Universidade Estadual de Campinas

Institute of Mathematics, Statistics and Scientific Computing

(IMECC - Unicamp)

Projeto Computacional 4:

Aplicação dos métodos da *distribuição Gaussiana* e da *filtragem colaborativa* e seus resultados

Aluno : Gabriel Borin Macedo || RA : 197201
Aluno : Matheus Araujo Souza || RA : 184145
Aluno : Raul Augusto Teixeira || RA : 205177

Janeiro 2021

Resumo

Modelos de recomendação de conteúdo, principalmente para plataformas onlines como *facebook*, *instagram*, *netflix*, plataformas de *e-commerce*, dentre outros exemplos, utilizando estratégias com *deep learning* [9] se tornou comum e praticamente vital para atrair novos usuários e ainda manter seus antigos clientes utilizando seus serviços. Além disso, também é comum criar modelos automatizados com *deep learning* para detecção de alguma anomalia [5]. Entretanto, ambas estratégias necessitam de um número elevado de dados para atingir um resultado satisfatório. Por isso, outros modelos mais simples tais como a *distribuição Gaussiana* e a *filtragem colaborativa* são excelentes alternativas que resolvem esse problema.

A partir disso, esse problema visa o estudo dos métodos da *distribuição Gaussiana* e a *filtragem colaborativa* com um breve resumo de como esse modelos funcionam, um teste prático de cada modelo que foram implementados utilizando a linguagem *python* e uma discussão dos resultados obtidos.

1 Estrutura Matemática

1.1 Detecção de anomalias utilizando distribuição Gaussiana

Esse modelo consiste em detectar anomalias em um conjunto de dados X_{teste} com base em uma distribuição de probabilidade da forma

$$y = \begin{cases} 1 & \text{se } p(x) < \epsilon \text{ (caso anômalo)} \\ 0 & \text{caso contrário (caso normal)} \end{cases}$$

onde ϵ é um *threshold* escolhido pelo usuário. Além disso, o conjunto de dados para treinamento, teste e validação devem ser compostos de majoritariamente de dados sem anomalias, pois esse modelo possui uma maior precisão para esse casos e como anomalias são casos que não deveriam ocorrer com frequência. Por fim, iremos analisar dois tipos de modelos : um onde as variáveis tem pouca correlação e outro onde as variáveis possuem uma alta correlação

1.1.1 Distribuição Gaussiana com pouco correlação

Dizemos que $x_i \in X^{(j)}$ e que segue uma distribuição gaussiana (normal) com média μ_i e variância σ_i^2. Ou seja, para cada variável x_i do dado $X^{(j)}$ do conjunto de dados, temos que $x_i \longrightarrow \eta(\mu_i, \sigma_i^2)\ \forall\ i \in [1, m] \subset N$, com m sendo a quantidade de variáveis para cada dado. Dessa forma, temos que

$$p(x_i : \mu_i, \sigma_i^2) = \frac{1}{\sigma_i \sqrt{2}} e^{-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}}$$

Onde $p(x_i : \mu_i, \sigma_i^2)$ indica que $p(X^{(j)})$ é parametrizado por μ_i e σ_i. Já esses parâmetros são obtidos a partir do treinamento da forma

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_i^{(j)} - \mu_i)^2$$

Assim, a probabilidade de uma anomalia para um dado é da forma

$$p(x^{(j)}) = \prod_{i=1}^{m} p(x_i^{(j)} : \mu_i, \sigma_i^2)$$

Um adendo é que todo esse processo pode ser vetorizado para se obter uma maior agilidade. Basta apenas vetorizar os vetores $x^{(j)}, \mu_i$ e σ_i^2.

1.1.2 Distribuição gaussiana multivariada (alta correlação)

Esse modelo é utilizado quando algum dos atributos são fortemente correlacionados. Nesse caso, não podemos modelar $p(x^{(1)}), p(x^{(2)}), \ldots, p(x^{(n)})$.Dessa forma, devemos analisar todas as variáveis de uma só vez.

Para $x \in R^n$, definimos $\mu \in R^n$ e $\Sigma \in R^{n \text{ x } n}$ (matriz de covariância). Nesse caso, temos que

$$p(x^{(j)}) = p(x^{(j)} : \mu, \Sigma^2) = \frac{1}{(2\pi)^{(n/2)}(\det(\Sigma))^{1/2}} \exp\left(-\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}\right)$$

Onde Σ é a mesma matriz de covariância vista no método PCA em [2]. Além disso, temos que μ e Σ são descritos da forma

$$\mu = \frac{1}{m}\sum_{j=1}^{m} x^{(j)}$$

$$\sigma = \frac{1}{m}\sum_{j=1}^{m}(x^{(j)} - \mu)(x^{(j)} - \mu)^T$$

Assim, basta agora calcular a expressão $p(x)$ descrita anteriormente e analisarmos em comparação com o *threshold*. Por fim, se Σ se aproximar por uma matriz diagonal, é possível utilizar a abordagem descrita na sessão anterior que também é eficiente.

1.1.3 Avaliação e análise de erro do modelo

Além da definição de y e de como é separado os dados para treinamento, teste e validação do modelo. São escolhidas outras estratégias para análise do erro como a *matriz de confusão* [1] e $F_1-score$ [8]. Também é possível utilizar o conjunto de validação para escolher um ϵ e os valores de atributos ideias.

