Departamento de Estatística
Universidade de Brasília

Juliano Cézar Sant'Anna
Matrícula: 110125894

Completamento de matrizes por decomposição em valores singulares via otimização convexa aplicado ao problema da *Netflix*

Brasília
2016

Juliano Cézar Sant'Anna

Completamento de matrizes por decomposição em valores singulares via otimização convexa aplicado ao problema da *Netflix*

Trabalho de Conclusão de Curso apresentado para obtenção do título de Bacharel em Estatística na Universidade de Brasília

Departamento de Estatística - Universidade de Brasília

Orientador: Donald Matthew Pianto

Brasília
2016

Resumo

Nesse estudo, foram realizadas previsões de valores faltantes, utilizando os dados da competição *Netflix Prize* de 2006. Com esse propósito, os dados foram representados como uma matriz esparsa e foi descrito um problema de otimização que utiliza a norma nuclear como relaxamento convexo, cuja solução é dada pelo algoritmo *Soft-Impute*. A partir de diferentes valores do parâmetro de regularização da norma nuclear, o algoritmo iterativamente faz a imputação de valores faltantes via *decomposição em valores singulares de limiar macio (Soft-thresholded SVD)*. O resultado é uma *SVD* que representa uma aproximação de posto baixo da matriz de avaliações da *Netflix*. Relatou-se também a economia de armazenamento resultante do uso de matrizes esparsas e o motivo de usar a *SVD* por meio de exemplos. Mostrou-se que o *Soft-Impute* é mais eficiente com o uso de *warm start* e que sua solução precisa passar por um pós processamento para obtenção de uma melhor previsão. Com a execução do algoritmo, apesar de não ter sido resolvido o problema convexo foram encontrados bons mínimos locais, resultando em um Erro Quadrático Médio de valor baixo.

Palavras-chave: *Netflix Prize*; *Soft-Impute*; Otimização convexa; Norma nuclear; *Soft-thresholded SVD*; Matriz esparsa; *Warm start*.

Lista de ilustrações

Sumário

1 Introdução

Os Sistemas de recomendação tem sido largamente utilizados nos últimos anos por diversas empresas de grande porte como *Amazon*, *Google* e *Netflix*. A ideia é filtrar informações de modo a prever o gosto de um usuário com relação a um item. Esse sistema tem duas técnicas principais: filtragem baseada em conteúdo e filtragem colaborativa. A primeira foca nas características de determinado item para recomendar outros itens parecidos, enquanto a segunda se baseia no comportamento anterior do usuário e em usuários semelhantes para fazer uma boa recomendação.

Uma aplicação das técnicas de filtro colaborativo é o problema da *Netflix*. A empresa de *streaming* utiliza sistemas de recomendação para fazer sugestões de filmes para seus usuários. Em 2006 a *Netflix* lançou uma competição com um prêmio de $\$1.000.000,00$ para quem conseguisse melhorar seu algoritmo de previsão de avaliações de filmes em 10% com relação a uma determinada medida de erro. A competição terminou em 2009.

Os dados fornecidos pela *Netflix*, podem ser representados na forma de uma matriz sendo que as linhas representam os usuários, as colunas os filmes e as observações são avaliações variando de 1 a 5, que os usuários deram a determinados filmes dependendo do quanto gostaram.

Uma vez que muitos dos filmes disponíveis na plataforma tem uma baixa proporção de avaliações, surge o interesse numa forma de previsão dessas avaliações. Mais especificamente, os dados fornecidos para a competição da *Netflix* foram aproximadamente 480 mil clientes, 18 mil filmes e 8.6 bilhões de entradas possíveis. Porém, na média, cada usuário avalia aproximadamente 200 filmes, então apenas 1.2% dos valores foram observados, o que torna essa matriz uma gigantesca matriz esparsa. Logo espera-se prever quais notas esses clientes dariam para determinados filmes levando em conta seu gosto, para que a empresa seja capaz de fazer uma sugestão adequada.

Esse problema se enquadra, em um tipo descrito por muitos autores (Candès e Recht, 2008; Candès e Tao, 2009; Rennie e Srebro, 2005) como *Matrix Completion problem*, o problema de completar matrizes. A ideia de acordo com Mazumder et al. (2010) é encontrar uma matriz $Z_{m\times n}$, uma aproximação, de alta dimensão, baseado nas poucas observações da matriz disponível. Eles consideram que Z pode ser bem representada por uma matriz de posto baixo.

Quando trabalhando com sistemas de recomendação, dizer que a matriz tem posto baixo justifica-se pela suposição comum na área de que são poucos os fatores que tem influência sobre o gosto de um determinado indivíduo. Por exemplo, uma estrutura de posto baixo pode sugerir que os filmes da *Netflix* podem ser agrupados em um pequeno número de gêneros e tipos de usuários.

Então partindo da suposição de posto baixo, Candès e Recht (2008), Candès e Tao

(2009), e Keshavan et al. (2009) mostraram teoricamente que sobre certas suposições nas entradas da matriz, localizações, e proporção de valores faltantes, é possível encontrar a matriz verdadeira escondida com grande acurácia através de um problema de otimização.

2 Metodologia

2.1 Otimização

Muitas situações que encontradas na área de estatística não possuem resposta analítica, o que leva a busca de uma alternativa numericamente. Na área de otimização utilizam-se algoritmos com o intuito de achar soluções numéricas para problemas de minimização ou maximização de funções. O objetivo é achar um ponto ótimo levando em consideração certas restrições, como um domínio específico, que são inerentes a cada questão em estudo.

Algoritmos de otimização são especialmente úteis ao lidar com grandes bases de dados, como em nosso tópico de estudo *The matrix-completion problem.*

Antes de apresentar o problema que foi resolvido, deseja-se introduzir algumas ideias que foram a base para chegar à otimização em questão. Sendo assim, essa seção segue conforme a exposição em Mazumder et al. (2010).

Seja $X_{m\times n}$ uma matriz com poucas entradas observadas, $Z_{m\times n}$ a matriz de aproximação e Ω o conjunto de índices dos valores observados de X. Tem-se o seguinte problema de otimização :

$$\begin{aligned} &\text{minimize} \quad \text{posto}(Z) \\ &\text{Sujeito a} \quad \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 \leq \delta \end{aligned} \tag{2.1}$$

no qual $\delta \geq 0$ é um parâmetro de regularização. Se for permitido que a matriz Z tenha o mesmo posto de X, é possível reproduzir a mesma, com perfeição através de Z, nesse caso δ seria igual a 0. Porém, sendo uma matriz de posto completo, ela não será adequada para a aplicação nos dados da *Netflix*, uma vez que será de difícil interpretação devido a grande quantidade de informações. Dito isso, para uma melhor compreensão da estrutura de Z, o ideal é ter uma matriz com posto pequeno. Para isso é preciso abrir mão de uma representação exata, e lidar com a situação de quão flexível deverá ser a representação de Z, o que irá determinar quão bem reproduzimos X. Como é a função posto que está sendo minimizada em (2.1), então esse é um problema não convexo, isto é, a função pode ter vários mínimos locais. Isso traz a dificuldade para descobrir o mínimo global, pois a otimização restrita a Ω, que representa apenas os índices dos valores observados, é combinatoriamente difícil (Srebro e Jaakkola, 2003). Essa expressão, em termos de complexidade de tempo, significa que não é possível resolver o problema com um algoritmo de tempo polinomial. Por outro lado, tratando-se de uma matriz X completamente observada, então a solução pode ser alcançada por meio de uma *decomposição em valores singulares truncada* de X.

Essa problemática da não convexidade leva a uma ligeira mudança em (2.1),

$$\begin{aligned} &\text{minimize} \quad \|Z\|_* \\ &\text{Sujeito a} \quad \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 \leq \delta \end{aligned} \tag{2.2}$$

na qual substitui-se a função posto pela norma nuclear tornando o problema convexo (Fazel, 2002). A vantagem de um problema convexo é que se existe um mínimo local, então ele também é mínimo global. Logo só pode existir um valor ótimo para o problema de minimização.

Em (2.2), $\|Z\|_*$ é a norma nuclear, que representa a soma dos valores singulares de Z. A otimização de (2.2) pode ser resolvida, por meio de algoritmos, para problemas pequenos de forma eficiente, entretanto se a matriz tiver dimensão alta, os algoritmos podem tornar-se exageradamente dispendiosos (Cai et al., 2008). O problema (2.2) pode ser reformulado sem restrições, na forma de Lagrange

$$\underset{Z}{\text{minimize}} \quad \frac{1}{2} \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 + \lambda \|Z\|_* \tag{2.3}$$

Aqui $\lambda \geq 0$ é o parâmetro de regularização controlando a norma nuclear da aproximação Z. Para λ pequeno existe um maior risco de excesso de ajuste (o que incorre em alta variância), se λ for grande, há um grande viés. Existe um mapeamento um a um entre $\delta \geq 0$ e $\lambda \geq 0$ sobre seus domínios ativos.

Para representar (2.3) foi adotada a notação de Cai et al. (2008). Defina a matriz $P_\Omega(Y_{m\times n})$

$$P_\Omega(Y)(i,j) = \begin{cases} Y_{ij}, & \text{se} \quad (i,j) \in \Omega, \\ 0, & \text{se} \quad (i,j) \notin \Omega, \end{cases} . \tag{2.4}$$

a qual é a projeção da matriz $Y_{m\times n}$ nas entradas observadas. Assim, com a projeção complementar sendo $P_\Omega^\perp$ via $P_\Omega^\perp + P_\Omega = Y$ e usando (2.4), pode-se reescrever $\sum_{(i,j)\in\Omega}(X_{ij} - Z_{ij})^2$ como $\|(P_\Omega(X) - P_\Omega(Z)\|_F^2$. O que nos leva ao problema de otimização em foco

$$\underset{Z}{\text{minimize}}\ f_\lambda(Z) := \frac{1}{2}\|P_\Omega(X) - P_\Omega(Z)\|_F^2 + \lambda\|Z\|_* \tag{2.5}$$

no qual o primeiro termo $\|P_\Omega(X) - P_\Omega(Z)\|_F^2$ é a norma de Frobenius[1] calculada apenas para a parte observada da matriz. A norma de Frobenius foi usada para medir a magnitude da fidelidade da aproximação, uma medida de erro empírico. Já o segundo termo $\lambda\|Z\|_*$ contém o parâmetro de regularização λ que está penalizando a norma nuclear.

Para calcular a norma nuclear é necessário encontrar os valores singulares de Z, os quais são obtidos por meio de uma decomposição em valores singulares.

[1] A norma de Frobenius de uma matriz A é definida como $\|A\|_F^2 = \sum_{i,j} |a_{ij}|^2$

2.2 Técnicas essenciais para a resolução do problema de otimização

Nessa seção são descritos conceitos importantes utilizados tanto para uma melhor compreensão do problema de otimização, quanto para sua resolução. Os assuntos se dividem em dois tópicos "Matrizes esparsas e formas de armazenamento" e "Decomposição em valores singulares".

2.2.1 Matrizes Esparsas e as principais formas de armazenamento

Se a maioria dos elementos de uma matriz são zeros, ela é considerada esparsa. Em contrapartida se a maioria dos seus elementos são diferentes de 0 ela é chamada densa.

