descendente e 2 para conjugado: 1

Gostaria que o custo e o gradiente obtidos fossem regularizados?  
1 para sim e 2 para não: 1

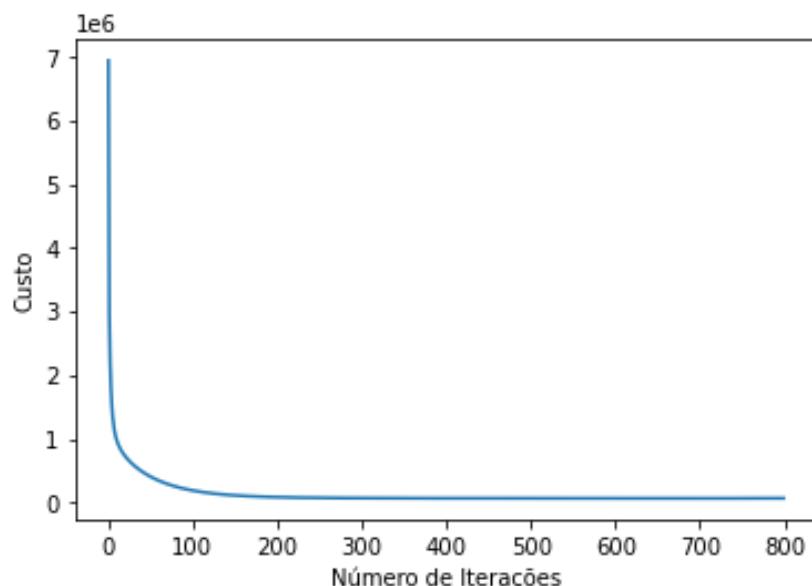
Qual seria o valor de lambda? 10

Qual seria o valor da taxa de aprendizado? 0.001
```

Como dados de saída no arquivo de texto, tivemos o seguinte:

```
Principais parâmetros e resultados do modelo:  
Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100  
O método de minimização utilizado foi o de gradiente descendente com uma  
taxa de aprendizado igual a 0.00100  
O número total de iterações para o treino do modelo foi de 800  
Houve regularização e o valor de lambda utilizado foi 10.00000  
O custo inicial foi de 6945513.2877, e terminou com 68817.1683  
O tempo total de execução do modelo foi de 66.991 segundos
```

O gráfico de custo por iteração para este exemplo foi o seguinte:



Curva de Custos do Gradiente Descendente Regularizado

O top 10 de melhores e piores filmes foi o seguinte:

As melhores recomendações para o usuário foram as seguintes:
Filme 408 Close Shave, A (1995) com nota 9.7
Filme 285 Secrets & Lies (1996) com nota 9.7
Filme 169 Wrong Trousers, The (1993) com nota 9.6
Filme 483 Casablanca (1942) com nota 9.5
Filme 511 Lawrence of Arabia (1962) com nota 9.3
Filme 100 Fargo (1996) com nota 9.3
Filme 357 One Flew Over the Cuckoo's Nest (1975) com nota 9.3
Filme 64 Shawshank Redemption, The (1994) com nota 9.3
Filme 114 Wallace & Gromit: The Best of Aardman Animation (1996) com nota 9.2
Filme 181 Return of the Jedi (1983) com nota 9.2

E as piores recomendações para o usuário foram as seguintes:

Filme 1582 T-Men (1947) com nota 1.0
Filme 1569 Vie est belle, La (Life is Rosey) (1987) com nota 1.0
Filme 1561 Tigrero: A Film That Was Never Made (1994) com nota 1.0
Filme 1576 Hungarian Fairy Tale, A (1987) com nota 1.0
Filme 1557 Yankee Zulu (1994) com nota 1.0
Filme 1566 Man from Down Under, The (1943) com nota 1.0
Filme 1571 Touki Bouki (Journey of the Hyena) (1973) com nota 1.0
Filme 1584 Symphonie pastorale, La (1946) com nota 1.0
Filme 1587 Terror in a Texas Town (1958) com nota 1.0
Filme 1580 Liebelei (1933) com nota 1.0

Como é possível observar, o desempenho nas recomendações para o usuário foram semelhantes ao do exemplo imediatamente anterior, se observando inclusive uma grande discrepância nas notas do top 10 de melhores filmes com a escala originalmente adotada (1-5). Em relação ao tempo, o desempenho foi superior ao do gradiente conjugado não regularizado e ligeiramente inferior ao do gradiente conjugado regularizado, sendo o custo final consideravelmente maior do que o de ambos modelos já analisados anteriormente.

Para um caso de gradiente descendente não regularizado, 800 iterações e taxa de aprendizado igual a 0.001, tivemos a seguinte entrada:

Endereço da pasta onde se encontram os dois arquivos com os dados (sem o nome dos mesmos): C:\Users\USUARIO\Desktop\projeto 3

Gostaria de Visualizar um plot com as distribuições de notas para cada filme?
1 para sim e 2 para não: 2

Qual você gostaria que fosse a dimensão de theta? 100

Gostaria de verificar se um filme está na lista de avaliados?
1 para sim e 2 para não: 2

Gostaria de verificar todos os filmes que saíram em um dado ano?
1 para sim e 2 para não: 2

Número de iterações desejado: 800

Gostaria de fazer a minimização através de gradiente descendente ou gradiente conjugado?
1 para descendente e 2 para conjugado: 1

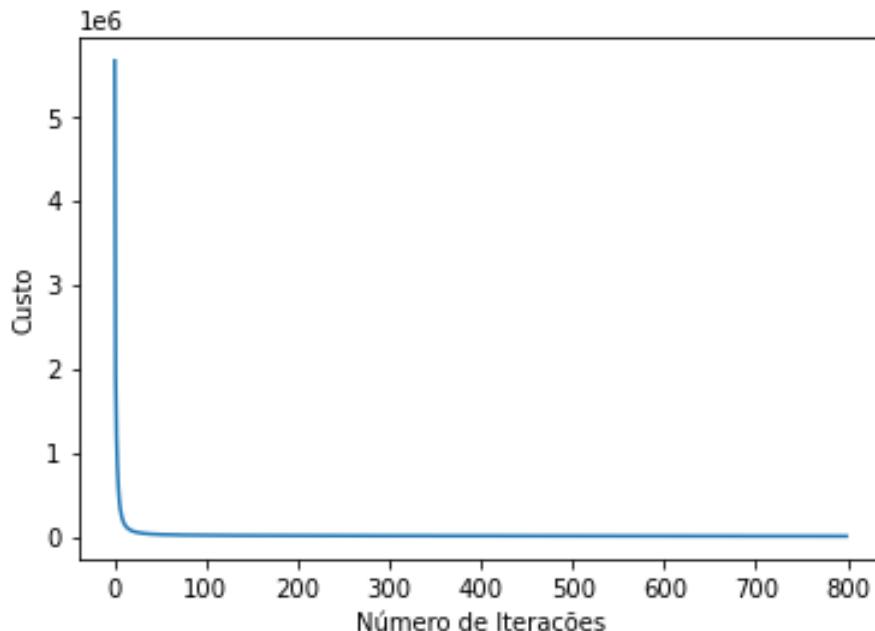
Gostaria que o custo e o gradiente obtidos fossem regularizados?
1 para sim e 2 para não: 2

Qual seria o valor da taxa de aprendizado? 0.001

Como dados de saída no arquivo de texto, tivemos o seguinte:

```
Principais parâmetros e resultados do modelo:  
Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100  
O método de minimização utilizado foi o de gradiente descendente com uma  
taxa de aprendizado igual a 0.00100  
O número total de iterações para o treino do modelo foi de 800  
Não houve regularização  
O custo inicial foi de 5672956.1067, e terminou com 3538.3428  
O tempo total de execução do modelo foi de 58.478 segundos
```

O gráfico de custo por iteração para este exemplo foi o seguinte:



Curva de Custos do Gradiente Descendente não Regularizado

O top 10 de melhores e piores filmes foi o seguinte:

```
As melhores recomendações para o usuário foram as seguintes:  
Filme 444 Blob, The (1958) com nota 10.0  
Filme 701 Wonderful, Horrible Life of Leni Riefenstahl, The (1993) com nota 10.0  
Filme 581 Kalifornia (1993) com nota 9.9  
Filme 583 Romeo Is Bleeding (1993) com nota 9.9  
Filme 1477 Nightwatch (1997) com nota 9.9  
Filme 318 Schindler's List (1993) com nota 9.9  
Filme 1394 Swept from the Sea (1997) com nota 9.9  
Filme 473 James and the Giant Peach (1996) com nota 9.9  
Filme 501 Dumbo (1941) com nota 9.9  
Filme 449 Star Trek: The Motion Picture (1979) com nota 9.9  
  