Caso um atributo não esteja na escala gaussiana, fazemos transformações de tal forma que fiquem nesta escala. Um dos exemplos de transformações é tomar x sendo $\log(x), \log(x+C), x^{1/2}, x^{1/3}, \ldots$, onde C é uma constante real, ou seja $C \in R$.

Durante a análise, são criados atributos nos exemplos classificados erroneamente. Assim, queremos encontrar os valores de $p(x^{(j)})$ nos quais $p(x^{(j)})$ seja grande para exemplos sem anomalias e pequeno para exemplo anômalos. Um problema que é mais comum é encontra um valor $p(x^{(j)})$ no qual seja comparável para exemplos normais e anômalos. Dessa forma, são criados atributos que apresentem valores que são excepcionalmente grandes ou pequenos de anomalia. Com isso, teremos uma classificação mais precisa.

1.2 Sistema de recomendação baseado em conteúdo

A ideia desse modelo é criar um algoritmo de regressão linear para cada usuário. Assim, é definido as seguintes variáveis

$$\begin{cases} n_\mu & \text{Número de usuário} \\ n_m & \text{Número de conteúdo (por exemplo, filmes)} \\ r(i,j) & \text{1 se o usuário } j \text{ deu nota ao filme } i \text{ e 0 caso contrário} \\ y^{(i,j)} & \text{Nota do usuário para o filme } i \text{ (definido apenas se r(i, j) = 1)} \end{cases}$$

Dessa forma, é possível criar um vetor de parâmetros para cada usuário e um vetor de atributos para cada filme. Assim, é definido os seguintes três parâmetros

$$\begin{cases} \theta^{(j)} & \text{Vetor de parâmetros para o usuário } j \\ X^{(i)} & \text{Vetor de atributos para o filme } i \\ m^{(j)} & \text{Número de filmes avaliados pelo usuário } j \end{cases}$$

Então, para o usuário j e filme i, a nota predita é dada pela forma

$$y_{\text{pred}}^{(i,j)} = (\theta^{(j)})^T (X^{(i)})$$

Como na regressão linear, convencionamos $X_0^{(i)} = 1$ de modo que $X^{(i)} \in R^{n+1}$ e $\theta^{(j)} \in R^{n+1}$. Esse procedimento é feito para facilitar o produto interno. Em sistemas de recomendação, os parâmetros $\theta^{(j)}$ são aprendidos da forma

$$J((\theta^{(j)})) = \arg\min_{\theta^{(j)}} \left(\frac{1}{2} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right)$$

Onde o fator $\frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$ é o fator de regularização. Entretanto, temos n_μ usuários, portanto temos que a função de perda é da forma

$$J(\theta) = \arg\min_{\theta^{(1)},\theta^{(2)},\ldots,\theta^{(n_\mu)}} \left(\sum_{j=1}^{n_\mu} \left[\frac{1}{2} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right] \right.$$

Com isso, podemos minimizar utilizando a técnica do *gradiente descendente* da forma

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)}) X_k^{(i)} \right) \text{ (se k = 0)}$$

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

Onde α é escolhido de forma arbitrária e os atributos são previamente calculados por um especialista.

1.2.1 Método do *gradiente conjugado* para atualização dos pesos da rede

O princípio desse método é resolver um sistema linear de equações ou que apresentam equações quadráticas.

Assim, a função que queremos minimizar será a função $J(\Theta)$. Entretanto, iremos calcular a função de custo em intervalos de t-dados denominados como *mini-batch* (subconjunto dos dados de treinamento que contém t dados). Assim, dado um subconjunto de dados X, a função de perda nesse subconjunto é da forma

$$L(\Theta) = \frac{1}{|X|} \sum_{x \in X} l(x^{(i)}, \Theta, j)$$

onde $|X|$ é a cardinalidade do subconjunto de dados $|X|$ (em outras palavras, a quantidade de dados em X) e $l(x, \Theta)$ é a função de perda para um único dado descrita pela equação

$$l(X^{(i)}, \Theta, j) = -[y^{(i,j)} \log (y_{\text{pred}}^{(i,j)}(X^{(i)}))(1 - y^{(i,j)}) \log (1 - y_{\text{pred}}^{(i,j)}(X^{(i)}))] \quad (1)$$

Entretanto, antes de realmente começar o algoritmo. É necessário utilizar um algoritmo que irá ajudar no processo do cálculo do *gradiente conjugado*. A seguir, é definido o cálculo da matriz pré-condicional que será da forma

Algorithm 1: Pseudo - Código para o cálculo da matriz pré-condicionada (MPC)

Dado Θ, Θ_{ant}, $\nabla L(\Theta)$, $\nabla L_{ant}(\Theta_{ant})$, e o passo t
if $t = 0$ **then**
  $H_{diag} = Id$ (Matriz identidade)
else
  $y = \nabla L(\Theta) - \nabla L_{ant}(\Theta_{ant})$
  $s = \Theta - \Theta_{ant}$
  $H_{diag} = H_{diag} + \frac{Diag(yy^T)}{y^T s} - \frac{H_{diag}\ Diag(ss^T)\ H_{diag}^T}{s^T\ H_{diag}\ s}$
$\Theta_{ant} = \Theta$
$\nabla L_{ant}(\Theta_{ant}) = \nabla L(\Theta)$
retorne $H_{diag}^{-1}, \Theta_{ant}, \nabla L_{ant}(\Theta_{ant})$

Dessa forma, temos o algoritmo do *gradiente conjugado* que será da forma

Algorithm 2: Pseudo - Código para o cálculo do *gradiente conjugado*.