A esparsidade de uma matriz pode ser medida pela divisão do número de elementos iguais a 0 pelo total de elementos. Seja uma matriz C da forma:

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \end{pmatrix}$$

Essa matriz tem 20 zeros e 25 elementos, logo ela tem esparsidade de 80%.

Matrizes esparsas quando eficientemente representadas, trazem economia no armazenamento e por consequência em manipulação. Dados esparsos são mais fáceis de serem comprimidos, uma vez que é possível através de estruturas especiais guardar, por exemplo, apenas os elementos diferentes de 0 e seus índices correspondentes, tornando operações em matrizes esparsas mais velozes e utilizando menos memória do que se fossem representadas com técnicas para matrizes densas.

Existem diversos formatos para armazenar matrizes esparsas, alguns são geralmente utilizados para construir matrizes, como o **COO**, e outros mais adequados para operações matriciais como o **CSR** e **CSC**. Esse métodos são descritos a seguir:

- **COO (Coordinate format)**:

  Representa a matriz por uma lista de coordenadas, da forma que cada elemento diferente de zero é representado pelos vetores (**i**=linha,**j**=coluna,**v**=valor). Para esse método pode ser usada qualquer ordem de armazenamento, seja por coluna ou linha.

  Pode-se armazenar a matriz C do seguinte modo:

  $\mathbf{i} = (2, 3, 3, 5, 5)$
  $\mathbf{j} = (2, 3, 5, 2, 5)$
  $\mathbf{v} = (3, 4, 1, 1, 2)$

Esse tipo de formato tem como vantagem a conversão rápida entre formatos esparsos, como a partir de e para **CSR** ou **CSC**.

- **CSR (Compressed Sparse Row)**:

  O formato **CSR** armazena a informação das linhas de forma comprimida. Para isso, também utiliza 3 vetores para representar uma matriz qualquer. Pode-se defini-los como :

  - **v**:
    São os valores diferentes de zero da matriz. Esses são dispostos no vetor, na ordem em que se encontram nas linhas da matriz. Usando C como exemplo

    $$\mathbf{v} = (3, 4, 1, 1, 2)$$

  - **j**:
    Guarda os índices das colunas de cada valor em **v**. Então para C:

    $$\mathbf{j} = (2, 3, 5, 2, 5)$$

  - **i**:
    Ao invés de guardar os índices das linhas para cada valor, nesse caso, observa-se a posição em **v** de cada elemento que começa uma linha na matriz. Caso não haja elementos em alguma linha, a posição do próximo número não-nulo deve ser repetido. Por convenção, o último elemento desse vetor é sempre igual o número de elementos diferentes de zero mais um. Esse último elemento é adicionado para identificar a primeira posição vazia nos vetores **v** e **j**. Para exemplificar o método, serão detalhados alguns passos para o vetor **i** da matriz C:
    Uma vez que na primeira linha não há nenhum valor diferente de zero, então escreve-se 1 em **i**, pois o próximo número que começa uma linha será o primeiro elemento. Passando para a segunda linha, o primeiro valor é 3, logo repete-se o número 1. Já na terceira linha, o primeiro que é distinto de zero é o 4, e ele tem posição 2. Seguindo dessa maneira o vetor completo será:

    $$\mathbf{i} = (1, 1, 2, 4, 4, 6)$$

    Esse tipo de formato tem como vantagem a eficiência na realização de operações aritméticas e obtenção de linhas, além de ser rápido para produtos entre matriz e vetor. Quanto a desvantagens é lento para operações de extração de colunas e mudanças na estrutura esparsa são dispendiosas.

- **CSC (Compressed Sparse Column)**:

  É semelhante ao anterior, mas dessa vez são as colunas que são guardadas de forma comprimida. Em relação ao anterior existem as seguintes diferenças:

  - **v**:
    Ainda é o vetor de valores não-nulos, mas agora segue a ordem das colunas da matriz;
  - **i**:
    Corresponde aos índices das linhas de cada elemento em **v**.
  - **j**: Agora se refere as posições de cada valor no vetor **v** que começam uma coluna na matriz. No caso em que não houver elementos na coluna repete-se o índice do próximo número distinto de 0.

    A matriz C pode ser escrita:
    $\mathbf{i} = (2, 5, 3, 3, 5)$
    $\mathbf{j} = (1, 1, 3, 4, 4, 6)$
    $\mathbf{v} = (3, 1, 4, 1, 2)$

    Esse tipo de formato também apresenta bom desempenho, tendo as mesmas vantagens que o último, com a diferença de que ao contrário do anterior esse método é eficiente na obtenção de colunas da matriz e lento para extrair linhas.

Em vista das vantagens vistas acima, para um banco de dados pouco observado, muitas vezes essa ausência de observações pode ser representada na forma de zeros, permitindo a utilização desses métodos de armazenamento para matrizes esparsas como é o caso dos dados da *Netflix*.

2.2.2 Decomposição em valores singulares (*SVD*)

Na seção 2.1 foi citada uma ferramenta chamada decomposição em valores singulares, devido a grande importância dessa técnica, vamos defini-la formalmente.

Decomposição em valores singulares é a fatoração de uma matriz X qualquer, no produto de três matrizes, isto é, $X = U\Sigma V^T$. Para X com m linhas e n colunas, a decomposição é o produto entre:

- $U_{m\times m}$ uma matriz ortogonal, cujas colunas são vetores singulares esquerdos de X, $\vec{u}$. Estes são um conjunto de autovetores ortonormais de XX^T;
- $\Sigma_{m\times n}$ uma matriz diagonal, cuja diagonal principal é composta pelos valores singulares σ_i de X. Esses valores são as raízes dos autovalores de XX^T e X^TX;

- $V^T_{n\times n}$ uma matriz ortogonal, cujas linhas são vetores singulares direitos de X, $\vec{v}$. Estes são um conjunto de autovetores ortonormais de X^TX;

Essa representação acima corresponde a versão "cheia" da *SVD*, isto é, a definição formal. Na prática utiliza-se normalmente as versões reduzidas, que são mais rápidas e economizam armazenamento. Abaixo algumas opções de *SVD* reduzido e sua representação na forma de somatório, supondo $X_{m\times n}$ uma matriz com posto r.

SVD fina (Thin *SVD*)

$X = U_n\Sigma_n V_n^T$ é a *SVD* fina de X, no qual as dimensões das matrizes decompostas(fatoradas) são $U_{m\times n}$, $\Sigma_{n\times n}$ e $V^T_{n\times n}$. Essa forma de *SVD* é mais rápida do que a *SVD* Completa se $n << m$. Escrevendo na forma de um somatório:

$$X = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

.

SVD Compacta (Compact *SVD*)

$X = U_r\Sigma_r V_r^T$ é a *SVD* compacta de X, no qual as dimensões das matrizes decompostas(fatoradas) são $U_{m\times r}, \Sigma_{r\times r} e V^T_{r\times n}$. Essa forma de *SVD* é mais rápida do que a *SVD* Fina se $r << n$. Na forma de um somatório:

$$X = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

SVD Truncada (Truncated *SVD*)

$X = U_t\Sigma_t V_t^T$ é a *SVD* truncada de X, no qual as dimensões das matrizes fatoradas são $U_{m\times t}$, $\Sigma_{t\times t}$ e $V^T_{t\times n}$. Essa forma de *SVD* é mais rápida do que a *SVD* Compacta se $t << r$. Na forma de um somatório:

$$X = \sum_{i=1}^{t} \sigma_i u_i v_i^T$$

É importante observar que a *SVD* Truncada não é mais uma reprodução exata da matriz X.

2.3 O problema de Otimização e sua solução pelo algoritmo *Soft-Impute*

2.3.1 Regularização pela Norma Nuclear

A partir da descrição do que é uma *SVD*, pode-se definir a solução para o problema convexo de regularização pela norma nuclear.

Seja uma matriz qualquer W, com posto r, temos o problema de otimização

$$\underset{Z}{\text{minimize}}\ \frac{1}{2}\|W - Z\|_F^2 + \lambda\|Z\|_* \tag{2.6}$$

a solução é dada por $\hat{Z} = \mathbf{S}_\lambda(W)$ na qual

$$\mathbf{S}_\lambda(W) \equiv UD_\lambda V^T \quad \text{com} \quad D_\lambda = \text{diag}[(d_1 - \lambda)_+, ..., (d_r - \lambda)_+], \tag{2.7}$$

UDV^T é a decomposição em valores singulares de W, $D = \text{diag}[d_1, ..., d_r]$, na qual $d_i \in (1, ..., r)$ são os r valores singulares de W e $t_+ = \max(t, 0)$. A notação $S_\lambda(W)$ se refere a decomposição em valores singulares de limiar-macio (soft-thresholded SVD) (Donoho et al., 1995). Essa representação aparece em Cai et al. (2008) e Ma et al. (2011).

2.3.2 Algoritmo *Soft-Impute* (Imputação Suave)

Para resolver o problema de otimização restrito em Ω:

$$\underset{Z}{\text{minimize}}\ f_\lambda(Z) := \frac{1}{2}\|P_\Omega(X) - P_\Omega(Z)\|_F^2 + \lambda\|Z\|_* \tag{2.8}$$

o algoritmo utilizado nesse trabalho foi o *Soft-Impute* (Mazumder et al.), cujos passos se encontram abaixo:

1. Inicializa-se $Z^{velho} = 0$
2. para $\lambda_1 > \lambda_2 > ... > \lambda_k$:
   a) Repita:
      i. Calcule Z^{novo} através de $\mathbf{S}_{\lambda_k}(P_\Omega(X) + P_\Omega^\perp(Z^{velho}))$.
      ii. Se $\frac{\|Z^{novo} - Z^{velho}\|_F^2}{\|Z^{velho}\|_F^2} < \epsilon$ terminar.
      iii. Defina Z^{velho} como Z^{novo}
   b) Defina $\hat{Z}_{\lambda_k}$ como Z^{novo}
3. As saídas serão as soluções $\hat{Z}_{\lambda_1}, ..., \hat{Z}_{\lambda_k}$

O primeiro passo começa com a definição da matriz de aproximação $Z^{velha} = 0$. Depois a partir de diferentes valores de λ (em ordem decrescente) o algoritmo repetidamente substitui as observações faltantes com o palpite atual, e então atualiza o palpite resolvendo a *soft-thresholded SVD*. A necessidade do cálculo dessa SVD para $(P_\Omega(X) + P_\Omega^\perp(Z^{velho}))$, a cada iteração, é a parte mais trabalhosa do algoritmo computacionalmente. Para facilitar o cálculo é possível reescrever $P_\Omega(X) + P_\Omega^\perp(Z^{velho})$ como:

$P_\Omega(X) + P_\Omega^\perp(Z^{velho}) = [P_\Omega(X) - P_\Omega(Z^{velho})] + Z^{velho}$

Com essa mudança tem-se que o primeiro termo da equação é esparso e o segundo de posto baixo. Com relação a parte esparsa pode-se dizer que tem baixo custo de armazenamento e cálculo. A parte de posto baixo também tem custo baixo para armazenar.

2.4 Pós processamento da solução *SVD*

Ao completar a matriz, o parâmetro de regularização penaliza o tamanho dos valores singulares, através do encolhimento (*shrinkage*) dos mesmos. Isso incorre no acréscimo de viés, subestimando o modelo, possivelmente representando de forma falha a importância de alguns parâmetros.

