3 Modelos de Fatoração Matricial

3.1 Introdução

Algoritmos baseados em fatoração matricial são as implementações de maior sucesso dos modelos de fatoração latente. Em termos gerais, esta técnica é capaz de caracterizar tanto itens quanto usuários através de vetores de variáveis latentes inferidas de padrões de avaliação dos itens. Uma alta correspondência entre as variáveis latentes de um item e um usuário levam a uma recomendação. Ela tem se tornado popular nos últimos anos por conseguir combinar uma boa escalabilidade com precisão nas predições. Alem disso, oferece grande flexibilidade para modelar diversas situações do dia-a-dia.

Os sistemas de recomendação dependem de diversos tipo de dados de entrada, os quais são geralmente estruturados numa matriz onde uma dimensão representa os usuários e a outra os itens de interesse. O objetivo nesses sistemas é prever como os usuários avaliariam um item que eles ainda não avaliaram para então conseguir recomendar itens para os mesmos. Assumindo, por exemplo, que temos 5 usuários e 10 itens, e que as avaliações são 1 para as positivas e -1 para as negativas, a matriz que representa o interesse dos usuários pelos itens toma a forma abaixo (um hífen significa que o usuários ainda não avaliou o item).

	I1	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**
U1	1	-	-1	1	-	-1	-1	-	1	-
U2	1	-	1	-	1	1	1	1	-1	-
U3	-1	1	-	-1	-	-1	1	-	-	1
U4	-	1	-1	1	1	-	1	-	-1	1
U5	1	-1	1	-	-1	-	-1	-1	1	-1

Tabela 3.1 - Um exemplo de matriz usuário item

Os dados mais convenientes para o bom funcionamento desses sistemas são os que o usuário fornece explicitamente como, por exemplo, avaliações sobre seu interesse ou desinteresse por um item em questão. Pilászy e Tikk [38] sugerem que mesmo um número pequeno de avaliações é melhor do que uma vasta quantidade de atributos e metadados dos itens ou dos usuários. A Netflix

coleta estes tipos de dados para seus vídeos através de um sistema de avaliação por estrelas em seu site numa escala que varia de 1 a 5. No TiVo, os usuários indicam suas preferências por séries de TV pressionando botões que indicam avaliações positivas ou negativas sobre o conteúdo assistido. Esses dados explícitos fornecidos pelos usuários geram uma matriz esparsa, já que um único usuário tende a avaliar apenas uma pequena parcela dos itens consumidos.

Assim, a tarefa de prever as avaliações que faltam pode ser considerada como preencher as células vazias (marcadas com hífen) com valores que fossem consistentes com as demais avaliações existentes.

A intuição por traz do uso da técnica de fatoração de matrizes para a resolução deste problema é a de que devem haver algumas variáveis latentes que determinam como um usuário avalia um item. Por exemplo, dois usuários avaliariam positivamente um filme se ambos gostam os atores daquele filme, ou se o filme é um filme de ação, o qual é um gênero que os dois gostam. Então, se conseguirmos descobrir estas características latentes, deveríamos conseguir também prever uma avaliação para um determinado usuário a cerca de um item, pois essas variáveis latentes associadas a um usuário deveriam casar com as de um dado item.

O ponto forte dessa abordagem é que ela permite a incorporação de fontes de informação adicionais. Quando as avaliações explicitas dos usuários não estão disponíveis, esses sistemas podem inferir as preferências dos usuários usando dados implícitos, os quais refletem indiretamente sua opinião através da observação de seu comportamento, como seu histórico de compras, seus vídeos vistos, sua navegação pelo site, suas buscas, ou até mesmo os movimentos do seu mouse. Esses dados implícitos denotam a presença ou ausência de determinados eventos e também são utilizados no preenchimento da matriz.