$M^{-1}, \Theta_{ant}, \nabla L_{ant}(\Theta_{ant}) = \text{MPC}(\Theta, 0, -\nabla L(\Theta), 0, 0)$
$s = M^{-1}r$
$d = s$
$\delta_{novo} = r^T d$
$\nabla L_{ant}(\Theta_{ant}) = \nabla L(\Theta)$
for $t = 0{:}t_{max}$; $t = t + 1$ **do**
  $\Theta = \Theta + \epsilon d$
  $r = -\nabla L(\Theta)$
  $\delta_{ant} = \delta_{novo}$
  $\delta_{aux} = r^T s$
  $M^{-1}, \Theta_{ant}, \nabla L_{ant}(\Theta_{ant}) = \text{MPC}(\Theta, \Theta_{ant}, \nabla L(\Theta), \nabla L_{ant}(\Theta_{ant}), t)$
  $s = M^{-1}r$
  $\delta_{novo} = r^T s$
  if *update = PolakRibiere [7]* **then**
    $\beta = \dfrac{\delta_{novo} - \delta_{aux}}{\delta_{ant}}$
  else
    $\beta = \dfrac{\delta_{novo}}{\delta_{ant}}$(atualização via *FletcherReaves* [6])
  if $\beta < 0$ **then**
    $\beta = 0$
  else
    $\beta = 1$
  $d = s + \beta d$

A grande vantagem desse algoritmo está em sua convergência em menos passos comparado com o *gradiente descendente*. Além disso, o *gradiente conjugado* procura o ponto mínimo através de uma reta, que em certos casos, ajuda na convergência.

Entretanto, pelo fato de que para cada movimento, o gradiente não retorna ao passo anterior (caso o valor atual seja pior comparado com o valor antigo) e pelo alto custo computacional, devido ao cálculo de matrizes inversas e de outras operações matemáticas custosas. O método do *gradiente conjugado* é inferior comparado ao *gradiente descente*.

1.2.2 Filtragem Colaborativa

Neste caso, utilizamos os comportamentos dos usuários para descobrir os atributos e assim fazer recomendações de filmes para pessoas que possivelmente assistirão esses filmes. Dessa forma, se conhecermos os vetores dos parâmetros $\theta^{(j)}$, então podemos obter o valor $X^{(i)}$ para minimizar o erro da predição $y_{\text{pred}}^{(i,j)} = (\theta^{(j)})^T X^{(i)} \approx c_{ij}$, onde c_{ij} é a nota do score. Com isso, iremos aplicar a minimização para $X^{(i)}$ ao invés de $\theta^{(j)}$. Portanto, teremos a seguinte função de perda para um único usuário

$$J((X^{(i)})) = \arg\min_{X^{(i)}} \left(\frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} ((x_k^{(i)})^2) \right] \Bigg)$$

Porém, temos n_m filmes e os vetores de atributos são aprendidos simultaneamente pela fórmula

$$J(X) = \arg\min_{X^{(1)}, X^{(2)}, \ldots, X^{(n_m)}} \left(\sum_{i=1}^{n_m} \left[\frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2 \right] \right)$$

Dessa forma, podemos juntar as expressões $J(X)$ e $J(\Theta)$ em uma única função de perda, deixando esse método mais optimizado. Portanto, a função de perda que será utilizada nesse relatório é da seguinte forma

$$\begin{aligned} J(X^{(1)}, X^{(2)}, \ldots, X^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_\mu)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T X^{(i)} - y^{(i,j)})^2 \\ + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_\mu} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \end{aligned}$$

Onde vamos minimizar a função de perda

$$J(X, \Theta) = \arg\min_{\substack{X^{(1)}, X^{(2)}, \ldots, X^{(n_m)} \\ \theta^1, \theta^1, \ldots, \theta^{n_\mu}}} J(X^{(1)}, X^{(2)}, \ldots, X^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_\mu)})$$

Como $X^{(i)}$ depende de $\theta^{(j)}$, iremos chutar um valor inicial $\theta^{(j)}$ e encontrar um valor $x^{(i)}$ e depois $\theta^{(j)}$ e assim por diante. Com isso, todos os usuários colaboram para a indicação de um determinando filme, apenas atribuindo nota a eles.

Além disso, calculamos $J(\theta^{(j)})$ e depois $J(X^{(i)})$ e aplicados o método do *gradiente descendente* ou *gradiente conjugado*, tanto para $X^{(i)}$ e $\theta^{(j)}$ (nesse caso, o gradiente *aprende* tanto para $x^{(i)}$ e $\theta^{(j)}$).

Em resumo, iniciamos $X^{(i)}$ e $\theta^{(j)}$ com valores aleatórios e pequenos. Após isso, minimizamos as funções $J(\theta^{(j)})$ e depois $J(X^{(i)})$. Por fim, atualizamos os valores de $X^{(i)}$ e $\theta^{(j)}$ pelo método do *gradiente descendente*

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

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

Também é possível utilizar outros métodos para calcular esses valores. E dessa forma, esse algoritmo indica filmes para os usuários que provavelmente assistirão essa recomendação. Um adendo é que nessa abordagem de filtragem colaborativa, não consideramos que $X_0^{(i)} = 1$.