Uma vez obtida a solução do *Soft-Impute*, pode se fazer um pós-processamento, a partir da expansão dos auto-valores da solução Z_λ com o propósito de retirar esse viés. Isso pode ser obtido pelo seguinte problema de otimização:

$$\begin{aligned} \alpha &= \underset{\alpha_i \geq 0, i=1,...,r_\lambda}{\arg\min} \|P_\Omega(X) - \sum_{i=1}^{r_\lambda} \alpha_i P_\Omega(u_i v_i^T)\|_F^2 \\ Z_\lambda^u &= U D_\alpha V^T \end{aligned} \tag{2.9}$$

no qual a matriz $D_\alpha = \text{diag}(\alpha_1, ..., \alpha_{r_\lambda})$ e r_λ é posto da solução Z_λ, com $Z_\lambda = U D_\lambda V^T$. Os valores singulares α_i são estimados na forma de uma regressão linear, por mínimos quadrados ordinários. Essa abordagem é viável por conta da esparsidade de $P_\Omega(u_i v_i^T)$ e dado que r_λ é pequeno. Em outras palavras, estimar os parâmetros da regressão não será tão dispendioso, pois Ω considera apenas os índices observados e o número de α_i's é limitado indiretamente por λ. Caso alguma solução α_i tenha sinal negativo, o sinal é absorvido pelo vetor singular correspondente.

2.5 O problema da *Netflix*

Antes de prosseguir para os resultados, vamos usar um exemplo fictício do livro *Mining of Massive Datasets*, com o intuito de demonstrar uma possível interpretação da decomposição em valores singulares para os dados da *Netflix* e mostrar de maneira simples uma forma de previsão de avaliações faltantes.

Para fins de esclarecimento, o exemplo a ser utilizado não é representativo da matriz real da *Netflix*, apenas uma maneira de descrever o nosso problema.

2.5.1 Interpretação do *SVD* e recontrução da matriz original

No exemplo em questão seja $X_{7\times5}$, a matriz de usuários por filmes de posto 3, cujas observações são os votos de cada indivíduo para determinados filmes variando de 1 a 5. Os zeros representam dados faltantes (NA's), isto é, uma casela com valor 0 significa que o usuário daquela linha não votou no filme da respectiva coluna. A seguir o exemplo

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	2	0	4	4
Jenny	0	0	0	5	5
Jane	0	1	0	2	2

Figura 1 – Matriz de Usuários × Filmes. Livro *Mining of Massive Datasets.*

Observando a figura 1, do livro *Mining of Massive Datasets*, percebe-se que existem pessoas com gostos similares. Por exemplo, os homens só avaliaram filmes de ficção científica (*Matrix*, *Alien* e *Star Wars*), já as mulheres avaliaram principalmente filmes de romance (*Casablanca* e *Titanic*), apesar de duas terem avaliado também o filme *Alien*. Logo os homens parecem gostar apenas de ficção enquanto as mulheres preferem romance, visto que deram notas baixas ao filme *Alien*.

Ao aplicar um *SVD Compacto* em X, as matrizes $U\Sigma V^T$ terão as seguintes interpretações:

- U: associa usuários, em termos de conceitos.

- V^T: relaciona filmes, em termos de conceitos.

- Σ: os elementos da diagonal trazem a intensidade de cada conceito.

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{pmatrix}}_{X} =$$

$$\underbrace{\begin{pmatrix} -.14 & -.02 & -.01 \\ -.41 & -.07 & -.03 \\ -.55 & -.09 & -.04 \\ -.69 & -.12 & -.05 \\ -.15 & .59 & .65 \\ -.07 & .73 & -.68 \\ -.08 & .30 & .33 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} 12.48 & 0 & 0 \\ 0 & 9.51 & 0 \\ 0 & 0 & 1.35 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} -.56 & -.59 & -.56 & -.09 & -.09 \\ -.13 & .03 & -.13 & .70 & .70 \\ -.41 & .80 & -.41 & -.09 & -.09 \end{pmatrix}}_{V^T}$$

Agora para observar o comportamento dessas matrizes, foi escolhido o usuário Joe. As entradas da primeira linha de U correspondem ao gosto de Joe. A parcela de maior grandeza, com relação a linha, foi $-.14$. Isso indica que apesar de Joe só ter interesse por filmes de ficção, ele os avalia como ruins. O segundo valor indica que Joe não gosta de filmes de romance. As entradas da primeira linha da matriz V^T indicam que os primeiros três filmes são do gênero de ficção científica, e as entradas de valor $-.09$ denotam que esses filmes não tem praticamente nada em comum com romance. A segunda linha nos diz que (*Casablanca* e *Titanic*) são estritamente romances, e tem pouca similaridade com o outro gênero. A terceira coluna de U e terceira linha de V^T apresentam um terceiro conceito difícil de explicar. Porém a matriz Σ mostra que a força desse conceito é pequeno, por conta da magnitude do valor singular. Isso nos leva ao nosso próximo passo que é utilizar um *SVD Truncado* para reduzir a dimensão de X, eliminando esse terceiro conceito.

Fazendo a redução de dimensão além de excluir informações sem importância, isso também resolve o problema do armazenamento. Uma vez que se estivesse sendo usado o banco de dados real da *Netflix*, ao invés do exemplo acima, armazenar a matriz X na forma $U\Sigma V^T$ não seria muito conveniente.

No caso da matriz X que tem posto 3, para reduzir a dimensão para 2 (número de conceitos), selecionou-se o menor dos três valores singulares e o reduziu a 0. Como transformou-se um valor singular em 0, ao multiplicar a decomposição, a terceira coluna de U e a terceira linha de V^T serão multiplicadas por 0, então elas podem ser simplesmente

retiradas. Abaixo encontrou-se a aproximação $\tilde{X}$ com as matrizes reduzidas.

$$\begin{pmatrix} -.14 & -.02 \\ -.41 & -.07 \\ -.55 & -.09 \\ -.69 & -.12 \\ -.15 & .59 \\ -.07 & .73 \\ -.08 & .30 \end{pmatrix} \begin{pmatrix} 12.48 & 0 \\ 0 & 9.51 \end{pmatrix} \begin{pmatrix} -.56 & -.59 & -.56 & -.09 & -.09 \\ -.13 & .03 & -.13 & .70 & .70 \end{pmatrix}$$

$$= \begin{pmatrix} 1.00 & 1.03 & 1.00 & 0.02 & 0.02 \\ 2.95 & 3.00 & 2.95 & -0.01 & -0.01 \\ 3.96 & 4.02 & 3.96 & 0.02 & 0.02 \\ 4.97 & 5.05 & 4.97 & -0.02 & -0.02 \\ 0.32 & 1.27 & 0.32 & 4.10 & 4.10 \\ 0.41 & 0.72 & -0.41 & 4.94 & 4.94 \\ 0.19 & 0.67 & 0.19 & 2.09 & 2.09 \end{pmatrix}$$

A decomposição foi obtida utilizando 4 casas decimais e foi realizado um arredondamento dos valores para 2 casas para uma melhor representação. Percebe-se que o resultado está próximo da matriz original, na qual a principal fonte de erro foi a redução do posto. Resumindo, pode-se concluir que é uma boa aproximação.

2.5.2 Previsão para novos usuários

Com a *SVD* calculada anteriormente também é possível fazer previsão para uma pessoa que não estava representada na matriz original de usuários e filmes (figura 1), contanto que exista pelo menos uma avaliação disponível para esse usuário com respeito a um dos filmes representados.

Suponha então que Marcos deu nota 4 para *The Matrix*, então ele pode ser representado pelo vetor $m = [4, 0, 0, 0, 0]$, que pode ser visto como uma das linhas da figura 1. Com esse vetor é possível mapear Marcos no "espaço conceitual" multiplicando m por V e em seguida por V^T para projeta-lo no espaço de filmes:

$$mVV^T = \begin{pmatrix} 4.00 & .00 & .00 & .00 & .00 \end{pmatrix} \begin{pmatrix} -.56 & -.13 \\ -.59 & .03 \\ -.56 & -.13 \\ -.09 & .70 \\ -.09 & .70 \end{pmatrix} \begin{pmatrix} -.56 & -.59 & -.56 & -.09 & -.09 \\ -.13 & .03 & -.13 & .70 & .70 \end{pmatrix}$$

$$= \begin{pmatrix} 1.32 & 1.31 & 1.32 & -.16 & -.16 \end{pmatrix}$$

Logo obtém-se o vetor com as notas previstas para Marcos, sugerindo sua preferência pelos filmes de ficção e desinteresse em filmes de romance.

Pode-se ver que tanto a reconstrução da primeira nota, que já era conhecida, ficou longe do real, quanto a previsão de novos valores não teve o resultado esperado. Isso ocorreu porque só era conhecida apenas uma avaliação de Marcos, logo o modelo precisava de mais informação para estimativas mais precisas.

Se for feita a previsão para o mesmo usuário, porém com uma nova avaliação, o resultado obtido parece ir de acordo com os conceitos:

Seja $\mathbf{m} = (4, 1, 0, 0, 0)$
$\mathbf{m}VV^T = (1.65, 1.66, 1.65, -.09, -.09)$

Com o acréscimo de mais uma nota, por mais que de valor baixo, todas as estimativas aumentaram. Como o voto foi para outro filme de ficção científica, isso reforça o gosto desse usuário por filmes desse gênero, resultando no aumento das 3 primeiras notas. Quanto aos filmes sobre romance, o fato desse novo voto ter sido baixo, explica o aumento das estimativas desse gênero, já que eles são conceitos opostos.

Para novos usuários $\mathbf{k}$ e $\mathbf{l}$ as estimativas são:
$\mathbf{k} = (0, 1, 4, 3, 0)$
$\mathbf{k}VV^T = (1.53, 1.88, 1.53, 1.41, 1.41)$

$\mathbf{l} = (0, 1, 3, 0, 5)$
$\mathbf{l}VV^T = (1.12, 1.70, 1.12, 2.44, 2.44)$

É visível que o primeiro está dividido entre os dois conceitos e o segundo prefere romance, mas também gosta de ficção. Porém nesses novos usuários percebe-se um padrão (viés) na aproximação da segunda nota. Nos dois casos, apesar de possuir a avaliação mais baixa possível, a estimativa para esse filme é sempre superior às outras do mesmo gênero. Logo que verifica-se que esse método de previsão é adequado apenas para prever novos valores.

3 Resultados

Para a realização dos experimentos nesse trabalho foram utilizados computadores com as seguintes especificações:

- `Windows 7, processador Intel(R) Core(TM) i5-4570 CPU @` 3.20 `GHz` 3.20 `GHz e` 8.00 `GB de memória RAM`.
- `Windows 10, processador Intel(R) Core(TM) i7-6500U CPU @` 2.50 `GHz` 2.60 `GHz e` 8.00 `GB de memória RAM`.

Além de máquinas físicas, foi utilizada uma conta no servidor do departamento de estatística com as especificações abaixo:

- `Linux, processador Intel(R) Xeon(R) CPU @` 2.40 `GHz e` 10.00 `GB de memória RAM`.

Esse servidor foi acessado por meio do *software* de emulação de terminal *PUTTY*, que permite acesso remoto a um servidor de forma segura. Para toda a parte de programação desse estudo, como analise exploratória do banco de dados e a previsão das avaliações faltantes, foi utilizado o *software* livre `R` versão 3.3.1.