3.2 Modelo Matemático

O conjunto $U = \{u_1, u_2, ..., u_p\}$ define os usuários do sistema de filtragem colaborativa e o conjunto $V = \{v_1, v_2, ..., v_p\}$ os itens disponíveis. O conjunto T é chamando de conjunto de treino:

$$T \subset U \times V \times O$$

com *O* representado os possíveis valores para as avaliações. O conjunto *T* define todas as avaliações conhecidas consistindo de um usuário, um item e a avaliação que o usuário deu para um item.

A função *e* explica o mapeamento de um usuário $u_i \in U$ e um item $v_j \in V$ para uma avaliação r_{ij}.

$$e : U \times V \rightarrow O$$

$$e(u_i, v_j) = r_{ij}$$

A função *e* é estimada por um modelo *ê* que prevê as avaliações para um dado par usuário-item. O modelo *ê* está relacionado com o correto mapeamento *e* por

$$\hat{e}(u_i, v_j) = e(u_i, v_j) + Z$$

onde *Z* é o erro entre a predição e a avaliação de fato. Geralmente, supõe-se que *Z* segue uma distribuição normal

$$Z \sim N(\mu, \sigma^2)$$

com media μ e variância σ^2.

A matriz de avaliações *R*, dada por

$$R = (r_{ij})_{i=1,\ldots,p,\ j=1,\ldots,q}$$

é a matriz com todas as avaliações, em que não existem células vazias. Na maioria dos casos esta matriz é desconhecida, pois dificilmente em um sistema de recomendação todos os usuários avaliam todos os itens. Geralmente, conhecemos apenas um sub-conjunto desta matriz.

Como falamos anteriormente, o objetivo de um sistema de recomendação é recomendar novos itens que um usuário pode estar interessado. Para isso, um preditor é necessário para aprender a partir das avaliações fornecidas de forma a

prever as avaliações que não foram dadas ainda. Desta forma, uma possível avaliação positiva pode ser dada por um usuário para um item e, por esta razão, o sistema recomendaria este item para o usuário.

Alguns sistemas de recomendação definem o erro médio quadrático (*RMSE*, do inglês *Root Mean Square Error*) como ferramenta para medir os erros de predição. Dessa forma, para um dado conjunto dados de teste $Q \subset U \times V$ podemos dizer que temos como objetivo minimizar a parte quadrática do *RMSE*.

$$\sum_{(u_i, v_j) \in Q} (\hat{e}(u_i, v_j) - r_{ij})^2 \rightarrow \min$$

Treinar $\hat{e}$ requer bons ajustes de $\hat{e}$. Estes ajustes poderiam ser encontrados minimizando o RMSE para o conjunto de treino. No entanto, utilizar apenas o conjunto de dados de treino para esta validação pode nos levar a um excesso ou especialização do treinamento.

3.3 Fatoração Matricial Incremental

O uso de fatoração matricial foi sugerido por Brandyn Webb (também conhecido por Simon Funk) em seu blog [18] durante a competição Netflix Prize [19]. Esta técnica está fortemente relacionada à decomposição de valores singulares (*SVD*, do inglês *Singular Value Decomposition*) [20] [22] [24].

A idéia por trás de uma abordagem de *SVD* clássica é decompor uma matriz de avaliações *P* de dimensão $n \times m$ em um produto de três matrizes,

$$P = A \Sigma B^T$$

onde *A* possui dimensão $n \times n$, Σ possui dimensão $n \times m$ e *B* possui dimensão $m \times m$. As matrizes *A* e *B* são ortogonais. Σ é uma matriz diagonal com *k* entradas diferentes de zero. Assim sendo, as dimensões efetivas das matrizes *A,* Σ e *B* são $n \times k$, $k \times k$ e $k \times m$, respectivamente. Estas *k* entradas diagonais ς_i da matriz Σ são todas positivas com $\varsigma_1 \geq \varsigma_2 \geq ... \geq \varsigma_k \geq 0$. As colunas de *A* e *B* são, respectivamente, chamadas de autovetores a esquerda e a direita de *P*.