1.2.3 *Low Rank Matrix Factorization* (LRMF)

Esse método consiste em utilizar o método da filtragem colaborativa, porém de forma matricial. Assim, seja $Y \in R^{n_m \text{ x } n_\mu}$ a seguinte matriz

$$Y \approx \begin{bmatrix} (\theta^{(1)})^T(X^{(1)}) & (\theta^{(2)})^T(X^{(1)}) & \dots & (\theta^{(n_\mu)})^T(X^{(1)}) \\ (\theta^{(1)})^T(X^{(2)}) & (\theta^{(2)})^T(X^{(2)}) & \dots & (\theta^{(n_\mu)})^T(X^{(2)}) \\ \vdots & \vdots & \dots & \vdots \\ (\theta^{(1)})^T(X^{(n_\mu)}) & (\theta^{(2)})^T(X^{(n_\mu)}) & \dots & (\theta^{(n_\mu)})^T(X^{(n_\mu)}) \end{bmatrix}$$

Com isso, definimos as matrizes $X \in R^{n_m \text{ x } m}$ com $(X^{(i)})^T$ em cada linha e $\Theta \in R^{n_\mu \text{ x } m}$ com $(\theta^{(j)})^T$ em cada linha da forma

$$X = \begin{bmatrix} (X^{(1)})^T \\ (X^{(2)})^T \\ \vdots \\ (X^{(n_m)})^T \end{bmatrix}$$

$$\Theta = \begin{bmatrix} (\theta^{(1)})^T \\ (\theta^{(2)})^T \\ \vdots \\ (\theta^{(n_\mu)})^T \end{bmatrix}$$

Dessa forma, a predição pode ser representada na forma $Y \approx Y_{\text{pred}} = X\Theta^T$
O nome dessa abordagem se deve ao fato de que a matriz Y tem posto baixo.

1.3 Filmes Relacionados

Dois filmes são considerados similares se $||X^{(i)} - X^{(j)}|| < \epsilon$ para o valor de ϵ escolhido durante a implementação. Assim, se queremos escolher os os filmes mais relacionados com o filme $X^{(i)}$, basta encontrar o filme $X^{(j)}$ com a menor norma subtraída de $X^{(i)}$ e que seja menor que ϵ

1.3.1 Preenchimento Da falta de Dados

Há casos em que os usuários não preenchem a nota de certos filmes ou poucos usuários deram nota para eles. Nesse caso, se considerarmos essas notas faltantes como zero, seria como se essas pessoas tivessem dado a nota 0 para os filmes, o que não seria um cenário realista.

Para evitar esse tipo de problema, é adotado a média dos filmes de cada usuário e fazemos o seguinte : seja M o vetor $R^{1 \text{ x } n_m}$ de média de cada usuário da forma

$$M = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_{n_m} \end{bmatrix}$$

Assim, temos que o vetor Y será da forma

$$Y \approx \begin{bmatrix} (\theta^{(1)})^T(X^{(1)}) - \mu_1 & (\theta^{(2)})^T(X^{(1)}) - \mu_1 & \dots & (\theta^{(n_\mu)})^T(X^{(1)}) - \mu_1 \\ (\theta^{(1)})^T(X^{(2)}) - \mu_2 & (\theta^{(2)})^T(X^{(2)}) - \mu_2 & \dots & (\theta^{(n_\mu)})^T(X^{(2)}) - \mu_2 \\ \vdots & \vdots & \dots & \vdots \\ (\theta^{(1)})^T(X^{(n_\mu)}) - \mu_{n_m} & (\theta^{(2)})^T(X^{(n_\mu)}) - \mu_{n_m} & \dots & (\theta^{(n_\mu)})^T(X^{(n_\mu)}) - \mu_{n_m} \end{bmatrix}$$

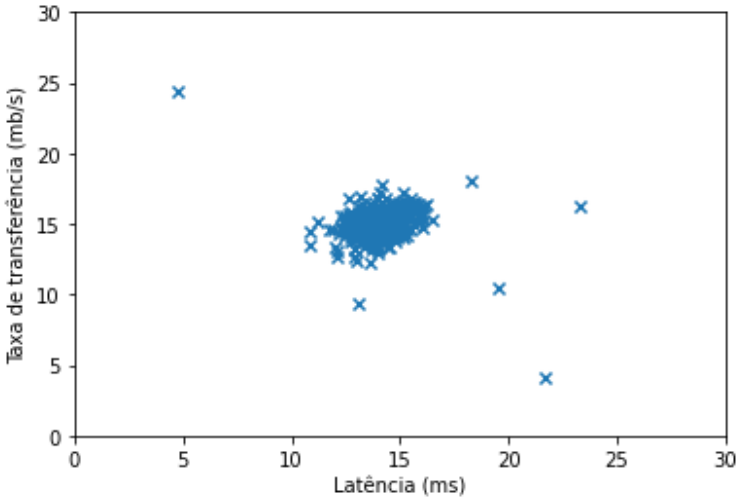
Dessa forma, para o usuário j e filme i, teremos que a nota será $y^{\text{pred}^{(i,j)}} = (\theta^{(j)})^T(X^{(i)}) - \mu_i$. Note que o valor μ_i só influência nos usuários que não atribuíram nota ao filme i.