O código da programação deste trabalho, desenvolvido no software livre `R`, está disponível no apêndice.

3.1 Banco de dados da *Netflix*

Para resolver o problema de otimização explicitado na seção 2.1 foram usados os dados fornecidos para o Prêmio *Netflix*. A seguir tem-se a descrição do banco de dados da forma como foi disponibilizado para a competição.

Descrição

Os dados foram coletados entre Outubro de 1998 e Dezembro de 2005 e refletem a distribuição de todas as avaliações recebidas pela *Netflix* nesse período. O arquivo fornecido contém os seguintes arquivos:

- **Dados de Treinamento (Training Dataset)**:

  A base de dados de treinamento contém 17770 arquivos, um para cada filme. Cada arquivo possui as variáveis abaixo:

  - Identificação de Filme:
    Um número inteiro escolhido de forma arbitrária, variando sequencialmente de 1 a 17770, para representar cada filme.
  - Identificação de Usuário:
    Um número inteiro escolhido de forma arbitrária, variando de 1 a 2649429 (com alguns faltantes), para representar cada usuário. Totalizando 480189 usuários.
  - Avaliação:
    Pontuação dada por usuários a determinados filmes. No total 100480507 avaliações variando de 1 a 5 (inteiros).
  - Data:
    data da avaliação, disponibilizada na forma ano-mês-dia, no intervalo 1998-11-01 até 2005-12-31.

    Os dados possuem o seguinte formato em cada arquivo:

(Identificação de Filme)	1:
(Identidade de Usuário, Avaliação, Data)	$1488844, 3, 2005-09-06$
(Identidade do Usuário, Avaliação, Data)	$822109, 5, 2005-05-13$
...	...

- **Títulos de Filme (Movie Titles)**:

  Esse documento contêm a variável identificação de filme, e outras 2:

  - Ano de Lançamento: Ano em que o filme foi lançado dentro do intervalo $[1890, ..., 2005]$. O ano pode ser tanto o ano de lançamento do filme nos cinemas, quanto ano de lançamento do dvd.
  - Título do filme: Título do filme em inglês como no site da *Netflix*.

    Os dados estão dispostos da seguinte forma no arquivo:

(Identificação de filme, Ano, "Título")	$10120, 1982$,"Blade Runner"
...	...
(Identificação de filme, Ano, "Título")	$17690, 2007$,"The Queen"
...	...

- **Dados de Qualificação (Qualifying Dataset)**:

Esse documento contém as variáveis identificação de filme, identificação de usuário e data.

Exemplo da organização dos dados:

(Identificação de Filme)	1:
(Identidade de Usuário, Data)	$1046323, 2005-12-19$
(Identidade do Usuário, Data)	$1080030, 2005-12-23$
...	...

- **Dados de Sondagem (Probe Dataset)**:

  Os dados presentes nesse arquivo são sobre as variáveis identificação de filme e identificação de usuário. As avaliações e datas para cada observação dessas duas variáveis estão também presentes nos dados de treinamento.

  Exemplo da organização dos dados:

(Identificação de Filme)	10:
(Identidade de Usuário)	1952305
(Identidade do Usuário)	1531863
...	...

O nosso objetivo para esse banco de dados foi prever as avaliações faltantes na matriz de usuários e filmes da *Netflix*. Para isso foram utilizados os arquivos **Dados de Treinamento** e **Dados de Sondagem**, contidos no arquivo descrito acima.

3.2 Importação e Preparação dos dados da *Netflix*

A parte de importação e preparação dos dados para análise apresentou algumas dificuldades resultantes da magnitude dos dados e da forma como foram organizados no arquivo. Por conta disso foram utilizados os pacotes de *Big Data* "dplyr" e "data.table" para a importação e manipulação de ambos os conjuntos.

3.2.1 Dados de Treinamento

Os dados de treinamento foram armazenados em 17770 documentos de texto, cuja estrutura interna foi descrita na seção 3.1. Essa maneira de organização dos dados trouxe dificuldades, tanto para importação, quanto para a composição de um único conjunto para análise. A fim de lidar com isso, eles foram importados na forma de uma lista e então reescritos em um único arquivo de texto, que por sua vez foi importado no formato

`data.table`. A classe `data.table` herda (*inherits*[1]) da `data.frame`, isto é, a primeira é uma extensão da segunda. A vantagem nesse formato é que ele permite manipulações mais rápidas do que ao lidar com o seu antecessor. Além disso, é compatível com funções e pacotes do R que foram criadas para `data.frame`.

Depois da importação foi necessária uma modificação nos identificadores dos usuários, que foram renomeados de modo que formassem uma sequência de 1 até o número de usuários ao invés de 1 a 2649429 (com alguns faltantes) como era inicialmente. A tabela 1 mostra as 5 primeiras e 5 últimas linhas do conjunto de treinamento ordenado pelos usuários:

Tabela 1 – Primeiras e últimas 5 linhas dos dados de treinamento.

	filmes	Usuários	notas
1	30	1	3
2	157	1	3
3	173	1	4
4	175	1	5
5	191	1	2
6	17560	480189	5
7	17580	480189	4
8	17622	480189	4
9	17627	480189	3
10	17692	480189	2

O motivo dessa mudança será explicado na seção 3.4.1.

3.2.2 Dados de Sondagem

Os dados de sondagem continham dois separadores em seu arquivo, um separando os filmes (:) e outro (,) os usuários. A forma encontrada para lidar com isso foi importar tudo como um vetor de caracteres. Uma vez importado, o vetor foi manipulado de modo a obter uma forma similar ao arquivo anterior, a diferença é que nesse caso o banco só possui informações sobre usuários e filmes. Assim como no conjunto de treinamento aqui também os usuários foram renomeados. A tabela 2 mostra as 5 primeiras e 5 últimas linhas do conjunto de sondagem ordenado pelos usuários:

[1] *Inheritance é um dos aspectos mais úteis em linguagens de programação orientadas a objetos como o* `R`*. Ela torna possível reaproveitar classes existentes para criar novas, evitando construções a partir do* 0*. Ao criar uma nova classe ela herda os aspectos da classe usada como base e adquiri novas características.*

Tabela 2 – Primeiras e últimas 5 linhas dos dados de sondagem.

	filmes	Usuários
1	501	1
2	658	1
3	825	1
4	285	2
5	6408	2
6	6289	480188
7	7281	480188
8	10358	480188
9	2186	480189
10	3333	480189

3.3 Análise Exploratória

Com a finalidade de compreender um pouco sobre os dados e as relações entre as variáveis foi feita uma análise exploratória. As avaliações escolhidas pelos usuários são observações de uma variável aleatória discreta, por conta disso a distribuição de suas frequências foi denotada por uma gráfico de barras. As tabela e gráficos abaixo mostram a dimensão e o comportamento da frequência de cada nota.

Tabela 3 – Frequência para as avaliações do conjunto de treinamento.

Avaliações	Frequência
1	4617990
2	10132080
3	28811247
4	33750958
5	23168232

Figura 2 – Gráfico de barras para as notas dos dados de treinamento.

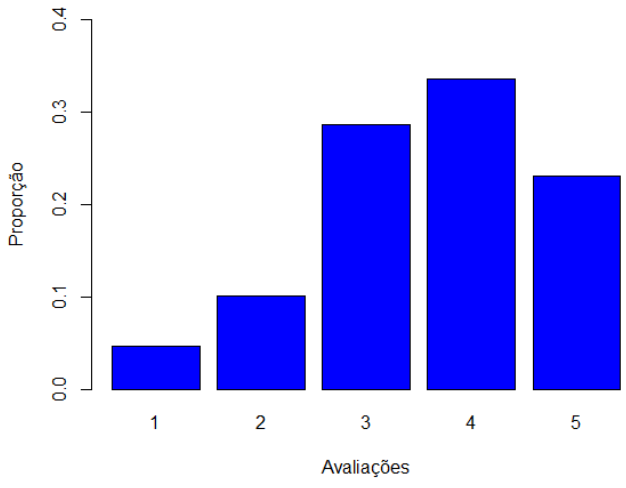
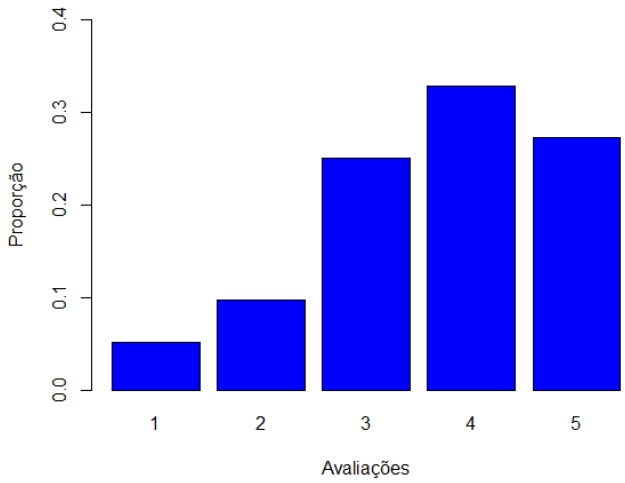


Figura 3 – Gráfico de barras para as notas dos dados de sondagem.



Comparando percebe-se que as distribuições das avaliações são bem similares, a diferença é que para o os dados de sondagem a proporção de notas altas é maior, especialmente de notas 5.

Sobre os dados de treinamento, é perceptível pela tabela 3 e figuras 2 e 3 que a quantidade de notas cresce conforme a nota aumenta, esse comportamento só muda da nota 4 para a 5 no qual há uma ligeira queda no número de usuários. A tabela 4 apresenta medidas descritivas para o banco de dados de treinamento :

Tabela 4 – Medidas descritivas para as avaliações dos dados de treinamento..

Medidas	
Média	3.6042900
Moda	4.0000000
Mediana	4.0000000
Variância	1.1776993
Desvio Padrão	1.085219
Assimetria	-0.5036536
Curtose	-0.3297949
Mínimo	1.0000000
Máximo	5.0000000

As medidas de tendência central (média, mediana e moda) indicam que os usuários tendem a dar notas altas para os filmes, em particular a nota mais escolhida foi a nota 4. Isso pode indicar que na maior parte do tempo os clientes da *Netflix* fazem escolhas adequadas sobre qual filme assistir ou que eles têm preferência por avaliar apenas filmes que os agradam. Observando as medidas de dispersão, como desvio e variância, pode-se dizer que existe pouca variação entre as notas. Pelo gráfico 2 nota-se que as frequências das avaliações $(1, 2)$ e $(3, 4, 5)$ são bem distintas, sendo que tanto as notas 1, quanto 2 correspondem a menos da metade de qualquer uma das outras. As medidas de assimetria

e curtose reforçam essa diferença, pois indicam assimetria a esquerda, implicando que a distribuição das notas se comporta de forma diferente para notas mais baixas do que para as mais altas.

Também foi observado o comportamento das avaliações quando agrupadas por filmes e usuários.

Figura 4 – Histograma para as médias dos filmes.

