Pontifícia Universidade Católica
do Rio de Janeiro

Felipe João Pontes da Cruz

Sistemas de recomendação utilizando Máquinas de Boltzmann restritas

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio

Orientador: Prof. Ruy Luiz Milidiú

Rio de Janeiro
Fevereiro de 2016

Pontifícia Universidade Católica
do Rio de Janeiro

Felipe João Pontes da Cruz

Sistemas de recomendação utilizando Máquinas de Boltzmann restritas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Ruy Luiz Milidiú
Orientador
Departamento de Informática — PUC–Rio

Prof. Daniel Schwabe
Departamento de Informática - PUC–Rio

Prof. Hélio Cortês Vieira Lopes
Departamento de Informática - PUC–Rio

Prof. Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 23 de Fevereiro de 2016

Felipe João Pontes da Cruz

Graduou–se em no curso de Bacharelado em Informática na PUC-Rio

Ficha Catalográfica

Felipe João Pontes da Cruz

Sistemas de recomendação utilizando Máquinas de Boltzmann restritas / Felipe João Pontes da Cruz; orientador: Ruy Luiz Milidiú. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2016.

v., 52 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. Aprendizado de máquinas. 3. Filtragem colaborativa. 4. Sistemas de recomendação. 5. Máquinas de Boltzmann restritas. I. Milidiú Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 510

Agradecimentos

Agradeci a minha família, filhas, esposa, mãe e irmãs pelo apoio ao longo dessa etapa. Também agradeço aos amigos que fiz ao longo do mestrado.

Resumo

Felipe João Pontes da Cruz; Milidiú Ruy Luiz. **Sistemas de recomendação utilizando Máquinas de Boltzmann restritas**. Rio de Janeiro, 2016. 52p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Sistemas de recomendação aparecem em diversos domínios do mundo real. Vários modelos foram propostos para o problema de predição de entradas faltantes em um conjunto de dados. Duas das abordagens mais comuns são filtragem colaborativa baseada em similaridade e modelos de fatores latentes. Uma alternativa, mais recente, foi proposta por Salakhutdinov em 2007, usando máquinas de Boltzmann restritas, ou RBMs. Esse modelo se encaixa na família de modelos de fatores latentes, no qual, modelamos fatores latentes dos dados usando unidades binárias na camada escondida das RBMs. Esses modelos se mostraram capazes de aproximar resultados obtidos com modelos de fatoração de matrizes. Nesse trabalho vamos revisitar esse modelo e detalhar cuidadosamente como modelar e treinar RBMs para o problema de predição de entradas vazias em dados tabulares.

Palavras–chave

Aprendizado de máquinas; Filtragem colaborativa; Sistemas de recomendação; Máquinas de Boltzmann restritas

Abstract

Felipe João Pontes da Cruz; Milidiú Ruy Luiz(Advisor). **Recommender systems using Restricted Boltzmann machines**. Rio de Janeiro, 2016. 52p. MSc Thesis — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Recommender systems can be used in many problems in the real world. Many models were proposed to solve the problem of predicting missing entries in a specific dataset. Two of the most common approaches are neighborhood-based collaborative filtering and latent factor models. A more recent alternative was proposed on 2007 by Salakhutdinov, using Restricted Boltzmann Machines. This models belongs to the family of latent factor models, in which, we model latent factors over the data using hidden binary units. RBMs have shown that they can approximate solutions trained with a traditional matrix factorization model. In this work we'll revisit this proposed model and carefully detail how to model and train RBMs for the problem of missing ratings prediction.

Keywords

Machine learning; Collaborative filtering; Recommender systems; Restricted Boltzmann Machines

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Nesse trabalho iremos olhar para o problema de recomendação modelado com dados tabulares, que são uma forma natural de representarmos relações entre elementos de 2 conjuntos distintos. Os elementos de um dos conjuntos são representados pelas linhas enquanto os outros são representados pelas colunas. De forma mais concreta, podemos representar em uma tabela, por exemplo, como um conjunto de usuários avaliam um conjunto de itens. Na falta de dados explícitos, como uma preferência informada pelo usuário, as entradas da tabela podem conter números que representam informação obtida de forma indireta, como por exemplo, quanto tempo um usuário visualizou um determinado item. Na figura a seguir podemos observar concretamente uma instância de dados tabulares:

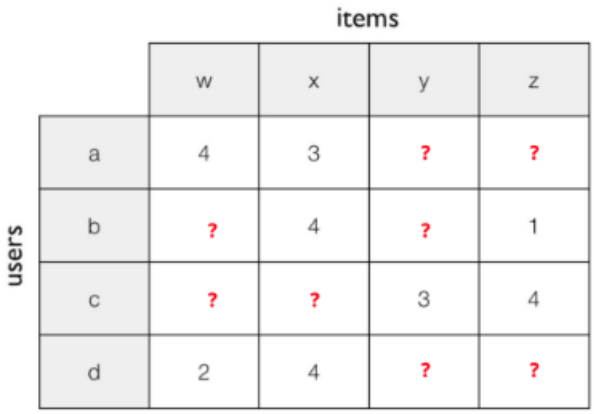

Figura 1.1: Dados fictícios sobre preferência de usuários por itens

Naturalmente, surgiu a necessidade de se usar a informação disponível na predição de entradas faltantes na tabela de dados. Dado que temos informações das preferências de um usuário por alguns itens, podemos usar essa informação para descobrir qual seria a preferência desse mesmo usuário para outros itens. A partir de então, esse tipo de problema passou a ser chamado de *Problema de recomendação*. Entendemos o problema de recomendação como uma instância de um problema de aprendizado de dados tabulares. Como muitos trabalhos acabaram focando em recomendação e utilizaram conjuntos de dados voltados para recomendação, o problema de recomendação tornou-se um dos mais populares dentre os problemas de aprendizado de dados tabulares.

Diversas empresas investiram em pesquisa para o problema de recomendação e isso acabou gerando um grande avanço tanto acadêmico quanto nos resultados obtidos, atingindo baixos níveis de erro mesmo para conjuntos de dados muito grandes. Empresas como Amazon, Netflix, Spotify e outras, utilizam um algoritmo de recomendação como parte integrante de seus produtos. Com o aumento da concorrência e dado o potencial econômico, essa área passou a ter grande importância na indústria, o que acabou estimulando mais pesquisa na área. Ainda hoje, as empresas continuam investindo nessa área com objetivo de ajudar seus clientes a encontrarem os produtos que eles realmente querem, poupando-os de gastar tempo com coisas que não são de seu interesse.

Só o mercado nacional de lojas eletrônicas tem aproximadamente 450 mil lojas, muitas das quais poderiam aumentar sua receita com alguma inteligência em recomendação de produtos. Algo que há pouco tempo atrás poderia ser um privilégio de poucas e grandes empresas, logo pode se tornar mais um commodity com grande potencial econômico.

Chamamos de *Sistema de recomendação* o sistema que recebe como entrada esses dados de preferências, aprende os padrões implícitos nos dados e consegue determinar um valor de preferência, dado um usuário e item arbitrários. Nesse trabalho, o foco é no núcleo do sistema, pois nosso interesse é apenas em desenvolver um método para aprender e predizer essas preferências. Vamos explorar um modelo chamado Máquinas de Boltzmann restritas (*Restricted Boltzmann Machine*) para implementar um sistema de recomendação. Esse modelo é um modelo do início dos anos 80 mas apenas recentemente descobriu-se um método eficiente de treino, que permitiu o seu estudo e uso em diversos problemas, inclusive no problema de aprendizado de dados tabulares e recomendação.

Esse modelo recebeu muita atenção nos últimos anos por ser o componente com as quais poderíamos montar um tipo de rede profunda, chamadas redes de confiança profunda - Deep beleief networks [13]. Esse modelo surgiu como uma das primeiras alternativas viáveis de redes profundas justamente no momento em que as redes neurais voltaram a receber mais atenção e outros tipos de redes, como as convolucionais, também emergiam como modelos de grande potencial. Esse momento gerou uma boa produção científica com as máquinas de Boltzmann restritas e um dos caminhos pesquisados foi a sua utilização em um problema de dados tabulares. Por esses motivos a escolha do tema do trabalho tenta reunir um modelo moderno com um problema extremamente popular e com resultados publicados.

No mundo real os melhores resultados, geralmente, são obtidos com

combinações de diferentes modelos. Um dos maiores exemplos já vistos de combinações de modelos foi visto na entrada vencedora do prêmio Netflix usada pelo time "BellKor"[4]. O modelo apresentado nesse trabalho foi um dos modelos utilizado nessa combinação e algumas variações dele também foram usadas. Esse modelo é uma alternativa um pouco mais recente a outros métodos e existe uma carência de material detalhado sobre como eles funcionam e como eles devem ser treinados. A publicação que é principal referência desse trabalho [18] deixa alguns detalhes importantes sem uma explicação extremamente precisa do que deve ser feito principalmente no espaço que existe entre o modelo matemático e a implementação exata do mesmo.

Também, temos uma outra característica marcante, que é a sua versatilidade, já que pode ser usado em diversas tarefas de aprendizado de máquinas como aprendizado não supervisionado de atributos [7] e classificação [17]. Reforçaremos a versatilidade do modelo, justamente no tema do trabalho apresentado: aprendendo dados tabulares. Atualmente o estudo de métodos de aprendizado não supervisionado tem recebido muita atenção o que torna a técnica utilizada relevante nos dias atuais.

Ainda que exista muita atividade nessa área, ainda existem muitos desafios. Por um lado, a quantidade de dados coletados cresce vertiginosamente e as fontes muitas vezes são heterogêneas, o que pode deteriorar a qualidade dos dados coletados. Conseguir manter o erro em níveis baixos, generalizando para quantidade de dados cada vez maiores é um desafio enorme. A escalabilidade para conjuntos de dados cada vez maiores acaba tornando alguns métodos mais estudados que outros pois são mais viáveis. Nesse sentido o método descrito nesse trabalho também figura como um método de baixa complexidade computacional que pode ser implementado com poucas operações matriciais e portanto interessante também economicamente.

Ainda que um caminho inicial tenha sido trilhado ainda existe muito a se pesquisar. Por exemplo, existe o que chamamos de recomendação sensível ao contexto, onde por exemplo, podemos considerar o fator temporal como contexto de uma recomendação ou até mesmo a geolocalização do usuário em questão. Hoje, além de entender as preferências de usuários por itens, é necessário entender qual o melhor momento de usar esses itens para estimular mais consumo pelos usuários.

A seguir, veremos a formalização do problema de recomendação e algumas técnicas conhecidas para esse tipo de problema. Em seguida, olharemos com mais detalhes o modelo de aprendizado de Boltzmann e a máquina de Boltzmann restrita. No próximo capítulo, veremos como usar uma máquina de Boltzmann restrita para um problema de recomendação e como

ajustar o modelo para obter resultados compatíveis com os métodos de estado da arte. Os resultados serão apresentados no capítulo seguinte, com diversas observações sobre como alcançar os melhores modelos e no final, veremos quais seriam possíveis extensões e problemas que podem ser resolvidos no futuro.

O objetivo desse trabalho é produzir um documento que explique a parte teórica e cubra todos os detalhes práticos que são enfrentados na hora de se obter bons modelos de recomendação. As máquinas de Boltzmann restritas são relevantes academicamente, porque podem resolver várias tarefas com poucas modificações e na prática pois pode ser implementada para GPUs com certa facilidade pois possui um algoritmo de treino computacionalmente simples.

2
Dados tabulares e recomendação

De forma a contextualizar melhor o problema e os modelos existentes, vamos visualizar a taxonomia de famílias de algoritmos e algoritmos para esse tipo de problema. O modelo estudado nesse trabalho se encontra dentro da família de modelo de fatores latentes.