2 Resultados Práticos

2.1 Detecção de anomalias utilizando distribuição Gaussiana

2.1.1 Parte A

Para a parte A, implementamos um *script* em linguagem *python* simples chamado *Parte1a.py*, no qual lê os dados de entrada do arquivo *dado1.mat* e plota os dados em um gráfico. O gráfico retornado é o gráfico abaixo.



2.1.2 Parte B

Implementamos um programa chamado *Parte1b.py* que possui duas funções: $estimateGaussian(X)$, que retorna a média μ e a matriz Σ de covariâncias dos dados apresentados, e a função $multivariateGaussian(X, \mu, \Sigma)$, que retorna um vetor de valores p com os valores da gaussiana em cada ponto de treinamento pertencente à matriz X. Seguimos as fórmulas das aulas teóricas de *Detecção de Anomalias* e *da aula prática de Detecção de Anomalias em Python* em [2]. Porém com uma modificação: na aula prática, é calculada uma gaussiana normal, e depois as variâncias são diagonalizadas para a gaussiana ser tratada como se fosse um caso específico da gaussiana multivariada.

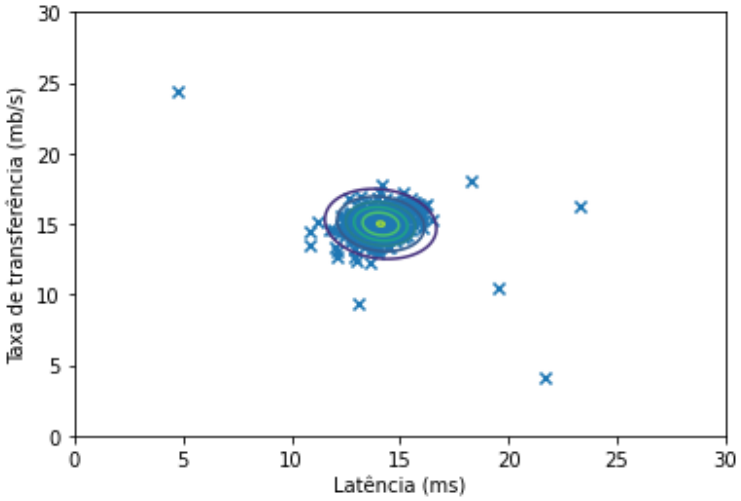
Para ajustar uma gaussiana multivariada aos dados do modelo, tivemos que implementar as fórmulas para o cálculo da matriz de covariância vistas em aula. Foi necessário implementar uma linha de código nova com uma formulação diferente das aulas, mas que realiza o mesmo cálculo da matriz de covariâncias:

$$\Sigma = \frac{1}{m}\Big((X-\mu)^T(X-\mu)\Big)$$

Além disso, Comparamos o valor da diagonal com o valor das variâncias caso fosse uma gaussiana normal, e os valores bateram. Com isso, notamos que a matriz sigma é simétrica, como esperado.

2.1.3 Parte C

Nesta parte, utilizamos o programa da parte B acrescido de alguns cálculos para realizar a plotagem de curvas de nível e os dados de X. O programa é chamado *Parte1c.py*, e o gráfico resultante está abaixo.



Podemos observar pelo gráfico uma característica peculiar das curvas de nível. Ficamos um pouco intrigados com o formato que tomaram, pois de acordo com os dados, deveríamos ter uma gaussiana em uma inclinação "mais perpendicular" à que obtemos. Porém, com uma análise mais atenta, pode-se perceber que as curvas de nível tomaram essa orientação devido ao fato de elas tentarem ajustar a gaussiana às anomalias também, o que não é comum de ocorrer de acordo com nossas aulas: normalmente, as anomalias são excluídas do conjunto de treinamento.

2.1.4 Parte D

Na parte D, implementamos o programa *Parte1d.py*, que utiliza a função de cálculo do ϵ e F_1 das aulas de *Detecção de Anomalias em Python*[2]. O programa ao final imprime esses valores. O resultado que obtivemos foi:

Melhor ϵ obtido por validação: $9.065769728392736e^{-05}$
Melhor F_1 na validação: 0.8750000000000001

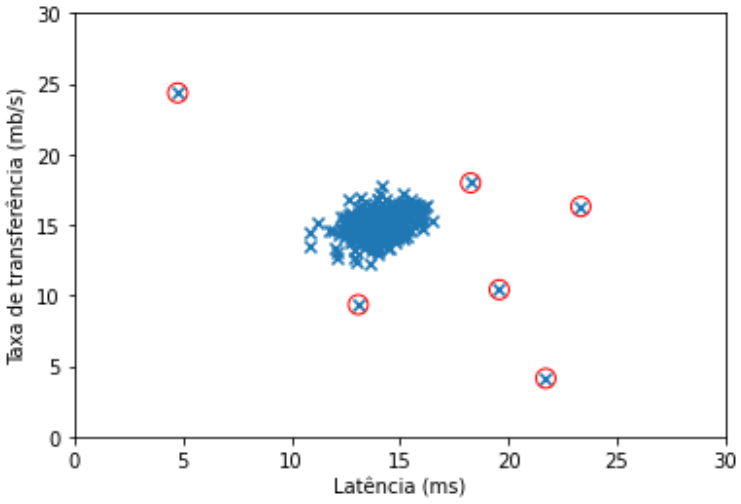
A execução do programa também retorna também um *Warning*:

RuntimeWarning: invalid value encountered in long_scalars prec = tp/(tp+fp)

Não sabemos muito bem o motivo desse *warning*, pois o programa roda normalmente e, conforme o próximo exercício (parte E), classificou bem os pontos anômalos.