E as piores recomendações para o usuário foram as seguintes:  
Filme 1172 Women, The (1939) com nota -0.0  
Filme 1622 Paris, France (1993) com nota -0.0  
Filme 1658 Substance of Fire, The (1996) com nota -0.0  
Filme 1001 Stupids, The (1996) com nota -0.1  
Filme 1442 Scarlet Letter, The (1995) com nota -0.1  
Filme 1549 Dream Man (1995) com nota -0.1  
Filme 1135 Doors, The (1991) com nota -0.1  
Filme 933 Funeral, The (1996) com nota -0.1  
Filme 986 Turbulence (1997) com nota -0.1  
Filme 1667 Next Step, The (1995) com nota -0.1
```

Mais uma vez tivemos um resultado satisfatório na questão de recomendação, porém, observamos novamente o problema de notas saindo da escala inicialmente estabelecida, corroborando ainda mais fortemente para a ideia de que filmes com poucas notas acabam por gerar resultados “ruins” quando treinamos o modelo. É interessante perceber que este foi o segundo exemplo mais rápido e o segundo com o menor custo final.

Por fim, conseguimos concluir que todas as implementações foram efetivas em sua tarefa de recomendar filmes para o usuário, algumas possuem vantagens, como por exemplo a implementação de gradiente conjugado regularizado que é muito rápida na execução do processo, em compensação, a implementação do gradiente conjugado não regularizado gerou o menor custo final de todos, ou seja, obteve a curva final que melhor se adequava aos dados. Importante ressaltar também que observamos um fenômeno que teoricamente já era previsto: o gradiente conjugado converge para um custo menor mais rapidamente que o gradiente descendente.

3.5 Busca Por Filmes:

Acabamos por implementar em nosso modelo uma busca simples que permite ao usuário averiguar se um dado filme se encontra na lista ofertada inicialmente, além disso, é possível realizar a busca por um dado ano, e assim visualizar todos os filmes que tiveram seu lançamento no mesmo.

A seguir, um exemplo da busca por nome do filme:

```
Nome do filme que se quer buscar: Toy Story  
  
Sim, o filme Toy Story, lançado em 1995, está na lista com uma nota média de 3.9/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Blade Runner  
  
Sim, o filme Blade Runner, lançado em 1982, está na lista com uma nota média de 4.1/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Star Wars  
  
Sim, o filme Star Wars, lançado em 1977, está na lista com uma nota média de 4.4/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Akira  
  
Sim, o filme Akira, lançado em 1988, está na lista com uma nota média de 3.4/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Matrix  
  
Não, o filme Matrix não está na lista  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 2
```

Como é possível averiguar, o usuário pode fazer o número de buscas que quiser, assim, se o filme está na lista, é mostrado seu ano de lançamento e nota média, caso contrário, o usuário é informado da ausência do filme.

Para um caso de busca por ano, temos o seguinte exemplo:

```
Gostaria de verificar todos os filmes que saíram em um dado ano?  
1 para sim e 2 para não: 1  
Ano que se quer verificar: 1990  
  
No ano 1990 saíram os seguintes filmes:  
-----  
Home Alone  
-----  
Dances with Wolves  
-----  
GoodFellas  
-----  
Nikita (La Femme Nikita)  
-----  
Cyrano de Bergerac  
-----  
Die Hard 2  
-----  
Hunt for Red October, The  
-----  
Ghost  
-----  
Amityville Curse, The  
-----  
Miller's Crossing  
-----  
Grifters, The  
-----  
Paris Is Burning  
-----  
Rosencrantz and Guildenstern Are Dead  
-----  
Pump Up the Volume  
-----  
Pretty Woman  
-----  
Days of Thunder  
-----  
Tie Me Up! Tie Me Down!  
-----  
Trust  
-----  
Young Guns II  
-----  
Marked for Death  
-----  
Every Other Weekend  
-----  
I, Worst of All (Yo, la peor de todas)  
-----  
American Dream  
-----  
King of New York  
-----  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
Ano que se quer verificar: 2000  
Nenhum filme da lista saiu em 2000
```

Como é possível visualizar, o usuário realiza a busca pelo ano e o programa retorna todos os filmes que lançaram no mesmo, para o caso de nenhum filme ter saído em dado ano, o usuário tem como retorno o aviso de que nenhum filme da lista saiu no ano buscado.

É possível também realizar a busca por um filme após obtermos as notas preditas pelo modelo, o processo é semelhante a busca anteriormente retratada:

```
Gostaria de buscar a nota predita pelo modelo de algum filme em específico?  
1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Toy Story  
  
O filme Toy Story, lançado em 1995, teve sua nota prevista de 4.5/5,  
enquanto através de média simples, sua nota foi 3.9/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Star Wars  
  
O filme Star Wars, lançado em 1977, teve sua nota prevista de 5.0/5,  
enquanto através de média simples, sua nota foi 4.4/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 1  
  
Nome do filme que se quer buscar: Blade Runner  
  
O filme Blade Runner, lançado em 1982, teve sua nota prevista de 4.8/5,  
enquanto através de média simples, sua nota foi 4.1/5  
  
Gostaria de uma nova consulta? 1 para sim e 2 para não: 2
```

E por fim, é possível visualizar o top “n” dos melhores e piores filmes:

```
Gostaria de visualizar um 'top' dos melhores e piores filmes  
1 para sim e 2 para não: 1  
  
Qual seria o 'top' que se quer visualizar? 5  
  
Melhores recomendações para você:  
  
Nota predita 5.4 para o filme 285 Secrets & Lies (1996)  
Nota predita 5.1 para o filme 169 Wrong Trousers, The (1993)  
Nota predita 5.1 para o filme 100 Fargo (1996)  
Nota predita 5.0 para o filme 408 Close Shave, A (1995)  
Nota predita 5.0 para o filme 50 Star Wars (1977)  
  
Piores recomendações para você:  
  
Nota predita 1.0 para o filme 439 Amityville: A New Generation (1993)  
Nota predita 1.0 para o filme 784 Beyond Bedlam (1993)  
Nota predita 1.0 para o filme 830 Power 98 (1995)  
Nota predita 1.0 para o filme 1320 Homage (1995)  
Nota predita 1.0 para o filme 1348 Every Other Weekend (1990)
```

Com essas implementações conseguimos ofertar diferentes possibilidades para visualização dos filmes tratados no modelo e suas características por parte do usuário, dando também uma função extra ao código desenvolvido.

3.6 Processo de Salvamento de Dados do Modelo:

Ofertamos ao usuário a possibilidade do mesmo salvar os dados mais importantes do modelo para utilização futura, para tal, salvamos dados como o número de iterações, tipo de minimização utilizada e entre outros em um arquivo de texto, além disso, guardamos a matriz X e a matriz Θ finais em arquivos CSV:

```
Gostaria de salvar as principais informações deste programa em um arquivo txt  
e a matriz X e theta em arquivos CSV? 1 para sim e 2 para não: 1  
  
Endereço para salvar os arquivos (sem o nome dos mesmos):  
C:\Users\USUARIO\Desktop\projeto 3  
  
Seus arquivos foram salvos no diretório C:\Users\USUARIO\Desktop\projeto 3  
com o nome de Resultado_recomendação.txt, Matriz_X.csv e Matriz_theta.csv :)
```

Um exemplo de arquivo de texto é o seguinte:

```
Resultado_Recomendação - Bloco de Notas  
Arquivo Editar Formatar Exibir Ajuda  
Principais parâmetros e resultados do modelo:  
Para a dimensão de colunas dos vetores theta, foi utilizado um valor de 100  
O método de minimização utilizado foi o de gradiente conjugado  
O número total de iterações para o treino do modelo foi de 606  
Houve regularização e o valor de lambda utilizado foi 10.00000  
O custo inicial foi de 6253040.6715, e terminou com 34027.4697  
O tempo total de execução do modelo foi de 53.686 segundos  
  
As melhores recomendações para o usuário foram as seguintes:  
Filme 285 Secrets & Lies (1996) com nota 5.4  
Filme 169 Wrong Trousers, The (1993) com nota 5.1  
Filme 100 Fargo (1996) com nota 5.1  
Filme 408 Close Shave, A (1995) com nota 5.0  
Filme 50 Star Wars (1977) com nota 5.0  
  