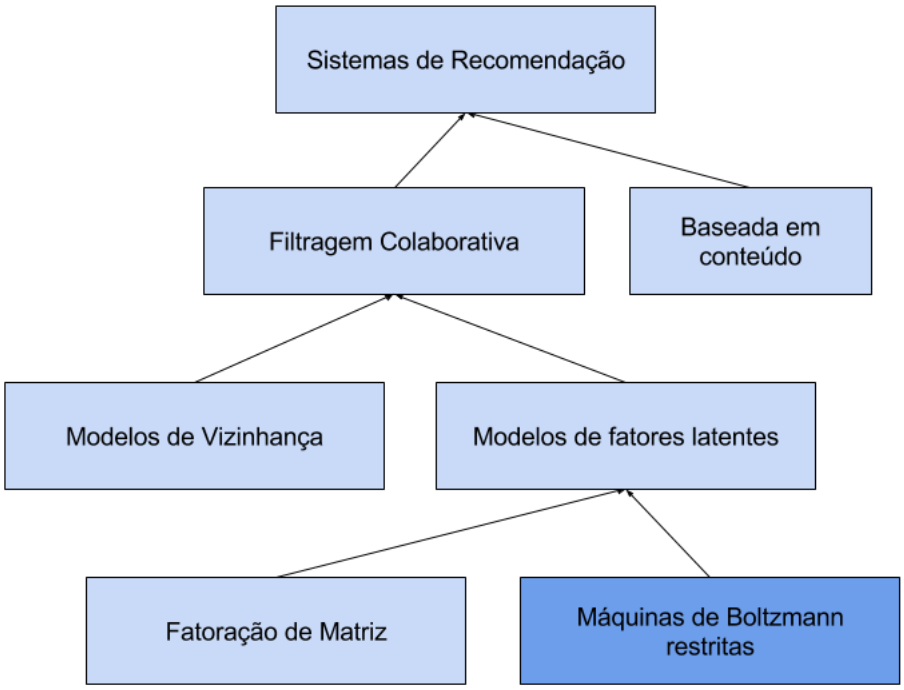

Figura 2.1: Famílias de algoritmos e técnicas para aprendizado de dados tabulares

Nessa taxonomia, existem 3 modelos que trabalham com dados tabulares: modelos de vizinhança, modelos de fatoração de matriz e modelos com máquinas de Boltzmann restritas. Vamos olhar algumas características de cada um deles e um pouco de como eles surgiram. Do ponto de vista matemático, podemos assumir que a entrada desses 3 modelos é uma matriz, que representa

a preferência de usuários por itens. O objetivo dos modelos é usar a informação disponível para calcular entradas faltantes do nosso interesse, sem nenhuma informação adicional.

Formalmente, queremos minimizar a diferença entre as preferências reais a as preferências calculadas pelo modelo. Definimos U como o conjunto de usuários, I com conjunto de itens, $\overline{r_{ij}}$ a predição de preferência de um usuário u para um item i e r_{ij} a preferência conhecida de um usuário u por um item i. Dessa forma, nosso objetivo é ajustar os parâmetros θ do modelo de forma a minimizar a soma de todas as diferenças absolutas, ou erros, entre predições e os dados reais.

$$\min_{\theta} \sum_{u \in U i \in I} |\overline{r_{ui}} - r_{ui}| \tag{2-1}$$

Existem 2 grandes tipos de abordagens quando falamos no problema de recomendação: baseada em conteúdo [3] e filtragem colaborativa [10]. Os modelos baseados em conteúdo tentam aprender, como atributos de produtos e atributos de usuários interagem, para produzir um número que descreva alguma relação, como uma relação de preferência. Nos modelos de filtragem colaborativa, a idéia central é aprender a estimar esse número, utilizando comportamento passado e usuários/itens similares. Nesse trabalho, toda nossa atenção está nos modelos de filtragem colaborativa, já que os dados dos modelos baseados em conteúdo, geralmente, não exploram a natureza tabular dos dados.

O grande sucesso dos métodos de filtragem colaborativa se deve ao fato de que obter apenas um número que represente a preferência de um usuário por um item, é mais barato do que coletar e armazenar diversos atributos para pessoas e produtos. Outro fator relevante é que ele é agnóstico ao tipo de usuário e item sendo utilizados no problema, ou seja, um sistema de recomendação por filtragem colaborativa generaliza com mais facilidade que um sistema baseado em conteúdo. Um ponto que pode ser considerado negativo em relação aos sistemas baseados em conteúdo é exatamente o fato de ele não modelar, de forma geral, atributos conhecidos de usuários ou produtos e que podem tornar o resultado final melhor.

A primeira abordagem para o aprendizado de dados tabulares foi através do uso de uma medida de similaridade. Em alto nível, calculamos a similaridade entre todos os usuários, entre si, e depois usamos usuários semelhantes para determinar a nota de um usuário para um item que ele não interagiu. Esse processo foi implementado como um *framework de recomendação* baseado no cálculo da similaridade entre usuários ou itens e que cujas similaridades são parâmetros no cálculo da preferência. O modelo de filtragem colaborativa

descrito, foi mais tarde chamado de modelo de filtragem colaborativa baseada em vizinhança, em inglês *neighborhood-based*.

Ele possui uma estrutura de alto nível, que possibilitou em 1999, que Herlocker [10] propusesse um *framework algoritmico* de para filtragem colaborativa. Nesse framework, a principal operação é o cálculo da similaridade entre dois usuários ou itens, e com esses valores de similaridade em mãos, fazer a predição é apenas uma conta que pondera similaridades e notas. Sendo um framework, os pontos de modificação são a função de similaridade e a função de predição.

Esses modelos, por sua vez, podem ser vistos sob dois pontos de vista: baseados em usuários ou baseados em itens. Quando falamos em filtragem colaborativa *user-based* a similaridade é calculada em cima de usuários distintos e quando falamos em *item-based* é porque a similaridade foi calculada usando itens distintos. Na prática, a implementação fica sendo totalmente agnóstica ao fato de estarmos usando um modelo baseado em usuários ou itens. As explicações do trabalho levam em consideração que estamos falando de um modelo baseado em usuários.

Em 1990, foi apresentado em [2] o primeiro sistema de filtragem colaborativa baseado na correlação de pearson. Basicamente, temos um conjunto de usuários $U = \{u_1, u_2, ..., u_m\}$ e um conjunto de itens $I = \{i_1, i_2, ..., i_n\}$. Temos uma matriz de preferências $R \in \mathbb{R}(m \times n)$, onde as células são definidas como r_{ai}, a preferência que o usuário a determinou para o produto i. Para lidar de forma mais eficiente com o viés que um usuário (ou item) pode ter, diminui-se de r_{ai} a média de preferências do usuário a ou a média de preferências do item i. No nosso exemplo, diminuiremos a preferência média do usuário a e definimos $\overline{r_a}$ como esse valor.

O conjunto I é composto de itens em comum entre os usuários a e u. Com essas definições, temos a nossa função de similaridade como uma função de correlação de Pearson:

$$Sim(a,u) = \frac{\sum_{i \in I}(r_{ai} - \overline{r_a})(r_{ui} - \overline{r_u})}{\sqrt{\sum_{i \in I}(r_{ai} - \overline{r_a})^2 \sum_{i \in I}(r_{ui} - \overline{r_u})^2}} \tag{2-2}$$

Vale lembrar, que a função anterior, é uma função para ser usada em um modelo baseado em usuários, já que calcula a similaridade entre usuários distintos.

Agora, precisamos de um procedimento que faça isso para todos os usuários, entre si, quando eles tiverem interagido com pelo menos 1 elemento em comum. Usuários que interagiram com itens em comuns serão chamados de usuários similares. O procedimento abaixo mostra, como podemos de forma ingênua, preencher a matriz de similaridades.

Algorithm 1 Calcula matriz de similaridades

```
1: procedure SIMILARITIES(U)
2:     M^{m×n} ← 0
3:     for each user u ∈ U do
4:         S ← similarUsers(u)
5:         for each user a ∈ S do
6:             M[u][a] ← similarity(u, a)
```

Tendo em mãos a matriz de similaridades, a predição é a simples aplicação de uma fórmula. Seguindo a mesma lógica anterior, quando precisamos determinar a preferência r_{ai} de um usuário a por um item i, pegamos todos os usuários $u \in K$ o qual temos a informação da preferências pelo item i, diminui-se a preferência média do usuário $\overline{r_u}$ e multiplicamos pela similaridade w_{au} entre o usuário a e o similar u e depois dividimos pela soma de todos os valores de similaridade.

$$p_{ai} = \overline{r_a} + \frac{\sum_{u \in K}(r_{ui} - \overline{r_u}) \times w_{au}}{\sum_{u \in K} w_{au}} \tag{2-3}$$

A função anterior é a peça faltante para a produção das recomendações. Geralmente, são selecionados os K maiores valores de p_{ai} para definirmos que itens serão recomendados para o usuário a. Os usuários mais semelhantes a um usuário qualquer são chamados de vizinhos, logo, são eles que são usados na formula definida anteriormente.

O modelo comentado é apenas um de uma série de modelos possíveis dentro desse framework de filtragem colaborativa. Outras funções de similaridade podem ser usadas assim como estratégias mais eficientes para preencher a matriz de similaridades. Outras funções comuns de similaridade que podemos citar são a função de *Jaccard* e *similaridade de cosenos*. Esse método também é agnóstico ao fato de estarmos usando preferências explícitas ou implícitas mas talvez sejam necessários mais alguns tratamentos adicionais para dados implícitos.

Os métodos de vizinhança, apresentados anteriormente são intuitivos, fáceis de implementar e conseguem generalizar de forma satisfatória. Uma das características mais relevantes desse tipo de modelo é o fato de serem agnósticos com relação ao que é representado nas duas dimensões, geralmente, usuários e itens, pelo fato de exigir apenas um número que signifique alguma força na relação entre o que é representado nas linhas e o que é representado nas colunas.

Outra família de modelos conhecida é a de modelos baseados em fatores latentes. A idéia central desses modelos é aprender alguma estrutura subjacente aos dados através de um tipo de redução de dimensionalidade. Em modelos

dessa família, se busca uma forma de representar tanto os usuários quanto os itens, em uma dimensão menor que a original, de forma que isso se capture pontos em comum entre usuários e itens e se generalize para predições. Apesar do resultado a ser buscado ser o mesmo, a forma de treino é completamente diferente.

Os modelos de fatores latentes tem como hipótese que usuários possuem comportamentos comuns e itens possuem características que geram interesse em grupos de usuários. Conhecendo as preferências dos usuários por um conjunto de itens, pode permitir aprender esses comportamentos comuns entre grupos de usuários e uma forma de se fazer isso é mapear um vetor de preferências para um tamanho bem menor do que o número de itens com o qual ele pode interagir.

Essa representação mais enxuta são os fatores latentes que queremos aprender. Pensando em uma abstração de mais alto nível, é como se quiséssemos modelar as preferências de um usuário, que pode ter interagido com muitos itens, em poucos valores, que capturam informações implícitas sobre os itens. Se estivermos falando de um Filme, talvez, em um dos fatores latentes poderia ser o gênero do filme.

Duas técnicas se encaixam nessa família: os modelos de fatoração de matrizes e os modelos com máquinas de boltzmann restritas, que é o tema desse trabalho.

Na formulação clássica do modelo de fatoração de matrizes, representamos os coeficientes de interação em uma matriz $R \in \mathbb{R}(m \times n)$ sendo m número de usuários e n número de itens e buscamos por duas matrizes menores, $U \in \mathbb{R}(m \times k)$ e $I \in \mathbb{R}(n \times k)$ tal que $R \approx UI^T$. O valor e k é justamente o número de fatores latentes. Na prática, não existe nenhuma regra nem heurística sobre a escolha do valor de k e geralmente ele é determinado em uma etapa de seleção de parâmetros.

Esse modelo pode ser resolvido com métodos de gradiente como em [8] ou com o método dos mínimos quadrados alternados utilizado em [14]. O método de mínimos quadrados alternados é um método bem popular e disponível em diversas ferramentas, justamente pela sua simplicidade e capacidade de generalização.