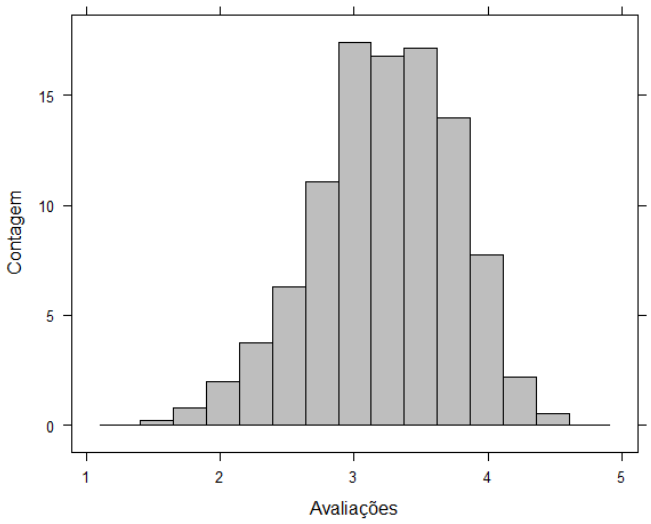
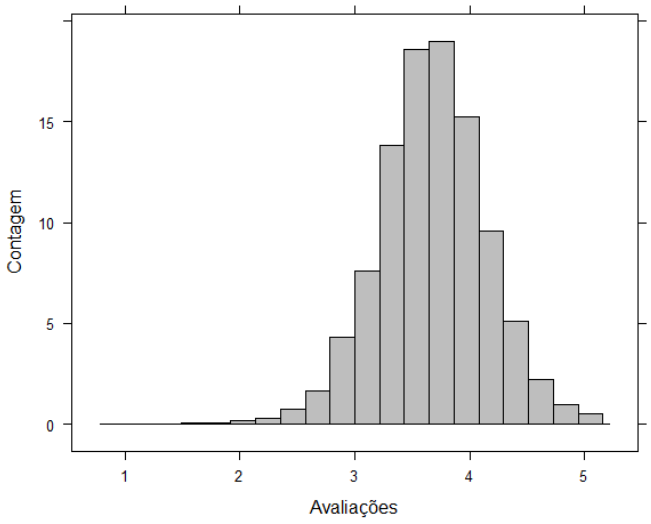


Figura 5 – Histogramas para as médias dos usuários.



Pela a figura 4, percebe-se que a maioria dos filmes teve sua nota média entre 2.5 e 4, isso pode indicar que a grande maioria dos filmes escolhidos para ser parte do catálogo da *Netflix* são bons filmes. A figura 5 mostra como as avaliações médias dos usuários são altas, reforçando a ideia de que os usuários em geral gostam da grande maioria dos filmes que assistem ou só votam no que gostam.

Em média cada um dos filmes recebeu 5655 avaliações, o que parece um pouco surreal, já

que é difícil imaginar que cada um dos 17770 filmes tenha recebido mais de 5000 avaliações. Calculando a mediana, que foi 561, percebe-se que a média está sendo influenciada por *outliers*, já que a mediana indica que metade dos filmes não receberam nem 600 avaliações. As tabelas 5 e 6 mostram, respectivamente, os filmes com maior e menor número de avaliações.

Tabela 5 – Os 5 filmes mais avaliados

Filme	Ano	Número de Avaliações	Nota média
Miss Congeniality	2000	232944	4.1539
Independence Day	1996	216596	3.4422
The Patriot	2000	200832	3.7839
The Day After Tomorrow	2004	196397	3.7242
Pirates of the Caribbean	2003	193941	3.3613

Tabela 6 – Os 5 filmes menos avaliados.

Filme	Ano	Número de Avaliações	Nota média
Mobsters and Mormons	2005	3	4
The Land Before Time IV	1996	5	4.2
Larryboy and the Rumor Weed	1999	10	2.4
Hockey Mom	2004	10	2.4
Dune: Extended Edition	1984	13	3.7692

É interessante notar que todos os 5 filmes mais avaliados podem ser caracterizados como do sub-gênero de ação, indicando a preferência do público pela categoria. Além disso, mesmo com a grande quantidade de votos, em especial *Miss Congeniality* tendo recebido avaliações de quase metade dos usuários, esses filmes mantiveram suas notas médias acima de 3, indicando seu sucesso dentre os usuários da *Netflix*. Quanto aos 5 menos avaliados, somando todos os seus votos, o número não chega nem a 0.02% das notas de *Miss Congeniality*. Observando os gêneros, não há padrão definido, os filmes se dividem entre animações, comédias e ficção científica.

Estudando as avaliações com relação aos usuários, verificou-se que em média cada um avaliou 209 filmes. Já a mediana, mínimo e máximo de avaliações foram respectivamente 96, 1 e 17653. Ou seja, cerca de metade das pessoas avaliaram menos de 97 filmes, sendo que 1269 só votaram em 1 filme e apenas 5 pessoas avaliaram mais de 10000.

3.4 Aplicação do algoritmo e seleção de modelos

O algoritmo que foi utilizado nesse trabalho, já foi implementado no software `R` dentro do pacote "softImpute". O pacote foi desenvolvido por Trevor Hastie e Rahul Mazumder e contém diversas funções que ajudaram na análise.

3.4.1 Representação na forma de Matriz Esparsa

Como o objetivo foi empregar modelos para a previsão de avaliações faltantes, foi necessário representar essa ausência de notas de alguma forma. Devido a enorme dimensão do banco de dados a representação foi feita no formato de uma matriz esparsa.

A mudança nos identificadores dos usuários, no banco de treinamento, foi feita para que a matriz fosse representada apenas pelos usuários que realizaram pelo menos uma avaliação, ou seja, de forma que não houvesse nenhum usuário sem votar. Em consequência foi necessário ajustar os ID's do banco de sondagem, para seu correspondente no de treinamento.

A fim de colocar os dados na forma de uma matriz esparsa foi feito uso da função `Incomplete()`, cujos argumentos são índices correspondendo às coordenadas de cada observação. Essa função herda da classe `dgCMatrix` que é uma classe de matrizes esparsas no formato *CSC*. Logo ela armazenou os bancos de dados de treinamento e sondagem nesse formato e teve como saída uma matriz esparsa. Quando na forma esparsa, foi mais simples observar a proporção de observações faltantes no conjunto de treinamento, isto é, sua esparsidade que foi de aproximadamente 98.82%.

3.4.2 As funções `lambda0()` e `softImpute()`

Após a criação da matriz, o próximo passo foi decidir a sequência dos parâmetros λ de regularização da norma nuclear e aplicar o algoritmo *Soft-Impute*. Existem funções no pacote "softImpute" que ajudaram em ambos os problemas.

Para o primeiro, a função `lambda0()` encontrou o maior valor singular para a matriz com zeros, representando os valores faltantes por meio do cálculo de um *SVD* de limiar macio e posto baixo através de regressão *ridge* ortogonal alternada.[2] Como no algoritmo *Soft-Impute* λ foi subtraído dos valores singulares, a saída dessa função é o menor valor para λ cuja o algoritmo retorna solução zero. Resumindo, com essa função foi decidido o chute inicial.

A aplicação do algoritmo foi simples, pois o pacote possui a função `softImpute()` que, com o argumento `type="svd"`, corresponde ao algoritmo descrito na seção 2.3.2. Logo bastou especificar os argumentos restantes da seguinte forma:

- `softImpute()`
  - `x`:
    A matriz observada com os valores faltantes na classe "Incomplete".

[2] Regressão *ridge* é uma variação da regressão linear comum, que regulariza as estimativas de seus coeficientes, ou seja, encolhe eles levando-os para perto do zero, introduzindo um pouco de viés para controlar a variância no modelo.

- `rank.max`:
  Um valor que restringe o posto da solução, de modo que se a solução tiver posto = `rank.max` não há garantia da resolução do problema de norma nuclear convexo de completar matrizes.
- `lambda`:
  Um valor para o parâmetro de regularização da norma nuclear.
- `thresh`:
  O limiar de convergência do algoritmo. Ele é medido como a mudança relativa na norma de Frobenius entre duas estimativas sucessivas.
- `maxit`:
  O número máximo de iterações do algoritmo.
- `trace.it`:
  Se for igual a `TRUE`, a função reporta o progresso de convergência.
- `warm.start`:
  É uma solução *SVD*, mais especificamente uma solução do algoritmo para ser usado como valor inicial. É usado para acelerar a convergência do algoritmo.

A saída da função corresponde estimativa $\hat{Z}_\lambda$ na forma de uma *SVD*.

3.4.3 Implementação do algoritmo para sequência de λ's

Calculando `lambda0()` para a matriz de treinamento obteve-se:

$$\lambda_0 = 18648.273$$

A partir desse valor foi criada uma sequência arbitrária de $\log(\lambda_0)$ até $\log(1)$ de comprimento 10 e depois aplicada a função exponencial, resultando nos valores abaixo:

$$(18648.273, 6253.501, 2097.045, 703.222, 235.818, 79.079, 26.518, 8.893, 2.982, 1)$$

Esses valores, com exceção do primeiro que é λ_0, e do último que provavelmente resultaria na reprodução quase exata da matriz (posto completo), formam a sequência S_1:

$$S_1 = \{18648.273, 6253.501, 2097.045, 703.222, 235.818, 79.079, 26.518, 8.893, 2.982, 1\}$$

cujos λ's foram utilizados para gerar os primeiros modelos $\hat{Z}_\lambda$.

Os primeiros 8 modelos tiveram os seguintes argumentos: `maxit` = 100, `rank.max` = 40, `thresh` = 10^{-5}, `trace.it` = TRUE, `warm.start` = NULL e `lambda` varia ao longo da sequência para cada modelo. O argumento `warm.start` foi definido como nulo, pois como esses modelos foram executados simultaneamente em diferentes computadores não havia solução prévia para ser usada como "aquecimento".

A tabela 7 mostra os modelos e algumas características.

Tabela 7 – Modelos $\hat{Z}_\lambda$ para os valores de S_1

modelo	λ	t (hrs)	n	posto	posto máximo
$\hat{Z}_{6253}$	6253.5010	1.289	25	1	40
$\hat{Z}_{2097}$	2097.0453	2.224	47	2	40
$\hat{Z}_{703}$	703.2220	3.370	86	8	40
$\hat{Z}_{235}$	235.8180	5.552	100^{**}	40^{*}	40
$\hat{Z}_{79}$	79.0791	4.675	100^{**}	40^{*}	40
$\hat{Z}_{26}$	26.5183	3.969	97	40^{*}	40
$\hat{Z}_{8}$	8.8926	3.750	96	40^{*}	40
$\hat{Z}_{2}$	2.9821	4.738	97	40^{*}	40

* Representa que o posto da decomposição foi limitado pelo argumento `rank.max`.

** Indica que o algoritmo terminou sem a função objetiva alcançar o limiar de convergência.

Conforme os valores de λ decresceram o posto da solução aumentou até o limite pré-definido 40. O tempo de execução do algoritmo e o número de iterações até convergência também aumentaram bastante ao longo de λ até $\hat{Z}_{235}$ que foi o primeiro limitado pelo algoritmo. Ou seja, até o modelo com $\lambda = 703.2220$ o problema de regularização aparentava ter sido resolvido. A partir dele todos tiveram seus postos truncados, o que implicaria que a solução foi um mínimo local, com exceção de $\hat{Z}_{235}$ e $\hat{Z}_{79}$, que terminaram por causa do número máximo de iterações, antes que a função objetiva alcançasse o limiar de convergência.

Depois de $\hat{Z}_{235}$ o tempo não se comportou da mesma maneira, primeiro diminuiu depois voltou a aumentar.