Se pegarmos apenas os $r << k$ valores singulares mais significativos, o

produto destas três matrizes será uma matriz de posto *r*, a qual referenciamos por *P′* e é a melhor aproximação de *P* com posto *r* considerando a norma de Frobenius.

$$\|P' - P\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} \left| p'_{ij} - p_{ij} \right|^2$$

O uso da técnica *SVD* também traz à tona alguns problemas quando aplicada ao domínio aqui proposto. Geralmente, a matriz de avaliações *R* é uma matriz esparsa e a técnica apresentada não funciona com este tipo de matriz. Algumas adaptações tem sido feitas para minimizar este problema, como preencher as entradas em branco com zeros ou utilizando outras técnicas mais sofisticadas [21] [23]. Um segundo problema é o fato de *R* ser, tipicamente, grande. Calcular o *SVD* para matrizes grandes requer uma enorme capacidade de processamento computacional.

A abordagem incremental para o *SVD* proposta por Simon Funk funciona consideravelmente bem e é também fácil de se implementar, endereçando diretamente estes problemas. Trata-se de um algoritmo de gradiente descendente que calcula a aproximação utilizando apenas os valores conhecidos de *R*.

A técnica é bem parecida com o *SVD* original. Ao invés de decompor *R* em três matrizes, focamos apenas em duas matrizes, as matrizes características.

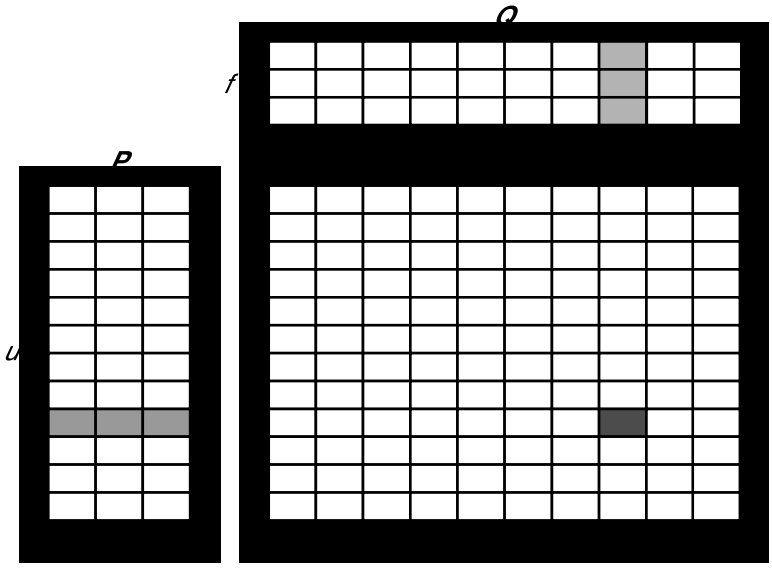

Figura 3.1 - Fatoração matricial incremental

$$R \approx PQ = R'$$

Retornando para o nosso domínio de recomendações, podemos formular o problema da seguinte forma: Suponhamos que temos uma matriz R (u x v) contemplando as avaliações dos usuários para os itens, e onde u é o número de usuários e v o número de itens. E assumamos que queremos considerar f variáveis latentes. Queremos calcular uma aproximação R' da matriz R, tal que $\|R' - R\|_F$ seja minimizada, e $R' = P_{u \times f}\left(Q_{v \times f}\right)^T$. A i-ésima linha da matriz P é o vetor de preferências do usuário i, e a j-ésima linha da matriz Q é o vetor de características do item j. Dessa forma, conseguimos extrair uma aproximação para os dados desejados, e podemos usá-los para preencher os valores desconhecidos da matriz R calculando o produto escalar das preferências do usuário pelas características do item

$$\hat{e}(u_i, v_j) = p_i^T q_j = \sum_{k=1}^{f} p_{ik} q_{kj}$$

onde p_{ik} e q_{kj} são componentes dos vetores de preferências do usuário i e do item j, respectivamente. Em seguida, podemos calcular o erro quadrático entre a estimativa e a avaliação de fato:

$$\varepsilon_{ij}^2 = \left(r_{ij} - \hat{e}(u_i, v_j)\right)^2 = \left(r_{ij} - \sum_{k=1}^{f} p_{ik} q_{kj}\right)^2$$