O modelo de predição, no método de mínimos quadrados alternados, é o produto interno entre o vetor de fatores latentes de um usuário e o vetor de fatores latentes de um item. Mais precisamente:

$$r_{ui} = i_j^T u_i \tag{2-4}$$

Para cada usuário u_i e item i_j queremos encontrar k fatores latentes que

generalizem suas preferências e minimizem o erro, medido como raiz quadrada do erro quadrátco médio. Queremos minimizar a equação a seguir, nossa função de custo, considerando r_{ij} como a entrada na tabela de preferências R, u_i como uma linha da tabela U, i_j como uma linha na tabela I^T e K como pares (i, j) de preferências conhecidas.

$$\min_{U,I} \sum_{(i,j)\in K} (r_{ij} - i_j^T u_i)^2 \tag{2-5}$$

A imagem a seguir mostra visualmente qual o objetivo desse tipo de modelo:

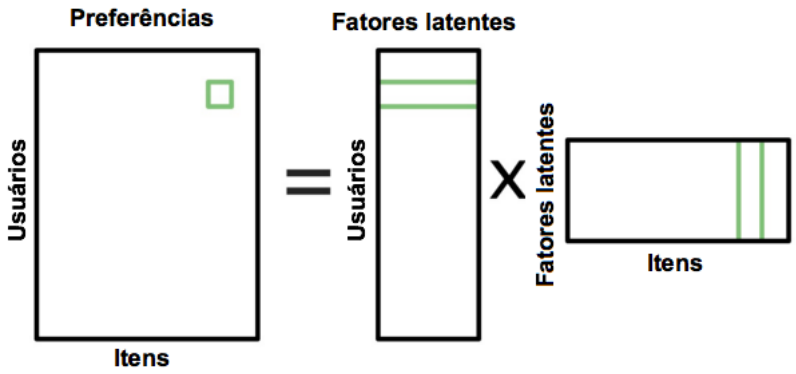

Figura 2.2: Um dos modelos da família de fatores latentes

A matriz de preferências é vista como o produto de duas matrizes, que contem uma entrada para cada usuário e item porém com a dimensão extremamente reduzida, precisamente de tamanho k.

Sendo i^T e u desconhecidos, a função de custo é não convexa, o que implica em vários mínimos locais e ausência e um algoritmo para solução eficiente. Para contornar isso, o método de mínimos quadrados alternados, explora o fato de que se fixarmos ou u ou i, conseguimos resolver a outra icógnita com um método de mínimos quadrados. A idéia é inicializar ambos com valores aleatórios, fixar um deles e encontrar os valores ótimos para o outro em uma etapa. Na próxima etapa, fixamos os valores ótimos encontrados e calculamos os valores ótimos das icognitas atuais.

Para aumentar o poder de generalização desse método, podemos usar um termo de regularização na função de custo. Com a regularização, nossa função de custo passa a ser:

$$\min_{U,I} \sum_{(i,j)\in K} (r_{ij} - i_j^T u_i)^2 + \lambda(||i_j||^2 + ||u_i||^2) \tag{2-6}$$

Uma outra opção para um modelo de fatoração de matrizes é usar a decomposição em valores singulares, ou *Singular Value Decomposition*

chamada de SVD. O grande problema dessa abordagem é que como, geralmente, as matrizes de preferência são esparsas e muitas entradas são desconhecidas, se aplicarmos a decomposição SVD nessa matriz, existe grande chance do sobreajuste(*overfitting*).

Depois de apresentados os modelos de vizinhança e um modelo da família de fatores latentes vamos iniciar o estudo dos modelos que usam as máquinas de Boltzmann restritas. Em 2007, Salakhutdinov [18] mostrou que é possível obter melhores resultados no problema de filtragem colaborativa, usando máquinas de Boltzmann restritas, do que com modelos bem ajustados de fatoração de matrizes. Isso gerou um grande interesse pois era mais um tipo de problema que podia ser resolvido com esse modelo e que produzia resultados comparáveis a de outras técnicas que dominavam até então.

As máquinas de Boltzamnn restritas, em sua versão original e binária, recebem como entrada um conjunto de vetores binários e aprendem a reconstruir esses vetores. Essa capacidade generativa já foi utilizada em problemas de completar partes faltantes de imagens de dígitos, com o famoso dataset MNIST. A capacidade generativa, se da ao fato, de que as unidades escondidas, ou fatores latentes, conseguem identificar padrões nos exemplos de entrada e consegue restaurar pedaços faltantes baseados nos padrões já aprendidos. Podemos interpretar que se 2 usuários se comportam de forma muito parecida, mais um deles não interagiu com um item específico, o modelo consegue reconstruir a preferência baseado nos exemplos que ele viu e portanto, irá predizer uma nota semelhante ao usuário com comportamento similar.

Sem modificações específicas para a tarefa de predição de preferência, uma implementação simples, seria modelar a entrada como um vetor binário $v \in \mathbb{R}^n$ e $v_i \in \{0, 1\}$ onde 1 significa alguma interação do usuário v com o produto i e 0 desconhecido. Nesse modelo, a camada escondida da RBM capturaria a informação implícita na estrutura dos exemplos de um conjunto de dados e conseguiria reconstruir padrões que se assemelham a algum padrão aprendido. A predição binária traria outros itens que ocorrem simultaneamente com os itens de um usuário específico. Ainda que simples, se quisermos predizer preferências, temos que modificar pequenos detalhes para que o erro seja realmente minimizado.

Em 2007 o trabalho de Salakhutdinov [18] mostrou um modelo, usando RBMs, treinado para estimar o coeficiente de preferência de um usuário por um item, assim como nas outras abordagens que vimos anteriormente. Nesse modelo, a utilização das RBMs é um pouco diferente do usual. Os nós de entrada são nós *"softmax"* que representam valores de 1 a 5. Com esses nós é possível aprender a relação de inter-dependência entre os coeficientes de

preferências entre usuários e itens. Um detalhamento mais preciso dos nós *"softmax"* pode ser visto no trabalho de Hinton chamado *"A Practical Guide to Training Restricted Boltzmann Machines"* em [11].

Nessa dissertação, olharemos detalhadamente como aprender dados tabulares usando RBMs.

3
Máquinas de Boltzmann e sua versão restrita

As máquinas de Boltzmann tem como inspiração original conceitos física. Usamos um conceito de sistema e componentes que estão conectadas entre si e onde cada componente possui um estado que influencia no estado dos componentes conectados. A imagem a seguir ilustra um sistema totalmente conectado e com 3 componentes, que chamaremos de unidades, com um estado diferente das demais:

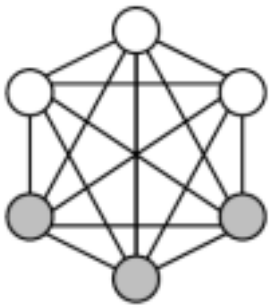

Figura 3.1: Uma máquina de Boltzmann totalmente conectada

Usando uma função de energia, determinamos um valor escalar (de energia) para cada configuração possível do sistema. A partir da função de energia definimos uma probabilidade que significará a probabilidade daquela configuração específica de acontecer. Tendo uma função de energia e uma probabilidade, buscamos ajustar o modelo para que pouca energia represente alta probabilidade e muita energia baixa probabilidade. Se interpretarmos as unidades como atributos e as conexões entre unidades como uma interferência que elas podem exercer entre si, conseguimos usar essa abstração para modelar uma distribuição de probabilidade de um conjunto de dados.

O problema inicial desse tipo de modelo foi a inexistência de um algoritmo de treinamento eficiente. Em 1985, Hinton [1], propôs um algoritmo de aprendizado para esse tipo de modelo. Na prática, ainda existiam problemas práticos com relação ao treinamento, por ser difícil e consumir muitos recursos. Em 1986, Smolensky [21], contornou os problemas práticos das máquinas de Boltzmann propondo restrições na sua topologia. Na prática, Smolensky dividiu o modelo em 2 tipos de unidades: unidade visível e unidade escondida. Na unidade visível as ativações estariam diretamente ligadas aos atributos dos exemplos e a camada escondida seria usada para capturar relações entre

esses atributos. A inexistência de conexões entre unidades da mesma camada, diminui sensivelmente a quantidade de parâmetro dos modelos e viabiliza o procedimento de amostragem usado no aprendizado. Sem as restrições, para realizar uma amostragem de cada unidade, todas as outras teriam que ter seu valor fixado, logo, cada amostragem teria que ser feita individualmente e sequencialmente.

A imagem abaixo ilustra o resultado do modelo original depois de aplicadas as restrições propostas:

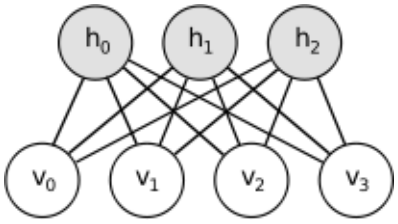

Figura 3.2: Uma máquina de Boltzmann restrita

A partir de então, passou-se a utilizar, em diversas tarefas, as máquinas de Boltzmann restritas, que chamaremos de RBM, do inglês *Restricted Boltzmann Machines*. Nesse momento ainda que houvessem problemas práticos, uma série de resultados posteriores foram conseguidos explorando o benefício das restrições. Mais recentemente, em 2002, Hinton [12] propôs um modelo de aprendizado eficiente e que vem sendo utilizado amplamente na academia e na indústria.

Podemos estudar as RBMs, também, pela ótica da ciência da computação. Nessa ótica, elas podem ser vistas como um tipo específico de *Modelo gráfico probabilístico*, mais precisamente um campo aleatório de Markov. A relação entre RBMs e modelos gráficos probabilísticos nos oferecem uma série de resultados teóricos relevantes e algoritmos bem estudados. Uma RBM é um modelo gráfico probabilístico, não direcionado e bipartido. A topologia de uma RBM é composta de 2 camadas: camada visível e camada escondida. Além de serem bipartidos, todos os nós de uma camada tem uma aresta com todos os nós da outra camada. Cada uma dessas arestas tem um peso associado. A camada visível corresponde aos componentes de um exemplo de entrada, como por exemplo, os pixels de uma imagem. A camada escondida modela a dependência entre os componentes da camada de entrada.

A impossibilidade de uma solução analítica para essa tarefa de aprendizado, faz com que elas tenham que ser treinadas com um método de gradiente, assim como diversos outros métodos de aprendizado de máquinas. A diferença é que o gradiente é uma aproximação, já que o cálculo exato dele leva a um número exponencial de operações. Essa aproximação do

gradiente foi chamada de *Divergencia contrastiva* ou *Contrastive divergence*. Através do arcabouço de modelos gráficos probabilísticos, consegui-se obter essa aproximação usando amostragem de Gibbs.