2.1.5 Parte E

Para a Parte E, implementamos o programa *Parte1e.py* utilizando o programa anterior com o acréscimo de uma linha de código para determinar os pontos anômalos e também a plotagem desses pontos. Obtivemos o seguinte gráfico



Podemos ver que o algoritmo classificou 6 pontos como anômalos, e que eles estão bem distantes dos outros pontos, então o programa foi bem implementado e funciona bem.

2.1.6 Parte F

Para esta parte final, utilizamos o programa da parte E para fazer o programa *Parte1f.py*, apenas adicionando uma linha de código para calcular o número de anomalias e algumas outras linhas para imprimir os valores de ϵ, F_1, número de anomalias e a porcentagem de anomalias classificadas dentro dos exemplos de treinamento. O programa retornou o seguinte resultado

Melhor ϵ obtido por validação: $1.7538488712458777e^{-18}$
Melhor F_1 na validação: 0.5517241379310345
Numero de anomalias encontradas: 122
Fração de anomalias nos exemplos de treinamento: 12.2%

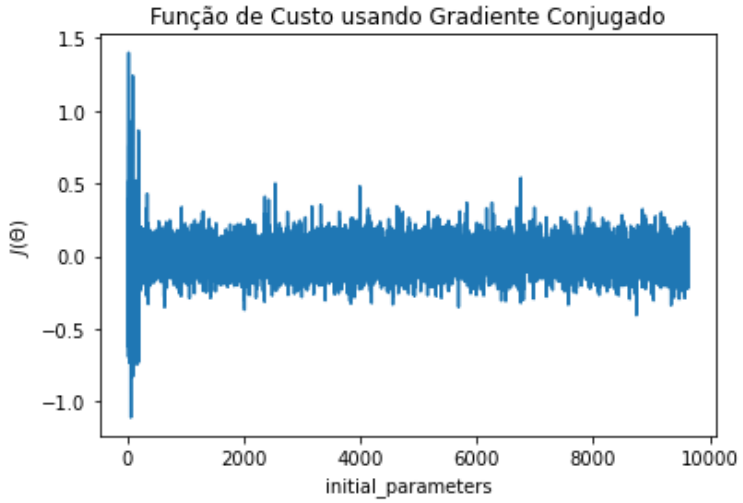
2.2 Filtragem Colaborativa

Nessa parte da resolução dos problemas propostos encontramos dificuldades na taxa de convergência usando apenas o método do minimize(*method='CG'*),

além da alta demora para resolução de apenas um pequeno grupo dos $[1682;:]$ dados ainda não tinha praticamente nenhuma convergência. Esse problema foi solucionado usando uma variante do método que usa 2 funções auxiliares no lugar de uma, que na realidade a função da nova seria fornecer a matriz das derivadas (*Jgrad*) para o *J_history = spopt.minimize(...)* da biblioteca *import scipy.optimize as spopt*. Assim, foi testado vários valores de α, λ, tol e para uma quantidade de filmes. A seguir, é possível visualizar alguns resultados

2.2.1 Melhores valores para $\alpha = 0.0001$, $\lambda = 15$, tol $= 10^{-15}$(tolerância de parada) para os dados reduzidos em $[20;:]$

Para essa escolha, foi possível obter os seguintes resultados

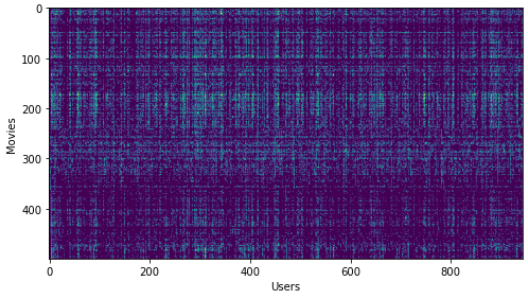


Com as melhores recomendações para o usuário.

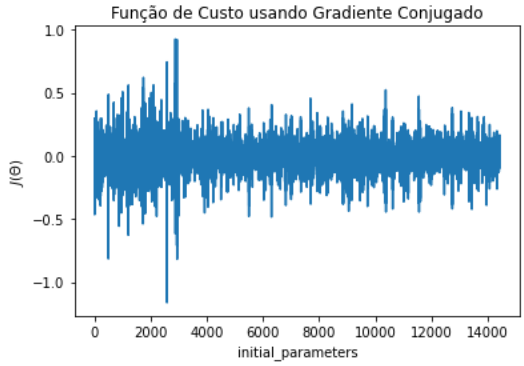
Nota predita 0.3 para o índice 8 *Babe (1995)*
Nota predita 8.1 para o índice 5 *Copycat (1995)*
Nota predita 7.7 para o índice 6 *Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)*
Nota predita 6.7 para o índice 18 *White Balloon (1995)*
Nota predita 5.3 para o índice 4 *Get Shorty (1995)*
Nota predita 5.1 para o índice 10 *Richard III (1995)*
Nota predita 4.2 para o índice 13 *Mighty Aphrodite (1995)*
Nota predita 3.9 para o índice 11 *Seven (Se7en) (1995)*
Nota predita 3.7 para o índice 9 *Dead Man Walking (1995)*
Nota predita 3.6 para o índice 20 *Angels and Insects (1995)*
—1.0185976028442383 seconds—