Devido ao fato de que usando o argumento `rank.max` $= 40$ e `maxit` $= 100$ não foi suficiente para que todos os ajustes convergissem para um mínimo global e o maior posto "verdadeiro" foi apenas 8, foram geradas novas sequências de ajustes com novos valores para λ.

A segunda sequência foi gerada de maneira similar a anterior, porém os valores estavam entre os λ_3 e λ_4 já que foi entre esses ajustes que os limites interferiram.

Então foi criada a sequência de log(703) até log(236) de comprimento 10, em seguida aplicou-se a função exponencial e retirou-se o primeiro e último valores de λ que são equivalentes aos ajustes $\hat{Z}_{703}$ e $\hat{Z}_{235}$, temos os valores abaixo:

$$S_2 = \{622.707, 551.585, 488.586, 432.782, 383.352, 339.568, 300.784, 266.430\}$$

Com os valores de S_2, e mantendo o argumento `rank.max` $= 40$ foi modificado apenas o máximo de iterações para 200, e observou-se o resultado dessas mudanças:

Tabela 8 – Modelos $\hat{Z}_\lambda$ para os valores de S_2

modelo	λ	t (hrs)	n	posto	posto máximo
$\hat{Z}_{622}$	622.7071	4.498	91	9	40
$\hat{Z}_{551}$	551.5849	4.816	95	11	40
$\hat{Z}_{488}$	488.5858	4.015	99	15	40
$\hat{Z}_{432}$	432.7822	5.091	103	21	40
$\hat{Z}_{383}$	383.3522	5.229	106	29	40
$\hat{Z}_{339}$	339.5678	5.368	109	40*	40
$\hat{Z}_{300}$	300.7842	5.718	112	40*	40
$\hat{Z}_{266}$	266.4302	4.501	115	40*	40

* Representa que o posto da decomposição foi limitado pelo argumento `rank.max`.

O aumento do número de iterações fez com que todos os modelos alcançassem o limite de convergência. Novamente alguns modelos, os últimos três, tiveram o posto limitado. O tempo variou aproximadamente de 4 a 5 horas e 40 minutos, crescendo conforme o posto aumenta, com exceção do terceiro modelo. Já o número total de iterações aumentou em todos os casos.

Continuando o processo, o posto máximo foi definido como 50 e os três últimos modelos foram testados novamente e apenas o primeiro não foi truncado.

Tabela 9 – Modelo $\hat{Z}_{339.1}$

modelo	λ	t (hrs)	n	posto	posto máximo
$\hat{Z}_{339.1}$	339.568	8.087	110	46	50

Comparando esse modelo com o mesmo para `rank.max` $= 40$, percebeu-se que foram necessárias praticamente a mesma quantidade de iterações e quase 3 horas a mais para que o algoritmo convergisse para seu posto real. Então novamente aumentou-se o posto máximo, dessa vez para 80 e apenas o modelo com $\lambda = 300.7842$ convergiu sem limitações.

Tabela 10 – Modelo $\hat{Z}_{300.1}$

modelo	λ	t (hrs)	n	posto	posto máximo
$\hat{Z}_{300.1}$	300.784	8.633	113	77	80

Observando a forma como o posto vinha crescendo era esperado que para $\lambda = 266.4302$ o ajuste necessitasse de um posto máximo maior do que 100. Por isso foram escolhidos `rank.max` $= 200$ e `maxit` $= 250$. Como o tempo de execução dos algoritmos estava se tornando inviável, e com o interesse de verificar a economia de tempo relacionada ao uso de *warm start*, decidiu-se aplicar o modelo com $\lambda = 266.4302$ utilizando $\hat{Z}_{339.1}$ e $\hat{Z}_{300.1}$ como *warm starts*. Porém o algoritmo foi interrompido durante as primeiras iterações por um erro de alocação de memória. Esse mesmo erro aconteceu para outros modelos cujo `rank.max` $= 200$, deixando claro que os computadores utilizados não foram capazes de lidar com um problema dessa dimensão, provavelmente por não possuírem memória *RAM*

suficiente.

Para resolver esse problema, calculou-se $\hat{Z}_\lambda$ para $\lambda = 266.4302$, decrescendo o argumento `rank.max` até algum valor, no qual o modelo não fosse interrompido por falta de memória. Para 180 e 150 o posto ainda foi um problema. Com `rank.max` = 130 conseguiu-se executar o algoritmo, com $\hat{Z}_{339.1}$ e $\hat{Z}_{300.1}$ como pontos iniciais.

.

Tabela 11 – Modelos $\hat{Z}_\lambda$ para $\lambda = 266.4302$ e $\lambda = 235.8180$ com diferentes *warm starts*

modelo	λ	t(hrs)	n	posto	posto máximo	*warm start*
$\hat{Z}_{266.1}$	266.4302	1.2321	9	124	130	$\hat{Z}_{339.1}$
$\hat{Z}_{266.2}$	266.4302	0.7114	5	107	130	$\hat{Z}_{300.1}$
$\hat{Z}_{235.1}$	235.8180	1.6206	12	130*	130	$\hat{Z}_{339.1}$
$\hat{Z}_{235.2}$	235.8180	1.2483	9	130*	130	$\hat{Z}_{300.1}$

* Representa que o posto da decomposição foi limitado pelo argumento `rank.max`.

Percebeu-se que com o auxílio de um ponto inicial todos esse modelos se tornaram extremamente eficientes, levando pelo menos 7 horas a menos, que o modelo mais lento sem *warm start* ($\hat{Z}_{300.1}$).

Notou-se também que apesar desses modelos possuírem o mesmo λ, o uso de *warm starts* diferentes resultou, em postos diferentes, isto é, não resultou em mínimo global e sim em mínimos locais. Como esse problema de otimização é considerado convexo, isso implica que o problema de completar matrizes utilizando relaxamento convexo pela norma nuclear não foi resolvido. Então como o limite de convergência utilizado no algoritmo, $thresh = 10^{-5}$, parece não ser pequeno o suficiente, ele foi redefinido como 10^{-6}. A tabela 12 mostra os últimos ajustes com o novo $thresh = 10^{-6}$.

Tabela 12 – Modelos $\hat{Z}_\lambda$ para $\lambda = 266.4302$ e $\lambda = 79.0791$ com diferentes *warm starts* e $thresh = 10^{-6}$.

modelo	λ	t(hrs)	n	posto	posto máximo	*warm start*
$\hat{Z}_{266.3}$	266.4302	16.9810	142	123	130	$\hat{Z}_{266.1}$
$\hat{Z}_{266.4}$	266.4302	17.6152	148	122	130	$\hat{Z}_{266.2}$
$\hat{Z}_{79.1}$	79.0791	20.5210	159	140*	140	$\hat{Z}_{266.1}$
$\hat{Z}_{79.2}$	79.0791	19.6416	150	140*	140	$\hat{Z}_{300.1}$

* Representa que o posto da decomposição foi limitado pelo argumento `rank.max`.

Observando os primeiros 2 ajustes, percebeu-se que mesmo diminuindo o limiar, ainda assim os resultados para modelos com mesmo λ foram diferentes. Todavia nesse caso as diferenças entre os postos foi de apenas uma unidade. Como critério de decisão para escolher o melhor dentre os dois ajustes, como não havia sido feita previsão ainda, foi observado qual modelo tinha a menor função objetiva. O escolhido foi $\hat{Z}_{266.3}$ com o valor de 0.94955 uma diferença de 0.00003 para $\hat{Z}_{266.4}$. Por último, os modelos com ($\lambda = 79$) foram executados com *rank.max* maior, mas mesmo assim eles foram limitados e resultaram em

posto 140.

3.5 Previsão das Avaliações faltantes

Nesse tópico foram utilizados os modelos escolhidos para fazer previsão das avaliações faltantes. A previsão foi realizada em dois casos diferentes para cada modelo, um com o ajuste original e outro com pós processamento como explicado na seção 2.4 retirando o viés causado pelo encolhimento dos valores singulares. Esse pós processamento foi feito através da função `deBias()` do pacote "softImpute".

- `deBias()`
  - `x`: A matriz com valores faltantes da classe "Incomplete".
  - `svdObject`: Um resultado da função `softImpute()`, isto é um modelo $\hat{Z}_\lambda$.

A saída da função é uma nova estimativa $\hat{Z}_\lambda$ na forma de uma *SVD*.

A previsão foi feita de forma similar à realizada em Mazumder et al. (2010).

Tanto a remoção do viés, quanto a previsão foram realizadas utilizando o banco de sondagem, porém é preciso dividir esses dados em duas partes. Para repartir os dados, foram selecionados aleatoriamente 10^5 pares de coordenadas desse conjunto, dos quais foram obtidas as respectivas notas do conjunto de treinamento. As observações restantes, 1308395 avaliações, foram usadas para a previsão. Esses passos foram dados com o auxílio das funções do tipo `join` do pacote "dplyr". Este foi escolhido por conta da sua simplicidade e eficiência para criar novos conjuntos e subconjuntos dos dados.

Com esses dados separados, a parte menor foi usada como argumento para a função `deBias()`, resultando nos modelos $\tilde{Z}_\lambda$. Enquanto a parte maior foi usada para a previsão através da função `impute()`.

- `impute()`
  - `object`: Um resultado da função `softImpute()`, isto é um modelo $\hat{Z}_\lambda$.
  - `i`: um vetor indicando as coordenadas das linhas de cada nota a ser prevista.
  - `j`: um vetor indicando as coordenadas das colunas de cada nota a ser prevista.

A saída de `impute()` é um vetor de previsões, criado a partir da reconstrução da matriz de posto baixo representada pela *SVD* $\hat{Z}_\lambda$. Para o resultado de `impute()` vamos dar o nome de **notas previstas**. Enquanto as notas verdadeiras para esses pares de coordenadas, que estão contidas no banco de treinamento, vamos chamar de **notas**. Então pode-se dizer que Δ é o conjunto que contém os índices das notas observadas escolhidas para previsão. Com esses dois vetores definidos, podemos calcular o erro preditivo, usando a fórmula do

Erro Quadrático Médio (*RMSE*) descrita abaixo:

$$\text{RMSE} = \sqrt{\frac{1}{|\Delta|}\sum_{i\in\Delta}(\text{notas} - \text{notas previstas})^2} \tag{3.1}$$

A tabela 13 mostra os *RMSE* tanto para os modelos $\hat{Z}_\lambda$ quanto para $\tilde{Z}_\lambda$ e também outros resultados.