O algoritmo de aprendizado que pode minimizar este erro, $\varepsilon_{ij}^2 \rightarrow \min$, utiliza o gradiente descendente estocástico para diferenciar a equação em relação às diferentes variáveis latentes e obter as equações de atualização

$$p'_{ik} = p_{ik} + \alpha \frac{\partial \varepsilon_{ij}^2}{\partial p_{ik}} = p_{ik} + 2\alpha \varepsilon_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial \varepsilon_{ij}^2}{\partial q_{kj}} = q_{kj} + 2\alpha \varepsilon_{ij} p_{ik}$$

onde α é taxa de aprendizado. Este número controla o quanto do erro entra no passo de atualização do valor da variável latente em questão. O valor de α deveria ser consideravelmente pequeno, algo em torno de 0.001. Com a definição

das equações de atualização, podemos definir o algoritmo base abaixo. Esta abordagem [25, 26, 27] combina uma fácil implementação com um tempo de execução relativamente rápido. Sua pseudo-implementação pode ser conferida a seguir.

Algoritmo 1 - Algoritmo base

Entrada: Matriz de avaliações R e número de variáveis latente f.

1: Inicialize as matrizes P e Q pela primeira vez com um número fixo ou de forma randômica
2: **para** $k = 1$ até f **faça**
3: **repita**
4: **para** $\forall(u_i, v_j, r_{ij}) \in T$ **faça**
5: calcule ε_{ij}.
6: atualize p_{ik} e q_{kj}.
7: **fim**
8: recalcule $\sum_{(u_i, v_j, r_{ij})} \varepsilon_{ij}^2$
9: **até** alguma condição de convergência ser atingida
10: **fim**

Saída: Matrizes P e Q com os vetores de variáveis latentes.

Algoritmo 1 - Versão básica

Existem diversas variações deste algoritmo. Alguns sugerem treinar uma variável latente de cada vez, outras consideram treinar todas simultaneamente. Esta última forma tende a convergir mais rápido com relação a quantidade total de iterações necessárias. No entanto, em nossa implementação notamos que, apesar de precisarmos de uma quantidade maior de iterações, o primeiro método foi mais rápido pelo fato de conseguirmos otimizar o código cacheando as variáveis latentes conforme elas forem sendo calculadas. Uma vantagem do treino simultâneo é que a variância de cada variável latente não se altera tanto quanto no caso do treino independente. Isto significa que a primeira variável latente acaba tendo um peso bem maior na predição enquanto que as demais tentam apenas prever resíduos menores.

3.4 O Fator de Regularização

Alguns incrementos a este modelo de fatoração matricial tem sido propostos. Um deles consiste em adicionar um fator de regularização β às equações de atualização. A inclusão deste parâmetro visa suprimir o problema de

sobre-treino e, portanto, melhorar a performance das predições. As equações de atualização ficam assim:

$$p'_{ik} = p_{ik} + \alpha\left(\frac{\partial \varepsilon_{ij}^2}{\partial p_{ik}} - \beta p_{ik}\right) = p_{ik} + \alpha\left(2\varepsilon_{ij} q_{kj} - \beta p_{ik}\right)$$

$$q'_{kj} = q_{kj} + \alpha\left(\frac{\partial \varepsilon_{ij}^2}{\partial q_{kj}} - \beta q_{kj}\right) = q_{kj} + \alpha\left(2\varepsilon_{ij} p_{ik} - \beta q_{kj}\right)$$