Antes de entrar na parte teórica vamos observar um simples experimento com uma máquina de Boltzmann restrita que mostra sua capacidade generativa. Nesse pequeno experimento, treinamos o modelo com 1 exemplo que representa um padrão visual, modelado como um vetor binário onde o valor 0 é representado pelo pixel preto e o valor 1 é representado pelo pixel vermelho. A seguir a imagem do exemplo que queremos aprender:

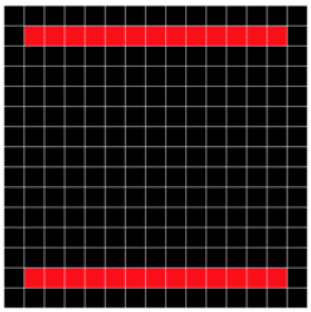

Figura 3.3: Representação visual de um vetor binário com um padrão

Depois de poucas épocas de aprendizado, conseguimos criar um modelo que dado um vetor de entrada vazio, consegue reconstruir o exemplo que foi aprendido. Vale ressaltar que o modelo não limita-se a aprender apenas um padrão mas diversos padrões existentes no conjunto de dados. Obviamente, a quantidade de padrões que pode ser reconhecida e reconstruída com qualidade depende do número de nós na camada escondida, que afeta diretamente no tamanho da matriz de pesos que o modelo contém. A imagem abaixo mostra 2 reconstruções geradas pelo modelo e que são diferentes entre si por que o modelo é estocástico:

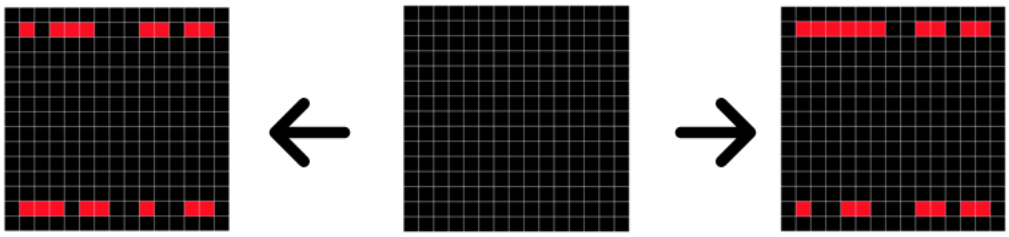

Figura 3.4: Representação visual de reconstruções geradas por uma máquina de Boltzmann restrita

Nos modelos baseados em energia, associa-se um valor escalar a cada configuração possível dos valores de entrada. Como mencionado no início desse trabalho, os atributos de entrada são variáveis binárias $x_i \in \{0, 1\}$. Facilmente

observamos que a quantidade total de configurações distintas com n atributos é 2^n. A tarefa de aprendizado, então, consiste em moldar o formato da função de energia, de forma, que ela consiga representar o conjunto de dados, isto é, influenciar o modelo de forma que retorne probabilidades coerente com os dados vistos no conjunto de dados.

A definição da probabilidade de um vetor de entrada x pela função de energia se dá por:

$$P(x) = \frac{e^{-Energy(x)}}{Z} \tag{3-1}$$

O numerador é exponencial do valor da energia da configuração x. O denominador é um fator de normalização, muitas vezes chamado de *função de partição*, e é a soma dos valores de todas as configurações. Com ele, garantimos que a soma de todas as probabilidades resultam em 1. Mais precisamente, temos:

$$Z = \sum_x e^{-Energy(x)} \tag{3-2}$$

A função de energia é construída com uma técnica de aprendizado de máquinas chamada *produto de experts* [12]. Na prática, temos um produtório de funções aplicadas no vetor de entrada, sendo cada uma dessas funções um "expert" f_i:

$$Energy(x) = \prod_i f_i(x) \tag{3-3}$$

A hipótese do modelo de produto de experts é que podemos modelar uma distribuição de probabilidade num espaço com muitas dimensões, usando uma combinação de "experts"que modelam distribuições mais simples. Até aqui, construímos um modelo capaz de atribuir valores de probabilidade para todas as combinações possíveis dos vetores de entrada.

O que precisamos agora, é incluir os fatores latentes, que enriquecerão o modelo, modelando aspectos que não estão explicitamente representados na distribuição de probabilidade, ou, como recurso para aumentar o poder de expressividade do modelo. Primeiro vamos redefinir a função de energia. Se definirmos o viés das unidades visíveis em um vetor chamado b, os viés da camada escondida c e a matriz de pesos que conecta cada unidade visível a todas unidades escondidas como W, temos a energia como:

$$Energy(x, h) = -\sum_i b_i x_i - \sum_j c_j h_j - \sum_i \sum_j x_i W_{ij} h_j \tag{3-4}$$

Em notação matricial definimos como:

$$Energy(x, h) = -b^T v - c^T h - v^T W h \tag{3-5}$$

Considerando os fatores latentes, podemos devemos alterar 3-1 para:

$$P(x,h) = \frac{e^{-Energy(x,h)}}{Z} \tag{3-6}$$

Repare que Z também tem seu valor alterado para:

$$Z = \sum_{x,h} e^{-Energy(x,h)} \tag{3-7}$$

Como apenas x é observado, podemos marginalizar em h para obter:

$$P(x) = \sum_{h} \frac{e^{-Energy(x,h)}}{Z} \tag{3-8}$$

Ainda que agora, a função de energia tenha 2 parâmetros, ela continuará sendo um produto de experts, mas agora, em função de x e h. A partir daqui, temos um modelo pronto, capaz de atribuir probabilidades a configurações de um conjunto de dados e levando em consideração, também, fatores latentes aprendidos e representados na camada escondida.

O que precisamos agora, é formular o problema de otimização que possibilitara o aprendizado para esse tipo de modelo. O primeiro passo é pensarmos em uma função de verossimilhança de parâmetros θ e conjunto de dados X que seria a multiplicação de todas as probabilidades dos exemplos, assumindo que eles são independentes e identicamente distribuídos:

$$\mathcal{L}(\theta, X) = \frac{1}{N} \prod_{x^{(i)} \in X} P(x^{(i)}) \tag{3-9}$$

Como esse valor tende a tornar-se extremamente pequeno trocamos o produtório por um somatório e adicionamos a função de logarítmo que permite que o resultado mantenha-se igual. Assim como na regressão logística, definimos a função de log-verossimilhança como:

$$\mathcal{L}(\theta, X) = \frac{1}{N} \sum_{x^{(i)} \in X} log\ P(x^{(i)}) \tag{3-10}$$

Esse tipo de função de custo pode ser otimizado com um método de máxima verossimilhança. Sendo a função de custo que queremos otimizar a log-verossimilhança negativa conseguimos otimizá-la com um método de máxima verossimilhança:

$$\ell(\theta, X) = -\mathcal{L}(\theta, X) \tag{3-11}$$

O primeiro ponto teórico é que esse tipo de problema não tem solução analítica fechada e portanto, precisa de ser aproximado numericamente. O problema prático que aparece aqui é que não existe uma forma eficiente de se calcular o gradiente da função de custo. Vale lembrar, que a nossa função de probabilidade tem uma constante de particionamento, que necessita de um

número exponencial de operações para ser calculada, ou seja, é intratável. A fórmula da distribuição conjunta também é intratável, porém, a distribuição condicional pode ser calculada e podemos realizar amostragem nela de forma tratável e eficiente.

O que Hinton mostrou em 2002 [12] foi uma forma de se aproximar o gradiente dessa função de custo, para que ela possa facilmente ser maximizada com um método de gradiente. Para calcular tal aproximação, devemos derivar a probabilidade condicional $P(h|x)$ da probabilidade conjunta.

$$\begin{aligned} P(h|x) &= \frac{P(x,h)}{P(x)} \\ &= \frac{\frac{1}{Z}e^{-E(x,h)}}{\sum_h \frac{1}{Z}e^{-E(x,h)}} \\ &= \frac{e^{-E(x,h)}}{\sum_h e^{-E(x,h)}} \end{aligned} \tag{3-12}$$

Agora vamos substituir a função de energia pela sua definição matricial:

$$\begin{aligned} P(h|x) &= \frac{e^{b^T x + c^T h + x^T W h}}{\sum_h e^{b^T x + c^T h + x^T W h}} \\ &= \frac{e^{b^T x} \cdot e^{c^T h} \cdot e^{x^T W h}}{\sum_h e^{e^{b^T x} \cdot e^{c^T h} \cdot e^{x^T W h}}} \\ &= \frac{e^{c^T h} \cdot e^{x^W h}}{\sum_h e^{e^{c^T h} \cdot e^{x^T W h}}} \end{aligned} \tag{3-13}$$

Nessa derivação conseguimos eliminar a constante Z intratável ganhando uma outra constante $Z\prime$ (o denominador) que poderá ser calculada. Seguimos a derivação para chegar na forma que nos interessa:

$$\begin{aligned} P(h|x) &= \frac{1}{Z\prime} e^{c^T h + x^T W h} \\ &= \frac{1}{Z\prime} exp\{\sum_{j=1}^{n} c_j h_j + \sum_{j=1}^{n} x^T W_j h_j\} \\ &= \frac{1}{Z\prime} \prod_{j=1}^{n} exp\{c_j h_j + x^T W_j h_j\} \end{aligned} \tag{3-14}$$

Aqui chegamos no ponto onde existe uma propriedade específica das RBMs que as tornam tratáveis. Se temos um produto e as probabilidades

das unidades h_j são independentes dado x, podemos interpretar $P(h|x)$ como o produto de entre $P(h_j|x)$ para os valores de j. Agora vamos entender melhor o que é o termo $P(h_j = 1|x)$:

$$\begin{aligned} P(h_j = 1|x) &= \frac{P(h_j = 1, x)}{P(h_j = 0, x) + P(h_j = 1, x)} \\ &= \frac{exp\{c_j + v^T W_{:j}\}}{exp\{0\} + exp\{c_j + v^T W_{:j}\}} \\ &= sigmoid(c_j + v^T W_{:j}) \end{aligned} \tag{3-15}$$

Podemos então redefinir a probabilidade condicional da camada escondida dado o vetor de entrada:

$$P(h|x) = \prod_{j=1}^{n} sigmoid(c_j + v^T W_{:j}) \tag{3-16}$$

E da camada visível dado o vetor na escondida:

$$P(x|h) = \prod_{i=1}^{m} sigmoid(b_i + W_{i:} h) \tag{3-17}$$

O fato de podermos calcular com facilidade as probabilidades individuais de cada unidade e em paralelo que nos permite obter uma aproximação do gradiente com amostragem. Essa aproximação do gradiente é obtida com amostragem de Gibbs, e que na prática tem uma execução muito simples.

Dentro do arcabouço de modelos gráficos probabilísticos a topologia da RBM torna os nós de uma mesma camada, independentes entre si, quando temos os valores da outra camada, justamente porque não existe conexão, ou seja, dependência, entre nós da mesma camada. Na visão podemos dizer que todos os nós da camada visível são a cobertura de Markov de todos os outros nós da camada escondida, individualmente, ou seja, se temos os valores de uma camada, podemos fazer uma etapa de amostragem na outra camada, em paralelo, para todos os nós. Essa propriedade tem grande importância prática pois torna viável o processo de amostragem, que era muito mais custoso do que numa máquina de Boltzman sem restrições.

Quando executamos a amostragem de Gibbs, iniciando com os valores de um exemplo na camada visível, obtemos uma representação binária, daquele vetor de entrada, na camada escondida. Podemos repetir esse processo no sentido contrário e obter uma reconstrução do exemplo original. A aproximação do gradiente é, mais precisamente, a o valor esperado da diferença entre o exemplo original e a reconstrução. Existe um aspecto teórico importante que motiva essa forma de se aproximar o gradiente. Aplicar o método de máxima verossimilhança pode se reduzir a minimizar a divergência de Kullback–Leibler

entre duas distribuições de probabilidade. As distribuições que consideramos são a distribuição dos dados e a distribuição gerada pelo modelo. Em teoria a distribuição gerada pelo modelo deveria ser gerada apenas quando o equilíbrio fosse atingido. Na prática com apenas poucas etapas de amostragem já conseguimos um exemplo gerado pelo modelo que aproxima de um exemplo gerado pelo sistema em equilíbrio. Todo detalhamento teórico sobre o que foi explicado pode ser visto em [12].

As condicionais nos permitem obter amostras da distribuição do modelo para utilizar no método de aprendizado. Essa era a peça faltante para permitir a aplicação de um método de gradiente para otimização do modelo.