Tabela 13 – Modelos $\hat{Z}_\lambda$ e $\tilde{Z}_\lambda$ e seus Erros Quadráticos Médio

modelo	λ	posto inicial	*RMSE* inicial	posto final	*RMSE* final
$\hat{Z}_{6253}$	6253.5010	1	3.1146	2	2.6044
$\hat{Z}_{2097}$	2097.0453	2	2.4073	3	2.0685
$\hat{Z}_{703}$	703.2220	8	1.8426	9	1.6419
$\hat{Z}_{622}$	622.7071	9	1.7923	10	1.6019
$\hat{Z}_{551}$	551.5849	11	1.7461	12	1.5619
$\hat{Z}_{488}$	488.5858	15	1.7017	16	1.5230
$\hat{Z}_{432}$	432.7822	21	1.6587	22	1.4842
$\hat{Z}_{383}$	383.3522	29	1.6180	30	1.4467
$\hat{Z}_{339.1}$	339.568	46	1.5759	47	1.4054
$\hat{Z}_{300.1}$	300.784	77	1.5352	78	1.3608
$\hat{Z}_{266.1}$	266.4302	124	1.5095	125	1.3395
$\hat{Z}_{266.2}$	266.4302	107	1.5048	108	1.3330
$\hat{Z}_{235.1}$	235.8180	130*	1.4713	130	1.3149
$\hat{Z}_{235.2}$	235.8180	130*	1.4656	130	1.3023
$\hat{Z}_{266.3}$***	266.4302	123	1.3713	124	1.2239
$\hat{Z}_{266.4}$***	266.4302	122	1.3700	123	1.2227
$\hat{Z}_{26}$	26.5183	40*	1.0929	40	1.0474
$\hat{Z}_{8}$	8.8926	40*	1.0611	40	1.0335
$\hat{Z}_{2}$	2.9821	40*	1.0508	40	1.0291
$\hat{Z}_{79.1}$***	79.0791	140*	1.0017	140	0.9049
$\hat{Z}_{79.2}$***	79.0791	140*	0.9998	140	0.9031

* Representa que o posto da decomposição foi limitado pelo argumento `rank.max`.
*** Indica os modelos que utilizaram um limiar de convergência menor igual a 10^{-6}

Com o uso do `deBias()` percebeu-se que todos os modelos ganharam um aumento de uma unidade no posto, com exceção daqueles que foram limitados por `rank.max()`. Além disso em todos os ajustes ao aplicar `deBias()` o erro diminui, provando a necessidade desse pós processamento. Na maioria dos casos o Erro Quadrático Médio, para os dois tipos de ajustes, decresceu conforme aumentou-se o posto, sugerindo que quanto mais dimensões melhor a aproximação. Porém isso não foi verdade em todos os casos, para $\hat{Z}_{266.1}$ e $\hat{Z}_{266.2}$, que tem mesmo λ e postos 124 e 107, respectivamente, a *SVD* de menor posto resultou no menor *RMSE*. O mesmo ocorreu para os modelos $\hat{Z}_{266.3}$ e $\hat{Z}_{266.4}$. A tabela também mostrou, que mesmo para modelos com posto baixo, se λ for pequeno é possível obter um erro baixo, como foi o caso dos modelos truncados por `rank.max` $= 40$. Por fim os ajustes

com melhores resultados, $RMSE = 0.9049$ e 0.9031, foram aqueles com maior posto, 140, e $\lambda = 79.0791$, um dos menores valores usados para o parâmetro de regularização.

3.6 Considerações

Uma outra maneira de se escolher os melhores parâmetros de regularização seria através de validação cruzada. Porém, nesse estudo, a forma como essa validação foi planejada encontrou algumas dificuldades. Para validação seria utilizado o método *kfold* com $k = 5$, isto é, o banco dados de treinamento seria dividido em 5 partes mutuamente exclusivas, das quais uma parte seria o conjunto de teste para a realização da previsão, enquanto as 4 restantes formariam um conjunto para construir os modelos $\hat{Z}_\lambda$, isto é, aplicar o algoritmo "Soft-Impute" para diferentes λ's. Esse processo seria realizado 5 vezes alternando as partes que compõem cada conjunto, de modo que todas as 5 partes fossem utilizadas para previsão. A dificuldade está ao dividir os dados de treinamento, pois se fossem retiradas observações de forma aleatória, algum usuário com poucas avaliações, poderia ficar sem representação no conjunto de treinamento, isto é, todas as notas que esse usuário deu iriam parar no conjunto de teste. Dessa forma não haveria como prever as notas para aquele cliente, uma vez que o modelo não teria informação sobre ele.

A solução encontrada foi retirar uma amostra de usuários, ao invés de avaliações. Então seria realizado o *k-fold*, porém a previsão, não seria feita com a função `impute()`, e sim de forma similar à descrita na seção 2.5.2.
Os usuários selecionados seriam representados na forma de uma matriz esparsa. Em cada etapa da previsão, uma avaliação de cada pessoa seria transformada em 0 e as restantes seriam usadas para prevê-la. Para deixar mais claro, pode-se pensar um exemplo com 1 usuário, com o vetor de notas $\mathbf{j} = [4, 0, 2, 1, 0]$. Então a previsão seria feita da seguinte forma, primeiro transforma-se o primeiro avaliação do vetor em 0, $\mathbf{j} = [0, 0, 2, 1, 0]$, para então fazer a previsão dessa nota através do produto $\mathbf{j}VV^T$. Esse processo seria repetido, apenas para os valores observados, armazenando a previsão das notas zeradas, para por fim calcular o *RMSE* usando as notas observadas e as previsões. Devido a complexidade dessa abordagem, no momento da implementação dessa metodologia no `R` surgiram vários problemas de alocação de memória. Por esse motivo e levando em consideração que essa forma de validação cruzada foi pensada perto do prazo de entrega, não foi possível aplica-la.

4 Conclusão

A partir da análise dos dados da *Netflix*, foi possível perceber que a maioria dos clientes da empresa, do fim de 1998 até o último dia de 2005, estava satisfeito com os filmes presentes no catálogo. Ficou claro também que o gênero preferido dos usuários foi de ação. Olhando os dados agrupados por usuários, percebeu-se que alguns avaliaram quase todos os filmes, trazendo o questionamento sobre a veracidade de certas avaliações, uma vez que é pouco provável que qualquer pessoa tenha assistido tantos filmes.

Foi possível entender a importância da *SVD* em problemas de completar matrizes, uma vez que mesmo usando um modelo bem rudimentar como o descrito na seção 2.5.1, foi possível encontrar padrões escondidos nos dados, nesse caso gêneros de filmes e grupos de usuários.

Tornou-se claro ao longo do estudo, que a execução do algoritmo só foi possível graças às estruturas especiais de armazenamento de matrizes esparsas. Especificamente o **CSC**, formato utilizado pela função `Incomplete()`.

Com relação ao algoritmo, mostrou-se que o uso de "aquecimentos" acelera muito o tempo de convergência do modelo, contudo, houveram alguns ajustes dispendiosos, chegando a demorar mais de 20 horas.

Apesar dos ajustes terem alcançado o limite de convergência, como foram encontrados diferentes postos para modelos com mesmo λ, sabe-se que o problema de regularização não foi resolvido. Ou seja, os limites 10^{-5} e 10^{-6} não foram suficientemente pequenos para que o problema fosse convexo. Logo é necessário executar o algoritmo para limites menores.

Mesmo sem encontrar mínimo global, alguns mínimos locais se mostraram adequados para previsão dos dados, isto é, alcançaram valores baixos para o Erro Quadrático Médio. No qual o menor deles, 0.9031, foi menor do que para o *Cinematch*, algoritmo usado pela *Netflix* à época.

APÊNDICE A – Script do R

Esse apêndice contém os códigos para obtenção dos resultados desse estudo.

O Banco de dados utilizado está disponível nesse link:
http://academictorrents.com/details/9b13183dc4d60676b773c9e2cd6de5e5542cee9a