E a equação de erro passa a ser:

$$\varepsilon_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{f} p_{ik} q_{kj}\right)^2 + \frac{\beta}{2}\sum_{k=1}^{f}\left(p_{ik}^2 + q_{kj}^2\right) = \left(r_{ij} - \sum_{k=1}^{f} p_{ik} q_{kj}\right)^2 + \frac{\beta}{2}\left(\|p\|^2 + \|q\|^2\right)$$

Chamamos $\beta\frac{1}{2}\sum_{k=1}^{f}\left(p_{ik}^2 + q_{kj}^2\right)$ de termo de regularização. Assim como a variância das variáveis latentes tende a cair durante o treino, a do parâmetros de regularização βp_{ik} e βq_{kj} também cai à medida em que uma nova variável latente é treinada. Isto significa que a adição deste termos introduz um erro artificial às equações, especialmente para as primeiras variáveis e sua alta variância. E este erro decai com o passar do tempo. Resumindo, este fator desloca o peso na predição das primeiras variáveis latentes para as seguintes e o algoritmo pode treinar com mais iterações e com menos sobre-treino.

3.5 Adicionando Vieses ao Modelo

Um dos benefícios de se utilizar uma abordagem de fatoração matricial para filtragem colaborativa é a flexibilidade em lidar com diversos aspectos dos dados do contexto e outros requisitos específicos do domínio de aplicação.

Muito das variações observadas nos valores das avaliações é devido a vieses associados aos usuários ou aos itens em questão, independente de qualquer interação. Por exemplo, alguns dados de sistemas de filtragem colaborativa demonstram uma tendência sistemática de alguns usuários a avaliarem os itens melhor do que outros usuários. Dessa forma, alguns produtos são percebidos como melhores, ou piores, que outros.

Ao invés de explicar uma avaliação por uma interação na forma $p_i^T q_j$, tentamos identificar a porção destes valores que os vieses dos usuários e dos itens poderiam explicar, deixando para as matrizes *P* e *Q* apenas a porção da interação em si.

Paterek [26] sugere a adição desses vieses ao modelo de predição através da inclusão de duas variáveis, uma para cada item e outra para cada usuário, no cálculo de $\hat{e}$. Estas constantes são treinadas simultaneamente. Propomos uma pequena modificação no cálculo de $\hat{e}$. Para cada variável latente, disponibilizamos duas variáveis da forma proposta por Paterek e treinamos as mesmas juntamente com as matrizes de variáveis latentes dos usuários e itens.

$$\hat{e}(u_i, v_j) = \sum_{k=1}^{f} p_{ik} q_{kj} + b_i + b_j$$

Assim, as avaliações observadas são quebradas em três componentes: vieses dos itens, dos usuários e a interação entre o usuário e o item. Isto permite que cada componente indique o seu peso na avaliação prevista.

As equações de atualização para os elementos das matrizes de variáveis latentes dos usuários e itens permanecem inalteradas. As equações de atualização para b_i e b_j são

$$b'_i = b_i + \chi\left(\frac{\partial \varepsilon_{ij}^2}{\partial b_i} - \delta b_i\right) = b_i + \chi\left(2\varepsilon_{ij} - \delta b_i\right)$$

$$b'_j = b_j + \chi\left(\frac{\partial \varepsilon_{ij}^2}{\partial b_j} - \delta b_j\right) = b_j + \chi\left(2\varepsilon_{ij} - \delta b_j\right)$$

onde χ é a taxa de aprendizagem e δ um outro parâmetro de regularização.

A equação para o erro de predição incluindo os novos parâmetros de vieses e regularização passa a ser

$$\varepsilon_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{f} p_{ik} q_{kj} + b_i + b_j\right)^2 + \frac{\beta}{2}\left(\|p\|^2 + \|q\|^2\right) + \frac{\delta}{2}\left(b_i^2 + b_j^2\right)$$

Como esses componentes de viés tendem a ter um peso grande na

predição das avaliações, uma boa precisão em sua modelagem é fundamental.