A fórmula que expressa a aproximação do gradiente é:

$$\frac{\partial log P(x)}{\partial w_{ij}} = \langle v_i h_j \rangle_{model} - \langle v_i h_j \rangle_{data} \tag{3-18}$$

Tendo em mãos esse gradiente aproximado, a regra de aprendizado torna-se bem simples. Basta aplicar um fator de aprendizagem nessa diferença.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{model} - \langle v_i h_j \rangle_{data}) \tag{3-19}$$

Esse processo de amostragem de Gibbs, exige o calculo da probabilidade de que cada unidade, seja ela da camada visível ou escondida, esteja ativada. No caso de um elemento da camada escondida, temos:

$$P(h_j = 1|x) = \sigma(c_j + \sum_i v_i w_{ij}) \tag{3-20}$$

Temos algo muito semelhante, mas agora, em relação a uma unidade da camada visível:

$$P(v_i = 1|h) = \sigma(b_i + \sum_j h_j w_{ij}) \tag{3-21}$$

Para concluir uma passo da amostragem, confrontamos $P(h_i = 1|x)$ e $P(v_i = 1|h)$, com um valor sorteado da distribuição uniforme. Caso seja maior, a unidade em questão será ativada. Esse procedimento é feito para todos os nós de uma camada e, pelas características da topologia da RBM, pode ser feito em paralelo para todas unidades de uma camada. Na prática, tendo a configuração de entrada, é possível fazer um passo da amostragem em paralelo para todas as outras unidades da unidade escondida. O mesmo vale quando temos uma configuração na camada escondida e podemos, também, realizar um passo da amostragem para todos os nós da camada visível, em paralelo.

O resultado final é um modelo que pode ser treinado com um método de gradiente e com características amigáveis ao paralelismo. Apesar de uma base teórica que envolve vários conceitos e que possui diversas formas de

ser estudada, algoritmicamente, conseguimos treinar uma RBM com um procedimento simples e que pode beneficiar-se do uso de GPUS.

No próximo capítulo, vamos entender como usar uma RBM binária para aprender dados tabulares, isso é, gerar um valor escalar para uma entrada perdida de uma tabela de dados, onde exista, alguma interdependência entre colunas ou linhas dessa tabela.

4
Máquinas de Boltzmann restritas e filtragem colaborativa

O foco desse trabalho é destrinchar como podemos usar uma RBM binária para aprender dados tabulares. Tudo que vimos até agora, considera o caso de uma RBM binária, onde cada exemplo é um vetor binário e a saída é uma reconstrução da entrada. Se faz necessário, então, um artifício para conseguirmos produzir um valor escalar, como no problema de filtragem colaborativa, onde queremos determinar o interesse de um usuário por um produto através desse valor escalar.

A partir de agora, vamos descrever e formalizar o modelo em questão. A entrada do nosso problema é uma matriz $R \in \mathbb{R}(m \times n)$ sendo m número de usuários e n número de produtos. Em muitos casos reais, essa matriz é esparsa, já que a maioria dos usuários não interage com a maior parte dos itens. Como a RBM que estamos analisando trata dados binários, vamos precisar modelar esse valor escalar em termos de unidades binárias, como uma etapa de pré processamento. Vamos considerar, sem perda de generalização, que as preferências são números entre 1 e k. A unidade v_{ik} é 1 se um usuário demonstrou preferência k pelo filme i. Quando a preferência do usuário pelo item é desconhecida, todas as unidades v_{ik} para um item i tem o valor 0. A figura a seguir mostra uma configuração de um usuário que deu nota 3 para o primeiro filme, nenhuma nota pro segundo e 5 pro terceiro filme:

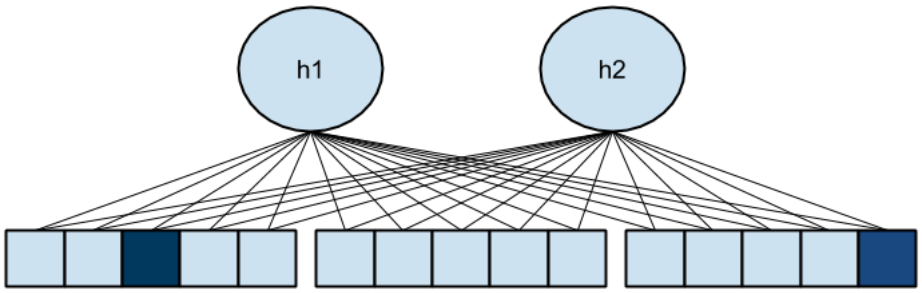

Figura 4.1: Uma RBM que modela valores de 1 a 5 por item

O número de unidades na camada visível da RBM deve ser, então, nk. Tendo uma RBM com nk nós na camada visível, e j nós na camada escondida, precisaremos de $W \in \mathbb{R}(nk \times j)$, a matriz de pesos e 2 vetores para o viés de cada camada, que chamaremos de $b \in \mathbb{R}^{nk}$ e $c \in \mathbb{R}^{j}$. Pelo nosso modelo, cada item é representado por k unidades. Como sempre temos k unidades

para um mesmo item interpretamos esse conjunto como uma unidade *softmax* em relação ao item i. O próximo detalhe do modelo é a alteração da função de ativação das unidades v_{ik}. Ao invés de manter a função de ativação vista anteriormente em 3-21, vamos normalizar a ativação de cada unidade pela soma das k unidades referentes ao item i em questão no mesmo formato que uma função softmax:

$$P(v_i^k = 1|h) = \frac{\exp(b_i^k + \Sigma_j W_{ij}^k h_j)}{\Sigma_{k=1}^K \exp(b_i^k + \Sigma_j W_{ij}^k h_j)} \tag{4-1}$$

O resultado dessa função de ativação é que, dado um filme i, cada unidade v_{ik} representa a probabilidade de um usuário atribuir a preferência k a esse item. Felizmente, essa mudança na regra de ativação das unidades visíveis não interfere no cálculo do gradiente então, o restante permanece igual ao modelo mais comum com RBMs.

No processo de treinamento, a correção dos pesos pode ser feita para cada usuário ou para um bloco usualmente pequeno de usuários. Dessa forma o modelo consegue aprender um pouco sobre cada padrão individual para que seja possível capturar a interdependência entre as preferências dadas para os itens. Nesse modelo quando não temos informação de um usuário a respeito de um item, todas as unidades referentes ao item terão valor 0 e tem que permanecer como 0 mesmo na reconstrução. Esse recurso evita que pesos relativos a itens que não estão nos dados de uma iteração sejam corrigidos. É uma forma de se contornar a enorme esparsidade dos dados.

A figura a seguir mostra em cinza quais unidades e quais arestas (pesos) não devem ser atualizados na etapa de correção dos pesos:

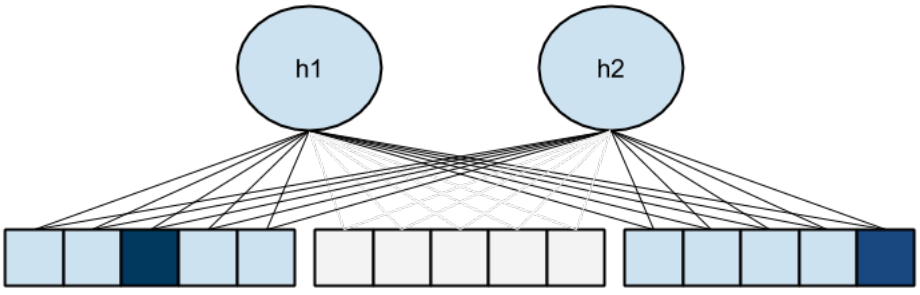

Figura 4.2: Detalhe na regra de atualização dos pesos

Esse detalhe é muito importante e precisa ser implementado de alguma forma. No modelo comum de RBMs, existiriam reconstruções para os nós e arestas em cinza, respectivamente viés e pesos das arestas, porém, eles devem ser ignorados na correção. Podemos descrever esse detalhe formalmente redefinindo a equação de correção para:

$$\Delta w_{ij} = \epsilon(\langle v_i, h_j \rangle_{model} - \langle v_i, h_j \rangle_{data}), v_i > 0 \tag{4-2}$$

$$\Delta w_{ij} = 0, v_i = 0 \tag{4-3}$$

O termo $v_i > 0$ nas equações acima significa que pelo menos uma das k unidades, referentes a um filme i, está com o valor 1. Concretamente, tanto no viés da camada visível quanto na matriz de pesos, todas entradas associadas a itens que não participaram da iteração devem permanecer inalteradas, por isso $\Delta w_{ij} = 0, v_i = 0$. Esse detalhe merece cuidado especial em implementações que utilizam operações matriciais, pois a reconstrução deve ser multiplicada, elemento a elemento, com uma máscara que zera todos os valores relacionados aos itens que não estão sendo referenciados na iteração de aprendizado. Se esse detalhe não for corretamente implementado o modelo não irá convergir e atingir os resultados reportados nesse trabalho. Caso o treinamento seja feito em blocos, a taxa de aprendizado deve ser proporcional ao tamanho do bloco utilizado e uma média pode ser calculada a partir das diversas reconstruções obtidas, para a definição do gradiente de uma iteração. A seguir, vamos ver o algoritmo de treino *online* onde cada usuário gera uma correção.

Algorithm 2 Procedimento de treino

procedure TRAIN(V, epochs)
  $W^{m \times nk} \leftarrow random(m, nk)$
  for 1 .. epochs **do**
    for each example $v \in V$ **do**
      for each hidden unit h_i **do**
        $activation = \sigma(c_j + \sum_i v_i w_{ij})$
        if $activation \geq sampleUniform(0, 1)$ **then**
          $h_j \leftarrow 1$
        else
          $h_j \leftarrow 0$
      for each visble unit v_i^k **do**
        $activation = P(v_i^k = 1|h) = \frac{\exp(b_i^k + \Sigma_j W_{ij}^k h_j)}{\Sigma_{k=1}^K \exp(b_i^k + \Sigma_j W_{ij}^k h_j)}$
        if $activation \geq sampleUniform(0, 1)$ **then**
          $v_i^k \leftarrow 1$
        else
          $v_i^k \leftarrow 0$
      $W \leftarrow W + \epsilon(\langle v_i h_j \rangle_{model} - \langle v_i h_j \rangle_{data})$
      $c_j \leftarrow c_j + \epsilon(\langle h_j \rangle_{model} - \langle h_j \rangle_{data})$
      $b_i \leftarrow b_i + \epsilon(\langle v_i \rangle_{model} - \langle v_i \rangle_{data})$

Esse algoritmo pode ser rodado quantas épocas se quiser e geralmente é parado quando $|E_n - E_{n-1}| < \alpha$ com $\alpha = 0.0001$ ou algum valor próximo.

Após o treinamento, a predição pode ser feita de duas formas. Primeiro, colocamos os dados das preferências do usuário na camada visível. A partir deles, fazemos uma etapa da amostragem de gibbs para determinar uma representação binária na camada escondida. Com a camada escondida, fazemos uma outra etapa, agora da camada escondida para a camada visível. O que obtemos de volta, na camada visível, é a reconstrução das preferências daquele usuário. Com isso, teremos os valores de ativação para as k unidades de um item i qualquer. Podemos simplesmente retornar o valor k com maior ativação ou, calcular o valor esperado:

$$E[v_i] = \sum_k P(v_i^k = 1|h)k \tag{4-4}$$

Outros recursos importantes e que tiveram seu impacto medido nesse trabalho são os meta-parâmetros de momentum e decaimento de pesos. Esses meta-parâmetros são chamados assim porque não fazem parte diretamente do modelo e sim do algoritmo de treinamento. Como vamos otimizar nosso modelo com um método de gradiente, geralmente, adota-se uma versão chamada de gradiente descendente estocástico em pequenos blocos. Esse algoritmo é uma versão especializada do método de gradiente descendente estocástico que é extremamente popular em problemas de aprendizado de máquina. Vamos ver qual a implicação dos meta-parâmetros momentum e decaimento de pesos no nosso modelo final e que foi exatamente o que foi usado nos experimentos desse trabalho.