O script do R está abaixo:

```
#### pacotes necessários
require(softImpute)
require(data.table)
require(pryr)
require(dplyr)
require(lattice)
require(xtable)
############### importando dados de treinamento ###############
## especificar diretório onde estão os dados no computador.
setwd("C:\\Users\\Juliano\\Desktop\\TCC-2.2016
\\Netflix dados tcc\\download\\training_set")

#lista os arquivos com mv
file_list<-list.files(pattern = "mv*")
#le todos os arquivos através de um loop
data_list = lapply(file_list, read.table, sep = ",",skip=1)
####### salvando
save(data_list,file="data_list.Rdata")
load("data_list.Rdata")

dim(data_list)
names(data_list)
summary(data_list)
str(data_list)

### reescrever banco de dados no formato de coordenadas,
### para transformar em uma matriz esparsa
for(a in 1:length(file_list)){
xx<-data.frame(rep(a,nrow(data_list[[a]])),data_list[[a]][,-3])
```

```
write.table(xx, file="mmm.txt", append=T,col.names = F,row.names = F)
}


####### agora ler o banco de dados no formato adequado
setwd("C:\\Users\\Juliano\\Desktop\\TCC-2.2016\\market matrix")

#lista os arquivos com nome mmm.txt
file_list<-list.files(pattern = "mmm.txt*")
### importa o banco de dados em um data.table
######## li o arquivo no formato de coordenada
importa<-fread("mmm.txt")
####### salvando
save(importa,file="importa.Rdata")
load("importa.Rdata")
##verificando o numero de usuários,
##uma vez que o numero de usuários é menor que o numero de ids.
length(unique(importa$V2))

##ordenando dados com relação aos usuários.
xo<-importa[order(importa$V2),]

##criar um vetor para substituir o vetor de usuários.
xox<-data.frame(table(xo$V2))
usuarios<-rep(1:length(xox$Var1),xox$Freq)
####### salvando
save(usuarios,file="usuarios.Rdata")
load("usuarios.Rdata")

## usando pacote para transformar os dados em uma matriz esparsa.
matespar<-Incomplete(i=usuarios,j=xo$V1,x=xo$V3)
####### salvando
save(matespar,file="matespar.Rdata")
load("matespar.Rdata")



################### importando dados de sondagem ###################
## especifica diretório dos dados do netflix
```

```
setwd("C:\\Users\\Juliano\\Desktop\\TCC-2.2016\\
Netflix dados tcc\\download\\")

list.files()
avaliados<-readLines("probe.txt")
## Leia o arquivo como um vetor de caracteres
length(avaliados)
ind.filmes = grep(":", avaliados)
length(ind.filmes)
## Remova o ":"

avaliados[ind.filmes] = sub(":","",avaliados[ind.filmes])

# Converte em numérico

avaliados = as.numeric(avaliados)

# Cria uma matriz com duas colunas com os índices da matriz
# onde os filmes foram avaliados.
# Para isso adicione um último elemento ao ind.filmes
# que é o comprimento de avaliados mais 1.

num.filmes = length(ind.filmes)
ind.filmes.m = c(ind.filmes, length(avaliados)+1)

gerar.matriz = function(num.filme) {
cbind(rep(avaliados[ind.filmes[num.filme]] ,
ind.filmes.m[num.filme+1] - ind.filmes[num.filme] - 1 ),
avaliados[(ind.filmes[num.filme] + 1):(ind.filmes.m[num.filme+1]-1)] )
}

avaliados.mat = sapply( 1:length(ind.filmes), gerar.matriz)


############ criando banco de dados de sondagem

setwd("C:\\Users\\Juliano\\Desktop\\TCC-2.2016\\market matrix")

load("importa.Rdata")
```

```
##ordenando dados com relação aos usuários.
xo<-importa[order(importa$V2),]
####### salvando
save(xo,file="xo.Rdata")
load("xo.Rdata")

user<-xo$V2
####### salvando
save(user,file="user.Rdata")
load("user.Rdata")

#### comparando coluna de usuários para id antigo e id novo
load("usuarios.Rdata")
compara<-cbind(user,usuarios)
head(compara,n=5)


us_fil<-do.call("rbind",avaliados.mat)
us_fil<-us_fil[order(us_fil[,2]),]

compara1<-unique(compara[,1])


##### vetor de usuarios correta
usu.final<-match(us_fil[,2],compara1)

##### criando data.frame com coluna de usuarios e filmes
us_fil[,2]<-usu.final
us_fil<-data.frame(us_fil)
names(us_fil)<-c("filmes","Usuarios")
head(usu.final)
head(us_fil)
#####salvando
save(us_fil,file="us_fil.Rdata")
load("us_fil.Rdata")
```

```
#####arrumando banco de dados de treinamento
##### para análise exploratória
load("xo.Rdata")
head(xo)
load("usuarios.Rdata")
xo$V2<-usuarios
dados.exp<-xo
colnames(dados.exp)<-c("filmes","Usuarios","notas")
head(dados.exp)
save(dados.exp,file="dados.exp.Rdata")
load("dados.exp.Rdata")
#### conjunto filmes e notas
dados.leng<-dados.exp[,c(1,3)]
save(dados.leng,file="dados.leng.Rdata")
load("dados.leng.Rdata")
#### conjunto usuarios e notas
dados.lengu<-dados.exp[,c(2,3)]
save(dados.lengu,file="dados.lengu.Rdata")
load("dados.lengu.Rdata")
#####arrumando banco de dados de sondagem
#####para análise exploratória

us_fil<-data.table(us_fil)
probe<-inner_join(dados.exp,us_fil)
probe<-data.table(probe)
save(probe,file="probe.Rdata")

################# Exemplos metodologia ###############
########################################################
options(digits = 5)
a<-matrix(c(1,3,4,5,0,0,0,1,3,4,5,2,0,1,1,3,4,5,
0,0,0,0,0,0,0,4,5,2,0,0,0,0,4,5,2),nrow=7)
b<-svd(a)

breconstruc<-round(b$u[,1:2],digits = 2)%*%
round(diag(b$d[1:2]),digits=2)%*%round(t(b$v)[1:2,],digits=2)

round(breconstruc,digits = 2)
### previsao exemplo
```

```
round(c(4,0,0,0,0)%*%round((b$v)[,1:2],digits=2)%*%
round(t(b$v)[1:2,],digits=2),digits = 2)

round(c(4,1,0,0,0)%*%round((b$v)[,1:2],digits=2)%*%
round(t(b$v)[1:2,],digits=2),digits = 2)

round(c(0,1,4,3,0)%*%round((b$v)[,1:2],digits=2)%*%
round(t(b$v)[1:2,],digits=2),digits = 2)

round(c(0,1,3,0,5)%*%round((b$v)[,1:2],digits=2)%*%
round(t(b$v)[1:2,],digits=2),digits = 2)

################################################

###############análise exploratória###################

####medidas de tendencia central :
###media mediana max min
resumo<-summary(dados.exp$notas)
#moda
moda<-function(x){
ux<-unique(x)
ux[which.max(tabulate(match(x,ux)))]
}
moda.notas<-moda(dados.exp$notas)
## numero de observações
nnotas<-length(dados.exp$notas)
### tabela de frequência
tabe<-table(dados.exp$notas)
save(tabe,file="tabe.Rdata")
xtable(data.frame(tabe))
xtable(tabe)

###### tabelas das primeiras e ultimas 5 observaçõoes
###### de cada conjunto

aa<-rbind(head(probe[,1:2,with=F],n=5),
tail(probe[,1:2,with=F],n=5))
```

```
xtable(aa)

bb<-rbind(head(dados.exp,n=5),tail(dados.exp,n=5))
xtable(bb)


############## medidas de dispersão das avaliações
#variancia das avaliações
var(dados.exp$notas)
#desvio
desvio<-sd(dados.exp$notas)


###################histogramas do treinamento e do probe


load("probe.Rdata")
load("dados.exp.Rdata")
load("tabe.Rdata")
barplot(tabe/sum(tabe), xlab = "Avaliações", ylab="Proporção",
col="blue",horiz = F,ylim=c(0,0.40))

tabep<-table(probe$notas)
barplot(tabep/sum(tabep), xlab = "Avaliações", ylab="Proporção",
col="blue",horiz = F,ylim=c(0,0.40))


################
################ histogramas das médias
media.u<-aggregate(dados.exp[,c(2,3)],list(dados.exp$Usuarios),mean)
load("media.u.Rdata")
media.f<-aggregate(dados.exp[,c(1,3)],list(dados.exp$filmes),mean)
load("media.f.Rdata")
histogram( ~ notas, data=media.u, xlab = "Avaliações",
ylab="Contagem",col="gray")

histogram( ~ notas, data = media.f, xlab = "Avaliações",
ylab="Contagem",col="gray")
```

```
##### estudo dos filmes
leng.filme<-aggregate(dados.leng,list(dados.leng$filmes),length)
load("leng.filme.Rdata")
length(leng.filme$Group.1)
which.max(leng.filme$notas)
leng.filme$notas[5317]
which.min(leng.filme$notas)
leng.filme$notas[13755]
ordenado<-order(leng.filme$notas)
leng.filme<-leng.filme[order(leng.filme$notas),]
####filmes com menor numero de avaliações
head(leng.filme,n=5)
####filmes com maior numero de avaliações
tail(leng.filme,n=5)

####em média cada filme recebeu x avaliações, var
mean(leng.filme$notas)
var(leng.filme$notas)

####em média cada usuário avaliou y filmes, var
leng.usu<-aggregate(dados.lengu,list(dados.lengu$Usuarios),length)
head(dados.lengu)
load("leng.usu.Rdata")
mean(leng.usu$notas)
var(leng.usu$notas)

####usuarios que mais e menos avaliaram
leng.usu<-leng.usu[order(leng.usu$notas),]
head(leng.usu)
tail(leng.usu)


#################### PREVISÃO ####################
#### indices dos dados  de tamanho 100000 para usar deBIAS
sdeb<-sample_n(us_fil,1e+05)
save(sdeb,file = "sdeb.Rdata")
load("sdeb.Rdata")
```

```
####

load("probe.Rdata")
###sdeb com notas
sdebn<-inner_join(probe,sdeb)
sdebn<-Incomplete(sdebn$Usuarios,sdebn$filmes,sdebn$notas)
save(sdebn,file="sdebn.Rdata")
#### porção dos dados de tamanho 100000 para deBias
load("sdebn.Rdata")
####

#### porção dos dados restantes
sprev<-anti_join(probe,sdeb)
save(sprev,file="sprev.Rdata")
load("sprev.Rdata")
####

probe.notas<-probe$notas
save(probe.notas,file="probe.notas.Rdata")
load("probe.notas.Rdata")

####### modelos softimpute: modelo genérico####
#setwd("C:\\Users\\aluno\\Desktop\\rodar")
load("matespar.Rdata")

### codigo para todos os modelos,
### basta mudar os argumentos da sequência
### e da função softimpute
lamseq=exp(seq(from=log(703),log(236),length=10))

lam5<-lamseq[5]
warm=NULL
limi<-10^(-5)
start.time <- Sys.time()
ajuste1.5=softImpute(matespar,lambda=lam5,rank.max=40,
warm=warm,trace.it = T,maxit=200,thresh =limi)

end.time <- Sys.time()
time1.5<- end.time - start.time
```

```
save(time1.5,file="time1.5.Rdata")
save(ajuste1.5,file="ajuste1.5.Rdata")


#### depois de gerar os modelos,
#### para cada modelo seguir os passos abaixo
#### substituindo os argumentos conforme o modelo

#####carregar modelo
load("ajuste1.Rdata")
#### ajuste 1, lambda=6253.501, rank.max=40
#### e rank=1.

###### previsao
prev1<-impute(ajuste1,i=sprev$Usuarios,j=sprev$filmes)
save(prev1,file="prev1.Rdata")
load("prev1.Rdata")
######

#### raiz do erro quadrático médio
rmse1<-sqrt((sum((sprev$notas-prev1)^2))/length(sprev$notas))
save(rmse1,file="rmse1.Rdata")
load("rmse1.Rdata")
####


### tirando o viés
ajusted1<-deBias(sdebn,ajuste1)
save(ajusted1,file="ajusted1.Rdata")
load("ajusted1.Rdata")
###

###### previsao do sem vies
prev1d<-impute(ajusted1,i=sprev$Usuarios,j=sprev$filmes)
save(prev1d,file="prev1d.Rdata")
load("prev1d.Rdata")
######

#### raiz do erro quadrático médio do sem vies
```

```
rmse1d<-sqrt((sum((sprev$notas-prev1d)^2))/length(sprev$notas))
save(rmse1d,file="rmse1d.Rdata")
load("rmse1d.Rdata")
```

Referências

[1] Leskovec, J; Rajaraman, Anand; D. Ullman, J (2014) *Mining of Massive Datasets*, 2nd ed., Cambridge University Press.

[2] Candès, J; Recht, B (2008) *Exact matrix completion via convex optimization*, Foundations of Computational Mathematics, 9:717–772.

[3] Mazumder, R; Hastie, T; Tibshirani, R (2010) *Spectral Regularization Algorithms for Learning Large Incomplete Matrices*, Journal of Machine Learning Research, 11:2287–2322.

[4] Candès, J; Tao, T (2009) *The power of convex relaxation: near-optimal matrix completion*,IEEE Transactions on Information Theory, 56(5):2053–2080.

[5] Srebro, N; Rennie, J (2010) *Fast maximum margin matrix factorization for collaborative prediction*, Proceedings of the 22nd International Conference on Machine Learning, pages 713–719. ACM.

[6] Keshavan, H; Oh, S; Montanari, S (2009) *Matrix completion from a few entries*, IEEE Transactions on Information Theory, 56(6):2980–2998.

[7] Srebro, N; Jaakkola, T (2003) *Weighted low-rank approximations*, Proceedings of the 20th International Conference on Machine Learning, pages 720–727, AAAI Press.

[8] Recht, B; Fazel, M; A. Parrilo, P (2007) *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, Disponível em http://www.citebase.org/abstract?id=oai:arXiv.org:0706.4138.

[9] Fazel, M (2002) *Matrix Rank Minimization with Applications*, PhD thesis, Stanford University.

[10] Cai, E; Candès, J; Shen, Z (2008) *A singular value thresholding algorithm for matrix completion*, Disponível em http://www.citebase.org/abstract?id=oai:arXiv.org:0810.3286.

[11] Donoho, D; Johnstone, I; Kerkyachairan, G; Picard, D (1995) *Wavelet shrinkage; asymptopia? (with discussion)*, Journal of the Royal Statistical Society: Series B, 57:201–337.

[12] Ma, S; Goldfarb, D; Chen, L (2011) *Fixed Point and Bregman Iterative Methods for Matrix Rank Minimization*, Mathematical Programming Series A. 128 (1): 321-353.

[13] Bai, Z; Demmel, J; Dongarra, J; Ruhe, A; van der Vorst, H (2000) *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.*, SIAM, Philadelphia.

[14] Jack Dongarra (1995) *Survey of Sparse Matrix Storage Formats.* Netlib Repository. URL http://netlib.org/.

[15] Feuerverger, A; He, Y; Khatri, S (2012) *Statistical Significance of the Netflix Challenge.* Statistical Science, 27 (2): 202–231.

[16] R Development Core Team (2009) *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

[17] Bhatta, R; Malla, S *R Inheritance.* Programiz. URL https://www.programiz.com/r-programming/inheritance.