3.6 Adicionando Dinâmicas Temporais ao Modelo

Até então, os modelos apresentados são todos atemporais. No mundo real, a percepção e a popularidade de um item está mudando constantemente a medida que novos itens surgem. Da mesma forma, as preferências dos usuários evoluem, levando-os a redefinir seu gosto. Sendo assim, o sistema deve levar em consideração esses efeitos temporais no mapeamento da natureza dinâmica e variável das interações entre usuários e itens.

A abordagem de fatoração matricial se adapta bem à modelagem desses efeitos temporais, permitindo que tenhamos uma precisão ainda maior na predição das avaliações. Decompor as avaliações em termos distintos permite ao sistema tratar cada efeito temporal separadamente. Especificamente, podemos considerar os seguintes efeitos: os vieses dos usuários $b_i(t)$ mudam com o tempo, assim como os vieses dos itens $b_j(t)$ e as preferências dos usuários $p_i(t)$. Por outro lado, não é de se esperar uma significativa variação temporal nas características dos itens $q_j(t)$, já que os mesmos são estáticos por natureza.

Neste trabalho, demos uma atenção especial à variação temporal da relevância dos itens. Por exemplo, vídeos podem se tornar populares ou impopulares devido a eventos externos como, por exemplo, a aparição de um determinado ator em uma nova novela. Assim, tratamos os vieses de um item $b_j(t)$ em nosso modelo de predição como uma função do tempo.

Koren [28] sugere dividir os vieses de um item em fragmentos baseados no tempo. Durante cada período de tempo em um fragmento tem-se a tradução de uma dada particularidade. A decisão de se dividir a linha do tempo em fragmentos deve balancear o objetivo de riqueza em detalhes (pequenos fragmentos) com a necessidade de um número mínimo de avaliações por fragmento (fragmentos maiores).

$$\hat{e}(u_i, v_j, t) = \sum_{k=1}^{f} p_{ik} q_{kj} + b_i + b_j(t)$$

Um dia *t* estaria associado a um fragmento *Bin(t)* de forma que a particularidade do item seja dividida entre uma parte estacionária e uma que evolui com o tempo:

$$b_j(t) = b_j + b_{j,Bin(t)}$$

Em termos práticos, $b_{j,Bin(t)} \approx MD = B(v \times s)$. B contempla as variações temporais para os itens, onde *v* é o número de itens e *s* o número de fragmentos temporais. A *j*-ésima linha da matriz *M* é o vetor de características do item *j* e a *b*-ésima linha da matriz *D* é o vetor de características temporais do fragmento *b*.

Podemos extrair uma aproximação para o efeito temporal sobre o item desejado pela equação $b_{j,Bin(t)} = m_j^T d_b = \sum_{k=1}^{f} m_{jk} d_{kb}$ onde $t \subset b$.

A equação para o erro de predição incluindo os novos parâmetros de vieses e regularização passa a ser

$$\varepsilon_{ij}^2(t) = \left(r_{ij}(t) - \sum_{k=1}^{f} p_{ik} q_{kj} + b_i + b_j + b_{j,Bin(t)} \right)^2 + \frac{\beta}{2}\left(\|p\|^2 + \|q\|^2 \right) + \frac{\delta}{2}\left(b_i^2 + b_j^2 + b_{j,Bin(t)}^2 \right)$$

A partir do erro calculado, utilizamos também o gradiente descendente estocástico para diferenciar a equação que contempla as dinâmicas temporais dos itens em relação às suas variáveis latentes e obter as equações de atualização

$$m'_{jk} = m_{jk} + \theta\left(\frac{\partial \varepsilon_{ij}^2}{m_{jk}} - \rho m_{jk} \right) = m_{jk} + \theta\left(2\varepsilon_{ij} d_{kb} - \rho m_{jk} \right)$$

$$d'_{kb} = d_{kb} + \theta\left(\frac{\partial \varepsilon_{ij}^2}{d_{kb}} - \rho d_{kb} \right) = d_{kb} + \theta\left(2\varepsilon_{ij} m_{jk} - \rho d_{kb} \right)$$

onde θ é a taxa de aprendizagem e ρ um outro parâmetro de regularização.