Ambos afetam a regra de atualização dos pesos. O momentum aplica um coeficiente α no gradiente da iteração anterior na iteração corrente. Formalmente:

$$\Delta w_{ij}^n = \epsilon(\langle v_i, h_j\rangle_{model} - \langle v_i, h_j\rangle_{data}) + \alpha\Delta w_{ij}^{n-1} \tag{4-5}$$

O decaimento de pesos também foi utilizado e influencia no gradiente. Aplicamos um coeficiente λ que irá penalizar pesos grandes e que nos dá a versão final da nossa equação de atualização de pesos:

$$\Delta w_{ij}^n = \epsilon(\langle v_i, h_j\rangle_{model} - \langle v_i, h_j\rangle_{data}) + \alpha\Delta w_{ij}^{n-1} - \lambda w_{ij} \tag{4-6}$$

O modelo matemático definido em 4-6 foi o modelo implementado pelo software entregue junto com esse trabalho. Alguns trabalhos propõem detalhar como implementar o algoritmo descrito especificamente para GPUS como [6] enquanto outros propõem realizar o treinamento de RBM em larga escala usando apenas uma GPU como [24]. A implementação utilizada nos experimentos desse trabalho teve como foco principal uma conexão fácil entre o modelo matemático e o que está no código justamente visando futuras

expansões.

Ao longo desse trabalho as explicações partiam do princípio que estávamos falando de um modelo baseado em usuários. Um último aspecto importante que vale lembrar é que tudo que foi explicado se aplica integralmente se quisermos usar as RBMs como um modelo baseado em itens sem nenhuma alteração adicional.

A seguir, iremos identificar nossos modelos como "U-RBM"e "I-RBM". Na próxima seção estão os resultados atingidos nesse trabalho.

5
Resultados obtidos

A seguir, vamos ver os resultados finais obtidos e alguns outros dados sobre o processo de treinamento e ajuste do modelo. Os experimentos foram feitos no dataset do GroupLens e que será identificado como "MovieLens 100k".

O dataset "MovieLens 100k"possui cem mil notas, de 1 a 5, atribuídas a a filmes, 1682 filmes, 943 usuários e que tem coeficiente de esparsidade 93.7%. Os resultados mais recente com U-RBMs está em [9] com 0.779 de MAE atingido. Para I-RBMs temos 0.775 nesse mesmo conjunto de dados, todos os resultados no mesmo trabalho. Esse trabalho realizou experimentos com o modelo U-RBM e I-RBM. Esses modelos são o ponto de partida para algumas outras variações que não foram estudadas nesse trabalho mas que merecem atenção por terem mostrado bons resultados.

Para fazer uma comparação correta, adotamos a mesma métrica do trabalho citado anteriormente que é o erro médio absoluto (*Mean Absolute Error*) definido como:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |f_i - y_i| \tag{5-1}$$

Estamos chamando f_i a preferência conhecida do conjunto de testes, y_i a predição do modelo e n o tamanho do conjunto de testes.

Em todos experimentos usamos o treinamento com o algoritmo CD-k1, onde a correção de pesos é feita com uma reconstrução depois de apenas 1 etapa de amostragem. Esse algoritmo encontra-se descrito em detalhes nesse trabalho [12] e em pseudo código em [algo: 2]. A etapa de definição da quantidade de nós na camada escondida e de outros parâmetros foi feita com validação cruzada aleatória de 80%/20% com 5 grupos de dados. Os resultados finais obtidos no dataset foram calculados depois de 5 experimentos, também de 80%/20% e a média deles foi reportada. Na primeira etapa, foi feito um experimento com diversos valores para a quantidade de nós na camada escondida: 10, 40, 50 e 100. Eles são o ponto de partida para todos os experimentos a seguir. Cada experimento teve deduração de 100 épocas.

Os melhores resultados (menor erro atingido ao longo do treinamento) para U-RBMs foram obtidos com 10 e 100 nós na camada escondida. Como os valores foram próximos, omitimos os valores de 40 e 50 para facilitar a comparação entre os 2 melhores. O gráfico abaixo, mostra a variação da taxa

de erro por época para cada um dos 2 experimentos que atingiram os melhores resultados:

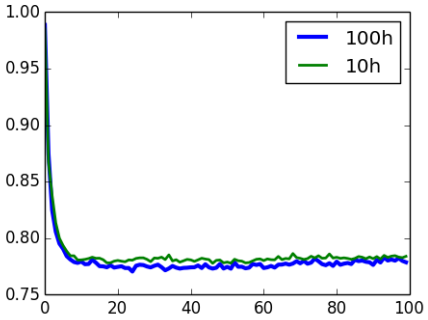

Figura 5.1: Gráfico da variação da taxa de erro para 2 valores possíveis para o números de nós na camada escondida

Os menores MAEs atingidos foram 0.791 para $h = 10$ e 0.790 para $h = 100$. Esse experimento fixou $h = 100$ para o experimento final com U-RBMs. Com esse valor definido seguimos definindo o valor dos outros parâmetros. O primeiro experimento foi feito para determinar o valor da taxa de aprendizado. Como o método de otimização que usamos é um método iterativo, vamos ter que escolher uma taxa de aprendizado que permita com que a convergência aconteça na medida certa.

No experimento para definir a taxa de aprendizagem testamos os valores 0.005, 0.0005, 0.00005:

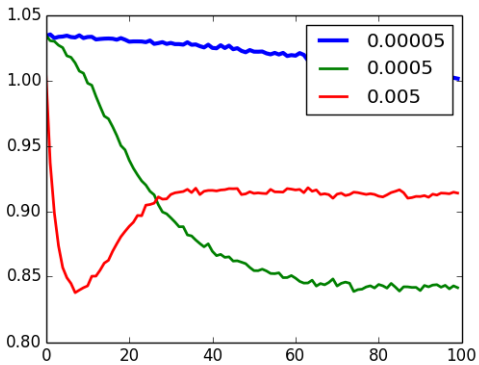

Figura 5.2: Variação do erro com diferentes taxas de aprendizado

Esse experimento mostra que taxas de aprendizado grandes podem gerar modelos piores mesmo depois de aparentemente levarem a bons resultados. Taxas pequenas demais podem tornar a convergência lenta o que também não é desejado. Esse experimento fixou a taxa de aprendizado em 0.0005.

Outros 2 meta-parâmetros que foram utilizados foram momentum e o decaimento de pesos. Como mencionado em [11] o parâmetro de momentum

pode acelerar bastante o aprendizado e quase sempre é usado. Como diversos outros trabalhos com RBMs também usam, fizemos experimentos para entender qual o impacto do meta-parâmetro e como achar um bom valor. A seguir dois gráficos, um que compara um treinamento com e sem momentum abaixo um experimento que compara 3 valores: 0.3, 0.6, 0.9.

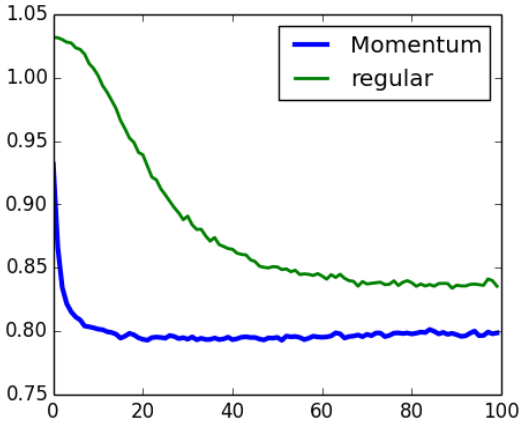

Figura 5.3: Variação do erro com e sem momentum

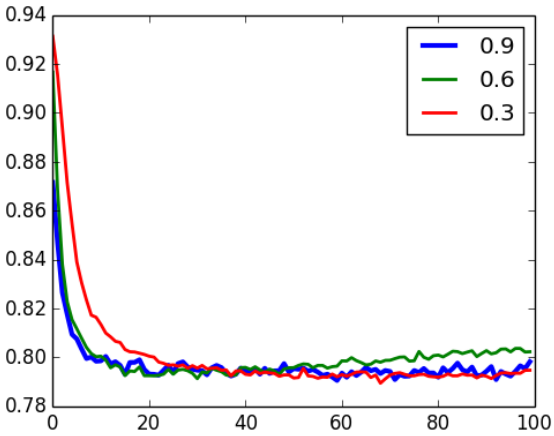

Figura 5.4: Variação do erro para 3 valores de momentum

Primeiro podemos observar que o momentum acelera drasticamente o aprendizado, especialmente no início. Esse gráfico deixa claro que esse parâmetro deve ser usado e que basta achar um bom valor para utilizá-lo. Analisando o gráfico dos diferentes valores de momentum, ainda que os resultados tenham sido razoavelmente próximos o valor que resultou no menor erro ao longo do treinamento foi 0.3. Esse valor é menor que o reportado em [9] que usou o valor 0.6, mas que também reportou não ter buscado pelo melhor valor e no trabalho original de [18] que reportou uma taxa inicial de 0.8 com

aumento para 0.9 no final. A nossa escolha baseia-se apenas em evidência empírica logo seguimos com o valor de 0.3.

O último parâmetro definido foi o decaimento de pesos. Esse parâmetro ajuda a manter os pesos do modelo menores, penalizando pesos grandes. Esse recurso, além de outras vantagens citadas em [11], ajuda a prevenir o sobreajuste de pesos o que sempre é desejável. Primeiro fizemos uma comparação de treino com e sem decaimento de pesos e depois uma comparação entre diferentes valores:

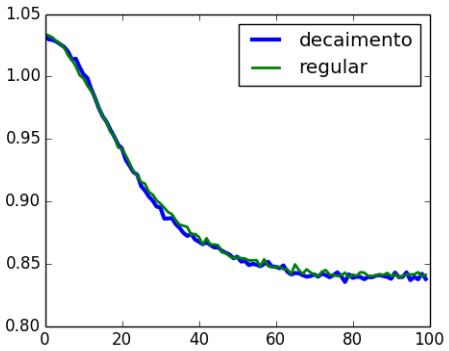

Figura 5.5: Variação do erro com e sem decaimento de pesos

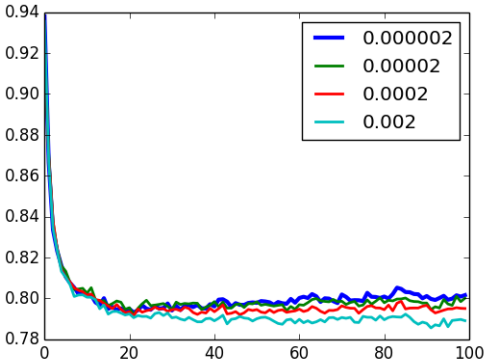

Figura 5.6: Variação do erro com diferentes valores para decaimento de pesos

Não ficou tão claro o impacto de treinar o modelo com e sem decaimento de pesos mas o menor valor de erro foi atingido com decaimento. No experimento para determinar o valor do decaimento podemos observar que o valor de 0.002 foi consistentemente melhor que os outros e portanto foi o valor escolhido.

Finalmente podemos rodar o experimento final com os parâmetros: número de nós na camada escondida 100, taxa de aprendizagem 0.0005, momentum 0.3, e decaimento de pesos 0.002. O resultado final obtido foi 0.773.

Agora vamos ver o mesmo processo para um modelo I-RBM. Podemos observar que o mesmo conjunto de dados, quando mudada a perspectiva, pode gerar um comportamento relativamente diferente para o modelo. Essa mudança tem relação com a natureza dos dados tabulares pois a ordem linha/coluna pode ser trocada mas os dados das células mantém-se iguais. No modelo U-RBM as linhas são os usuários enquanto no modelo I-RBM as linhas são os itens.