2.2.2 Melhores valores para $\alpha = 0.0001$, $\lambda = 25$, tol $= 10^{-15}$(tolerância de parada) para os dados reduzidos em $[500; :]$

Para esses valores, podemos observar como nossos dados estão inicialmente distribuídos, através do *color map* para saber como estão as distribuição de notas de cada filme:



Com a função de custo



Com as melhores recomendações para o usuário

Nota predita 9.8 para o índice 267 *unknown*
Nota predita

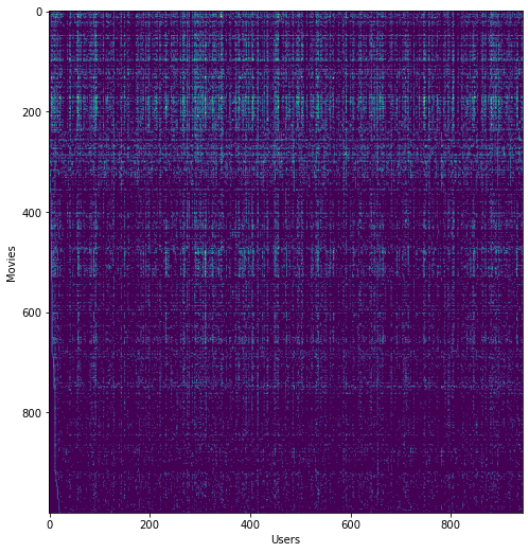
9.7

para o índice 23 *Taxi Driver (1976)*
Nota predita 9.4 para o índice 332 *Kiss the Girls (1997)*

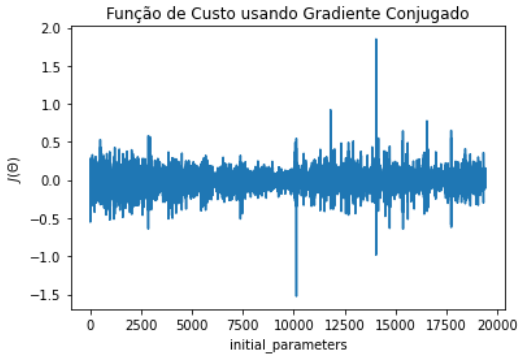
Nota predita 9.4 para o índice 277 *Restoration (1995)*
Nota predita 9.3 para o índice 156 *Reservoir Dogs (1992)*
Nota predita 9.3 para o índice 477 *Matilda (1996)*
Nota predita 9.3 para o índice 39 *Strange Days (1995)*
Nota predita 9.2 para o índice 437 *Amityville 1992: It's About Time (1992)*
Nota predita 9.0 para o índice 479 *Vertigo (1958)*
Nota predita 9.0 para o índice 384 *Naked Gun 33 1/3: The Final Insult (1994)*
—3.1825335025787354 seconds—

2.2.3 Melhores Valores para $\alpha = 0.0001$, $\lambda = 25$, tol $= 10^{-15}$(tolerância de parada) para os dados reduzidos em $[1000; :]$

Dessa forma, temos o *color map* para saber como estão as distribuição de notas de cada filme