E as piores recomendações para o usuário foram as seguintes:  
Filme 439 Amityville: A New Generation (1993) com nota 1.0  
Filme 784 Beyond Bedlam (1993) com nota 1.0  
Filme 830 Power 98 (1995) com nota 1.0  
Filme 1320 Homage (1995) com nota 1.0  
Filme 1348 Every Other Weekend (1990) com nota 1.0
```

Exemplo de Arquivo de Texto Gerado Pelo Programa

E um exemplo de CSV é o seguinte:

	A	B	C	D	E	F	G	H	I	J	K	L
1	-0.17703062758342616	-0.19703591198160111	0.09788777676704008	-0.23441088169808597	-0.04920680172148595	-0.40798097						
2	0.07281729419650655	0.010804865781338971	-0.16375074668060263	-0.006945545922449953	0.08070756208555385	0.03105712						
3	0.32414244104927	0.0852508904651589	-0.2238134972747899	0.06697507378790714	-0.10332790839706815	0.1376795789013171						
4	0.01074461121122574	0.03073416944569598	0.08748028251261616	-0.19375946043565412	0.06685578937909424	0.0699195568						
5	-0.09004681115807787	-0.1242598814465219	0.16508383491711	-0.3224569771741822	0.018674703856355845	0.28193723215943						
6	-0.0879780025839811	-0.09668492603823159	0.04710043032555401	0.30400421416463286	0.002909434280004368	-0.084256509						
7	0.008763915294545888	0.06432390459048759	-0.0731325192139096	-0.12091487532154892	0.19544174058172586	-0.4715199546						
8	0.00398126437924785	0.005558308943250655	-0.0379202850524686	-0.07726159693321331	-0.06029251601047261	-0.046920846						
9	0.06474152766025411	0.039327863602653904	-0.03049240806491589	-0.05222726098225693	0.041458727291793174	-0.03516898						
10	0.11341256142586434	0.2389803494036651	0.07722404274173685	-0.011764427936757593	0.02374592120928335	0.07262185447						
11	0.21811281915725877	0.07057778892820968	0.0008029624753965817	-0.23377020910207919	0.0934954562439805	0.1711647345						
12	0.0032391637807129585	0.043803964648048165	-0.01635943541837517	0.08315491889563303	0.03155394667316062	-0.22243412						
13	0.4627764898102966	-0.12831446160610702	0.2032366997012242	0.065886599562559	-0.003410306922469424	-1.1025735951326						
14	-0.08013577454250403	-0.06269013900401355	0.29137735413040367	-0.05525146443332284	0.2032512209802253	0.07183337040						
15	-0.30724921873691985	-0.3121940499885126	-0.20276837410603407	0.025127852109709206	-0.20530045570260153	0.263216142						
16	0.030100095216973603	0.16280588001689897	0.13048785713206829	-0.24035636025274287	0.2972489816381008	-0.07079432318						
17	-0.03869576616519791	-0.07452556425631693	-0.0016930351765003068	-0.09382164670433076	-0.14957719274097778	-0.044745						

Arquivo CSV Com a Matriz Theta

4 Conclusão

A partir deste projeto conseguimos desenvolver um amplo conhecimento, teórico e prático, acerca dos temas abordados, a “Análise de Componentes Principais” e os “Sistemas de Recomendação”, duas áreas amplas e com inúmeras aplicações em tempos modernos. A partir das implementações em Python obtivemos uma visualização completa do funcionamento complexo e preciso das duas áreas tratadas, foi algo extremamente importante para nosso desenvolvimento como criadores e modeladores de códigos.

Por fim, ficamos satisfeitos com os resultados obtidos, sempre que pudemos adicionamos funções e ideias extras que pudessem complementar o código de forma dinâmica de maneira a permitir uma ampla gama de interações e análises por parte do usuário.

5 Bibliografia:

- [1] <https://www.onespire.net/news/history-of-recommender-systems/>
- [2] <https://www.losangelesblade.com/2021/10/11/tiktoks-algorithm-leads-users-from-transphobic-to-far-right-rabbit-holes/>
- [3] <https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/>
- [4] <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>
- [5] <https://setosa.io/ev/principal-component-analysis/>
- [6] <https://www.datacamp.com/community/tutorials/recommender-systems-python>
- [7] <https://www.smarthint.co/en/ai-product-recommendation-engine/>
- [8] <https://growthhouse.com.br/blog/o-que-voce-precisa-saber-sobre-os-algoritmos-das-redes-sociais/>
- [9] <https://revistaseletronicas.pucrs.br/index.php/revistafamecos/article/view/34074>
- [10] <https://www.acervodigital.ufpr.br/handle/1884/67816>
- [11] <https://www.analyticssteps.com/blogs/what-are-recommendation-systems-machine-learning>
- [12] https://www.sas.com/en_hk/insights/articles/big-data/recommendation-systems.html
- [13] Machine Learning - Stanford University (Coursera)
- [14] Aulas e slides do professor da disciplina MS960, João Batista Florindo