O primeiro parâmetro a ser definido foi a quantidade de nós na camada escondida Definimos a quantidade de nós na camada escondida realizando experimentos com 10, 40, 50, e 100 nós:

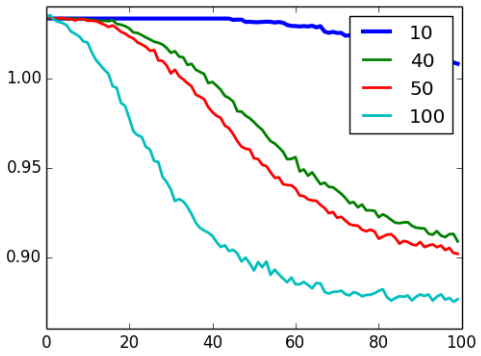

Figura 5.7: Gráfico da variação da taxa de erro para 2 valores possíveis para o números de nós na camada escondida

O que pudemos observar é que no modelo I-RBM a quantidade de nós na camada escondida afeta a velocidade do aprendizado: quanto mais nós mais rápido se aprende os padrões inerente aos dados. Vale lembrar, que no conjunto de dados "MoviesLens 100k"temos 943 usuárioe e aproximadamente 1600 filmes, ou seja, quando o algoritmo aprende os padrões por usuário, o conjunto total é composto de 943 exemplos enquanto aprender os padrões de notas dos filmes, o número de padrões para ser aprendido é quase o dobro, o que pode explicar uma dificuldade maior em se aprender os padrões dos filmes.

Novamente o melhor resultado foi atingido com 100 nós na camada escondida. Esse número de nós é reportado no trabalho original [18] e em um dos experimentos do trabalho mais recente [9]. Em termos práticos, em alguns casos é possível se obter um resultado muito próximo com muito menos o que também acaba sendo uma vantagem prática.

A seguir, fizemos o próximo experimento para determinar a taxa de aprendizado. Vale lembrar que se pensarmos em blocos(batches) de 10 exemplos, no modelo U-RBM uma época gera 90 correções enquanto no modelo

I-RBM seriam 160 correções e esse próximo gráfico vai poder mostrar se existe impacto comparando os valores: 0.005, 0.0005, 0.00005:

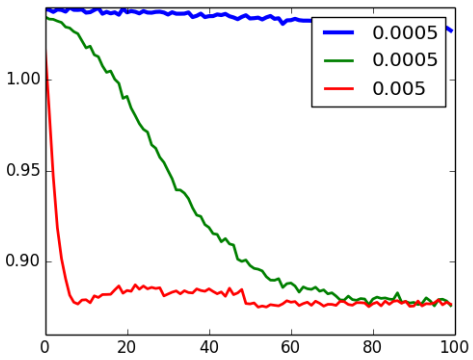

Figura 5.8: Variação do erro com diferentes taxas de aprendizado

Diferente do experimento com U-RBM a taxa de aprendizado de 0.005, a maior dentre as 3, levou ao melhor resultado. Ela também não teve nenhum momento de grande mudança de faixa de resultado e manteve o aprendizado estável até o final. Por isso continuamos os experimentos com a taxa de aprendizado como 0.005. Podemos interpretar que no modelo I-RBM 100 épocas não foi tempo suficiente para fazer o modelo divergir, possivelmente porque há mais padrões a se aprender.

Agora vamos a escolha dos meta-parâmetros momentum e decaimento de pesos. Realizamos um processo idêntico ao do experimento com U-RBMs. Primeiro, um gráfico mostra a variação do erro com e sem momentum definido:

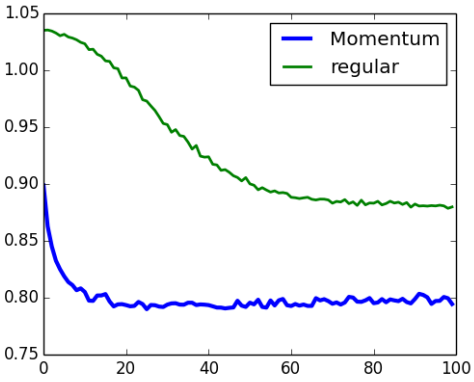

Figura 5.9: Variação do erro com e sem momentum

Novamente observamos que o meta-parâmetro de momentum é essencial pro aprendizado atingir o resultado desejado e numa velocidade muito maior. Seguimos agora com a escolha do melhor valor do meta parâmetro momentum. O gráfico a seguir compara a variação do erro entre os valores 0.3, 0.6 e

0.9 valores estes também retirado dos experimentos feitos nos trabalhos de referência:

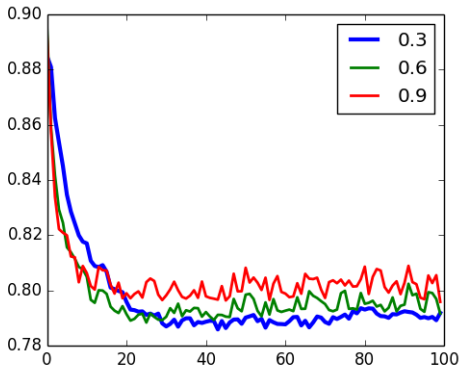

Figura 5.10: Variação do erro momentum 0.3, 0.6 e 0.9

O melhor valor foi o valor de 0.3 que é um valor menor que o reportado em [9] que usou o valor 0.6 mas reportou não ter buscado pelo melhor valor e no trabalho original de [18] que reportou uma taxa inicial de 0.8 com aumento para 0.9 no final.

O útilmo meta-parâmetro a ser definido é o decaimento de pesos. Primeiro veremos uma comparação onde esse meta-parâmetro é utilizado:

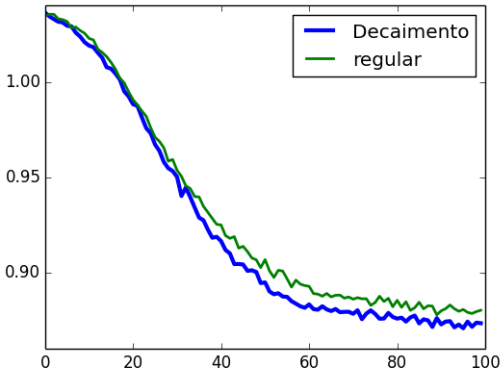

Figura 5.11: Variação do erro com e sem decaimento de pesos

Nesse experimento ficou mais claro que a regularização do decaimento de pesos foi mais benéfica pois alcançou melhor resultado de forma consistente. Assim como nas duas principais referências o meta-parâmetro de decaimento de pesos foi utilizado. Ainda que na seleção do modelo para U-RBMs o decaimento de peso tenha tido pouco impacto no modelo I-RBM o decaimento melhorou o resultado final e por isso iremos repetir o processo de seleção do parâmetro novamente para o modelo I-RBM. No gráfico a seguir veremos uma comparação

entre diversos valores que também foram extraídos dos trabalhos usados como referência.

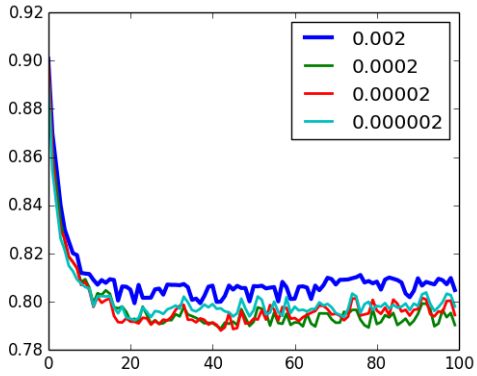

Figura 5.12: Variação do erro com diferentes valores para decaimento de pesos

Quando comparamos os valores, 2 valores tem os melhores resultados: 0.00002 e 0.0002. Quem chegou no menor erro foi o valor 0.00002 e por isso foi utilizado no experimento final. Agora que passamos cuidadosamente pelas etapas de seleção de modelo e validação cruzada podemos pegar os parâmetros e meta-parâmetros escolhidos e rodar o experimento que nós dará o resultado final reproduzido por esse trabalho.

No rakning de melhores resultados, que está na tabela 5.1 na página 44, recentes para esse dataset o resultado desse trabalho figura entre os 8 melhores reportados. Considero um bom resultado visto que alguns dos outros métodos são mais especializados e outros são combinações de métodos. O modelo de filtragem colaborativa com RBM é competitivo por que tem um bom resultado se avaliarmos o erro atingido e a facilidade de se computar. Além disso, quando bem ajustados, conseguem atingir o melhor resultado em poucas iterações e podem ser computados de forma rápida. Além disso, é possível conseguir o segundo melhor resultado com o custo computacional de 2 modelos independentes [9] reproduzindo o modelo UI-RBM, algo que pode ser estudado em um futuro próximo.

Tabela 5.1: Ranking de métodos

Modelo	MAE
SVD PCA [23]	0.793
H-NLPCA [23]	0.784
Esse trabalho U-RBM	**0.776**
Esse trabalho I-RBM	**0.773**
Item-based CF [20]	0.726
SVD [19]	0.733
PCA + Kmeans [15]	0.726
PCA + RRC [15]	0.700
UI-RBM [9]	0.690
Latent CF [16]	0.685

6
Conclusões

Nesse trabalho analisamos mais um caso de aplicação de RBMs para problemas de dados tabulares. Os resultados obtidos são muito semelhantes aos melhores resultados publicados com esse modelo e também são competitivos com outros métodos populares para esse tipo de problema. Um dos objetivos desse trabalho era criar um material que cobrisse a parte teórica central do modelo matemático e detalhes práticos, sobre o uso das RBMs nesse tipo de problema. Também, foi produzida uma implementação compatível com GPUs, que ficará disponível para que outros alunos deem continuidade ou utilizem-a como ferramenta em seus trabalhos. Vale reforçar que todos os gráficos apresentados mostram uma relação entre o erro mensurado ao longo da execução das épocas de treinamento.

Apesar de termos explorados os modelos de RBM baseada em usuários e o modelo baseado em itens, existe um modelo híbrido que trabalha internamente com cada um desses modelos e com uma regra de correção de pesos diferente para considerar 2 predições simultaneamente e tirar uma espécie de média. Esse tipo de modelo poderá ser facilmente implementado usando como base o projeto entregue junto com a dissertação e permitirá experimentos que podem alcançar resultados ainda melhores.

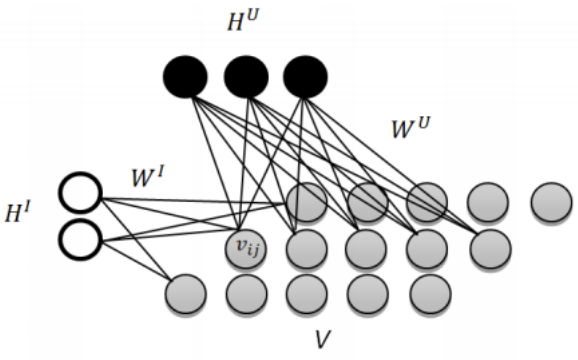

Figura 6.1: UI-RBM representação gráfica do modelo

Como representado pela imagem a ativação v_{ij} sofre influência de 2 RBMs simultaneamente: uma que é baseada em usuários e outra que é baseada em itens. A nova função de ativação passa a ser definida como:

$$v_{ij} = \frac{1}{2}\Big[a_i^U + \sum_{p=1}^{F^U} w_{ip}^U h_{ip}^U + a_j^I + \sum_{q=1}^{F^I} w_{jq}^I h_{jq}^I\Big] \tag{6-1}$$

Esse modelo, chamado de UI-RBM, além de obter um erro ainda menor tem um custo computacional de duas RBMs o que pode ser visto como uma grande vantagem computacional frente a melhoria obtida.

As RBMs também possuem outras variações como RBMs condicionais [18], que podem melhorar os resultados obtidos e não foram implementadas por falta de tempo. Experimentos com combinações com o resultado de outros algoritmos também podem ajudar a se alcançar erros menores, já que em teoria cada método pode produzir erros diferentes. RBMs também podem ser adaptadas para recomendações sensíveis ao contexto se olharmos para as RBMs temporais, que modelam uma sequencia de padrões no tempo, o que pode refletir em recomendações mais precisas.

Outro aspecto marcante é que RBMs são modelos para aprendizado não supervisionado e podem contribuir em diversos domínios. RBMs podem ser empilhadas, em uma estrutura profunda chamada Redes de confiança profunda [13] que tem demonstrado bons resultados também. Por todos esses motivos esse trabalho buscou explicar o modelo e sua aplicação na tarefa proposta e registrar o resultado validado empiricamente dele.