Com a função de custo

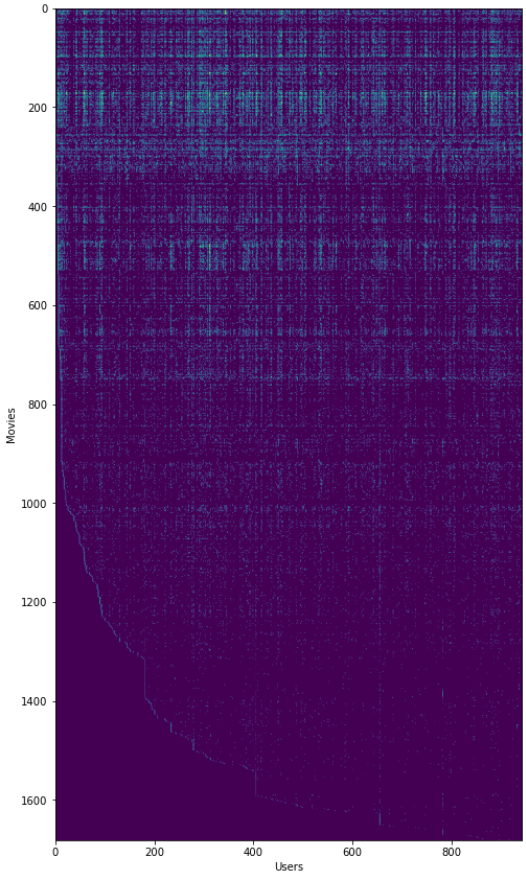


Com as melhores recomendações para o usuário

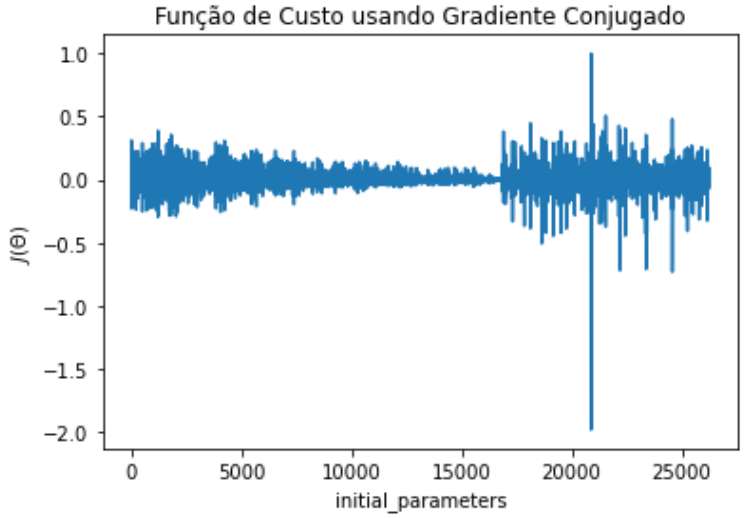
Nota predita 10.0 para o índice 727 *Immortal Beloved (1994)*
Nota predita 10.0 para o índice 404 *Pinocchio (1940)*
Nota predita 10.0 para o índice 71 *Lion King (1994)*
Nota predita 9.9 para o índice 497 *Bringing Up Baby (1938)*
Nota predita 9.9 para o índice 207 *Cyrano de Bergerac (1990)*
Nota predita 9.7 para o índice 608 *Spellbound (1945)*
Nota predita 9.7 para o índice 992 *Head Above Water (1996)*
Nota predita 9.7 para o índice 373 *Judge Dredd (1995)*
Nota predita 9.6 para o índice 353 *Deep Rising (1998)*
Nota predita 9.6 para o índice 589 *Wild Bunch, The (1969)*
—6.886363744735718 seconds—

2.2.4 Melhores Valores para $\alpha = 0.0001$, $\lambda = 40$, tol $= 10^{-15}$(tolerância de parada) para os dados reduzidos em $[1682;:]$

Abaixo, é possível visualizar todos os dados com a distribuição de notas para cada filme



Com a função de custo



Com as melhores recomendações para o usuário.

Nota predita 9.3 para o índice 389 *Black Beauty (1994)*
Nota predita 9.0 para o índice 38 *Net (1995)*
Nota predita 8.8 para o índice 486 *Sabrina (1954)*
Nota predita 8.8 para o índice 708 *Sex, Lies, and Videotape (1989)*
Nota predita 8.5 para o índice 1613 *Tokyo Fist (1995)*
Nota predita 7.9 para o índice 1531 *Far From Home: The Adventures of Yellow Dog (1995)*
Nota predita 7.7 para o índice 1594 *Everest (1998)*
Nota predita 7.7 para o índice 1175 *Hugo Pool (1997)*
Nota predita 7.6 para o índice 814 *Great Day in Harlem, A (1994)*
Nota predita 7.5 para o índice 185 *Psycho (1960)*
—7.533543825149536 seconds—

Nossa distribuição de notas não ficou na faixa de 0 a 5 como esperávamos, mas conseguimos reduzir e muito o tempo de convergência para o método do *Gradiente Conjugado*, também observamos uma estabilidade melhor e uma convergência mais estável da função de custo. creditamos que para resolver o primeiro ponto teríamos que encontrar um valor de α mais adequado para encontrar nossos parâmetros.

3 conclusão

Dessa forma, mostramos que esses modelos são úteis e apresentam resultados eficientes. Mesmo que com o método da *filtragem colaborativa* não apresentou um resultado esperado na faixa de notas entre 0 a 5, ainda sim é um problema que é facilmente resolvido adequando um melhor valor de α mais adequado ou até mesmo de λ ou tot. Portanto, esses problemas ainda são opções excelentes de modelos que podem ser usado quando há poucos dados ou quando a aplicação modelos de *deep learning* como as *Boltzmann machine* [4] em certos problemas não gera bons resultados.

References

[1] Ivo Düntsch and Günther Gediga. Confusion matrices and rough set data analysis. *Journal of Physics: Conference Series*, 1229:012055, May 2019.

[2] Joao B. Florindo. Ms960 : Tópicos especiais em processamento de imagens. `https://www.youtube.com/playlist?list=PLGwGFVrptiyRmFoDWxruNGgTu2cSPnTLX`, 2020.

[3] Aurelien Geron. Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build intelligent systems, 2017.

[4] Guido Montúfar. Restricted boltzmann machines: Introduction and review. *CoRR*, abs/1806.07066, 2018.

[5] G. Pang, Chunhua Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ArXiv*, abs/2007.02500, 2020.

[6] Barak Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6, 02 1970.

[7] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 3(R1):35–43, 1969.

[8] Nan Ye, Kian Ming A. Chai, Wee Sun Lee, and Hai Leong Chieu. Optimizing f-measures: A tale of two approaches. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1555–1562, Madison, WI, USA, 2012. Omnipress.

[9] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system. *ACM Computing Surveys*, 52(1):1–38, Feb 2019.