A fragmentação dos parâmetros funciona bem para mapearmos a evolução temporal da relevância dos itens. No caso dos usuários, o desafio é maior. É necessário uma resolução detalhada dos vieses dos mesmos para detectar pequenos efeitos temporais. No entanto, é baixa a expectativa por um número suficiente de avaliações dos usuários para cada fragmento de tempo.

Nesse caso, diferentes formas têm sido consideradas para parametrizar o comportamento temporal dos usuários [28, 29, 31].

3.7 O algoritmo final

Como veremos no próximo capítulo, o conjunto de dados utilizado em nossos experimentos possui algumas peculiaridades que precisaram ser levadas em consideração para que o algoritmo tivesse uma performance aceitável.

Uma das alterações feitas consistiu em definir um número mínimo $e_{\min}$ de passos de treino para cada variável latente, assim como um número máximo $e_{\max}$. Estes dois limites são dinamicamente ajustados dependendo do número de variáveis latentes que estamos treinando. Isto é necessário porque caso contrário o algoritmo encerraria o treino logo após alguns poucos passos. Para as últimas variáveis, é necessário um número de passos de treino maior para garantir que houveram melhorias nas predições (redução do *RMSE*).

A melhoria no *RMSE* é medida em cinco passos, ou seja, a diferença no *RMSE* para a variável latente *k* e o passo *e* é dada por:

$$\Delta_k^e = RMSE_k^{e-5} - RMSE_k^e$$

Um mínimo de 5 passos de treino é necessário para cada variável latente.

No caso de o número mínimo de passos de treino ser atingido e o *RMSE* parar de cair, o algoritmo encerra o cálculo daquela variável e parte para a seguinte.

Por fim, armazenamos os valores das variáveis latentes conforme as mesmas iam sendo calculadas para acelerar o cálculo das demais. O algoritmo de treino final é apresentado a seguir.

Algoritmo 2 - Algoritmo final

Entrada: Matriz de avaliações *R*, número de variáveis latentes *f* e número de segmentos temporais *b*.

1: inicialize as matrizes *P* e *Q* pela primeira vez com um número fixo ou de

forma randômica
2: inicialize os vetores de vieses dos usuários b_i e itens b_j com um número fixo ou de forma randômica
3: inicialize as matrizes *M* e *D* de dinâmicas temporais para os itens com um número fixo ou de forma randômica
4: **para** $k = 1$ até f **faça**
5: **para** $e = 1$ até $e_{\max}$ **faça**
6: **para** $\forall(u_i, v_j, r_{ij}) \in T$ **faça**
7: calcule ε_{ij}.
8: atualize p_{ik} e q_{kj}.
9: atualize b_i e b_j
10: **para** $t = 1$ até b **faça**
11: atualize $m_{j,}$ e d_{kt}
12: **fim**
13: recalcule $\sum_{(u_i, v_j, r_{ij})} \varepsilon_{ij}^2$ e o *RMSE*
14: recalcule Δ_k^e
15: **se** $\Delta_k^e < 0.0001$ **ou** $e > e_{\min}$
16: **interrompa**
17: **fim**
18: armazene o valor da variável calculada
19: **fim**

Saída: Matrizes *P* e *Q* com os vetores de variáveis latentes, vetores b_i e b_j com os vieses dos itens e usuários e matrizes *M* e *D* contemplando o efeito temporal nos vieses de um item.

Algoritmo 2 - Versão final com viés e dinâmicas temporais