RBMs também podem ser vistas como autoencoders estocásticos pois tem como objetivo reconstruir a entrada através de uma representação interna expressa na camada escondida. Na fase atual de pesquisa em aprendizado de máquinas os autoencoders são extremamente relevantes pois podem compactar uma entrada grande demais como extrair atributos automaticamente. Nos próximos anos continuaremos vendo aplicações de RBM tanto em problemas como o desse trabalho como possivelmente em outros. Enquanto a indústria adota o que é economicamente mais viável a academia trabalha no que talvez pareça mais robusto. Na opinião do autor existe espaço tanto para entender melhor a viabilidade econômica desse modelo quanto estudo, desenvolvimento teórico e aplicações bem sucedidas em outros domínios.

A conclusão final nos da segurança de que a implementação criada para esse trabalho está correta pois reproduziu com grande precisão os resultados reportados para o modelo de RBMs no mesmo dataset. Sendo assim, o código entregue fica como legado para que no futuro outros alunos possam reproduzir o resultado obtido e até mesmo avançar com outros modelos que podem ter como base o modelo apresentado nesse trabalho.

7
Referências bibliográficas

[1] David H. Ackley, Geoffrey E. Hinton e Terrence J. Sejnowski. "Connectionist Models and Their Implications: Readings from Cognitive Science". Em: ed. por David Waltz e Jerome A. Feldman. Norwood, NJ, USA: Ablex Publishing Corp., 1988. Cap. A Learning Algorithm for Boltzmann Machines, pp. 285–307. ISBN: 0-89391-456-8. URL: http://dl.acm.org/citation.cfm?id=54455.54465.

[2] Robert B. Allen. "User Models: Theory, Method, and Practice". Em: *Int. J. Man-Mach. Stud.* 32.5 (maio de 1990), pp. 511–543. ISSN: 0020-7373. DOI: 10.1016/S0020-7373(05)80032-X. URL: http://dx.doi.org/10.1016/S0020-7373(05)80032-X.

[3] Marko Balabanović e Yoav Shoham. "Fab: Content-based, Collaborative Recommendation". Em: *Commun. ACM* 40.3 (mar. de 1997), pp. 66–72. ISSN: 0001-0782. DOI: 10.1145/245108.245124. URL: http://doi.acm.org/10.1145/245108.245124.

[4] Robert M. Bell, Yehuda Koren e Chris Volinsky. *The BellKor solution to the Netflix Prize*.

[5] James Bergstra et al. "Theano: a CPU and GPU Math Expression Compiler". Em: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX, jun. de 2010.

[6] Xianggao Cai et al. "GPU-accelerated Restricted Boltzmann Machine for Collaborative Filtering". Em: *Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*. ICA3PP'12. Fukuoka, Japan: Springer-Verlag, 2012, pp. 303–316. ISBN: 978-3-642-33077-3. DOI: 10.1007/978-3-642-33078-0_22. URL: http://dx.doi.org/10.1007/978-3-642-33078-0_22.

[7] A. Coates, H. Lee e A.Y. Ng. "An analysis of single-layer networks in unsupervised feature learning". Em: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Geoffrey Gordon, David Dunson e Miroslav Dudík. Vol. 15. JMLR Workshop and Conference Proceedings. JMLR WCP, 2011, pp. 215–223. URL: http://jmlr.csail.mit.edu/proceedings/papers/v15/coates11a.html.

[8] Rainer Gemulla et al. "Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent". Em: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: ACM, 2011, pp. 69–77. ISBN: 978-1-4503-0813-7. DOI: 10.1145/2020408.2020426. URL: http://doi.acm.org/10.1145/2020408.2020426.

[9] Kostadin Georgiev e Preslav Nakov. "A non-IID Framework for Collaborative Filtering with Restricted Boltzmann Machines". Em: *Proceedings of the 30th International Conference on Machine Learning, Cycle 3*. Vol. 28. JMLR Proceedings. JMLR.org, 2013, pp. 1148–1156.

[10] Jonathan L. Herlocker et al. "An Algorithmic Framework for Performing Collaborative Filtering". Em: *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: ACM, 1999, pp. 230–237. ISBN: 1-58113-096-1. DOI: 10.1145/312624.312682.

[11] Geoffrey E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". Em: *Neural Networks: Tricks of the Trade - Second Edition*. 2012, pp. 599–619. DOI: 10.1007/978-3-642-35289-8_32. URL: http://dx.doi.org/10.1007/978-3-642-35289-8_32.

[12] Geoffrey E. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". Em: *Neural Comput.* 14.8 (ago. de 2002), pp. 1771–1800. ISSN: 0899-7667. DOI: 10.1162/089976602760128018. URL: http://dx.doi.org/10.1162/089976602760128018.

[13] Geoffrey E. Hinton, Simon Osindero e Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". Em: *Neural Comput.* 18.7 (jul. de 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. URL: http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[14] Yifan Hu, Yehuda Koren e Chris Volinsky. "Collaborative filtering for implicit feedback datasets". Em: *In IEEE International Conference on Data Mining (ICDM 2008*. 2008, pp. 263–272.

[15] Dohyun Kim e Bong-Jin Yum. "Collaborative filtering based on iterative principal component analysis." Em: *Expert Syst. Appl.* 28.4 (4 de jul. de 2006), pp. 823–830. URL: http://dblp.uni-trier.de/db/journals/eswa/eswa28.html#KimY05.

[16] Helge Langseth e Thomas Dyhre Nielsen. "A Latent Model for Collaborative Filtering". Em: *Int. J. Approx. Reasoning* 53.4 (jun. de 2012), pp. 447–466. ISSN: 0888-613X. DOI: 10.1016/j.ijar.2011.11.002. URL: http://dx.doi.org/10.1016/j.ijar.2011.11.002.

[17] Hugo Larochelle e Yoshua Bengio. "Classification Using Discriminative Restricted Boltzmann Machines". Em: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 536–543. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390224. URL: http://doi.acm.org/10.1145/1390156.1390224.

[18] Ruslan Salakhutdinov, Andriy Mnih e Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering". Em: *In Machine Learning, Proceedings of the Twenty-fourth International Conference (ICML 2004). ACM*. AAAI Press, 2007, pp. 791–798.

[19] Badrul Sarwar et al. "Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems". Em: *Proceedings of the 5th International Conference in Computers and Information Technology*. 2002. URL: citeseer.ist.psu.edu/663385.html.

[20] Badrul Sarwar et al. "Item-based Collaborative Filtering Recommendation Algorithms". Em: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: ACM, 2001, pp. 285–295. ISBN: 1-58113-348-0. DOI: 10.1145/371920.372071. URL: http://doi.acm.org/10.1145/371920.372071.

[21] P. Smolensky. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". Em: ed. por David E. Rumelhart, James L. McClelland e CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Cap. Information Processing in Dynamical Systems: Foundations of Harmony Theory, pp. 194–281. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104290.

[22] Wei Tan, Liangliang Cao e Liana L. Fong. "Faster and Cheaper: Parallelizing Large-Scale Matrix Factorization on GPUs". Em: *CoRR* abs/1603.03820 (2016). URL: http://arxiv.org/abs/1603.03820.

[23] Manolis G. Vozalis, Angelos I. Markos e Konstantinos G. Margaritis. "Collaborative Filtering through SVD-Based and Hierarchical Nonlinear PCA." Em: *ICANN (1)*. Ed. por Konstantinos I. Diamantaras, Wlodek

Duch e Lazaros S. Iliadis. Vol. 6352. Lecture Notes in Computer Science. Springer, 2010, pp. 395–400. ISBN: 978-3-642-15818-6. URL: http://dblp.uni-trier.de/db/conf/icann/icann2010-1.html#VozalisMM10.

[24] Yun Zhu, Yanqing Zhang e Yi Pan. "Large-scale restricted boltzmann machines on single GPU." Em: *BigData Conference*. Ed. por Xiaohua Hu et al. IEEE, 2013, pp. 169–174. ISBN: 978-1-4799-1292-6. URL: http://dblp.uni-trier.de/db/conf/bigdataconf/bigdataconf2013.html#ZhuZP13.

A
Apêndice 1

Junto com esse trabalho, foi desenvolvida uma implementação do modelo RBM para o problema da filtragem colaborativa. Essa implementação foi desenvolvida para melhor desempenho em computadores com uma GPU disponível, ou seja, computadores com uma placa gráfica funcional. Tal implementação foi desenvolvida usando a linguagem Python e um framework chamado Theano [5] que faz as operações matemáticas rodarem tanto em GPUs quando em computadores com sem elas, sempre com melhor desempenho quando elas estiverem presentes. Atualmente implementações para placas gráficas tem conquistado um espaço cada vez maior tanto na indústria quando na academia pois aceleram e muitas vezes viabilizam determinadas pesquisas. Um trabalho recente [22] apresenta uma implementação do método de mínimos quadrados alternados, que foi mencionado nesse trabalho, implementado em CUDA, uma linguagem especial para placas gráficas e com um desempenho impressionante que o torna economicamente interessante frente a outras alternativas. A implementação desse trabalho nos mostra, mais uma vez, que existem milhares de decisões a serem tomadas quando transportamos um modelo matemático para um programa de computador. Decisões relativas a em que tipo de memória guardar os dados ou sobre quando sincronizar threads tornam ainda mais complexa a tarefa de rodar um algoritmo de otimização matemática de forma eficiente.

Na prática o código para placas gráficas implica em apenas uma restrição que é o tamanho máximo da memória da placa gráfica. Isso se deve ao fato de que apenas o que está na GPU poderá ser processado pelo algoritmo e portanto quando o conjunto de dados é maior que a memória da GPU ele deve ser quebrado em batches que caibam na própria memória da GPU. Outro ponto importante é evitar loops e tentar implementar o máximo de coisas em operações matemáticas como as operações da Álgebra Linear que são base das operações matemáticas realizadas nos modelos de redes neurais. Por ter sido usado o framework Theano [5], algumas complexidades foram abstraídas pela API que esse framework nos oferece mas o autor desse trabalho acredita que uma implementação desse memo modelo diretamente em CUDA poderia ser muito mais eficiente.

Outro aspecto pouco reforçado sobre os modelos RBMs é que eles podem ser empilhados criando uma arquitetura "deep". Os modelos "deep", hoje,

recebem uma enorme atenção pois são fácilmente mapeados para programação com GPUS e se beneficiam diretamente da maior distribuição desse tipo de dispositivo. Atualmente, milhares de telefones móveis já dispõem de placas gráficas dedicadas em seus hardwares. De uma forma mais global, muitas das novas implementações das ferramentas para criação de modelos "Deep"serão em cima de bibliotecas que já suportam GPUs como o projeto TensorFlow do google, o framework CNTK da Microsoft entre alguns outros. O avanço dessas ferramentas oferece mais poder de processamento a pesquisadores e engenheiros além de facilidades como rodar modelos a partir de uma descrição textual estruturada da topologia da rede.

Visto pela ótica da indústria e aplicação prática, paraticamente qualquer modelo otimizado para GPUS tem grandes chances de ter uma excelente performance, desde que usados da maneira correta e com boas práticas. Além disso o preço desse tipo de placa é razoavelmente acessível tanto para usuários normais, como o autor deste texto, como empresas que compram para uso em produção. Todo esse contexto favorece tanto a pesquisa acadêmica quando a aplicação na indústria. O que torna essa área extremamente promissora e com inúmeros desafios.

Do ponto de vista da pesquisa acadêmica o código entregue junto com o trabalho escrito pode ser utilizado para implementação de outros modelos como as RBMs temporais ou até mesmo as UI-RBMs que tem um resultado superior ao apresentado nesse trabalho. O código pode ser obtido em: https://github.com/felipecruz/CFRBM e foi liberado sob a licença MIT o que permite alteração e distribuição posterior. O conjunto da dissertação mais o código entregue cumprem a missão de criar um trabalho em portugês brasileiro sobre os modelos das máquinas de boltzmann restritas e de entregar para futuros alunos e profissionais um código validado por um dataset reconhecido no meio acadêmico.