Funk Svd - Research Challenge #1

Marcelo Sartori Locatelli
UFMG
Belo Horizonte, Brasil
locatellimarcelo@dcc.ufmg.br

ABSTRACT

KEYWORDS

recommender systems, matrix factorization

ACM Reference Format:
Marcelo Sartori Locatelli. 2018. Funk Svd - Research Challenge #1 . In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUÇÃO

Sistemas de recomendação, em geral, são sistemas usados para a previsão do que um dado usuário gostaria de consumir. Em sua forma mais simples, esses sistemas levam em conta algumas estatísticas básicas dos dados, como por exemplo a média das notas de um dado item ou popularidade dos itens. Porém, ao longo dos anos foram desenvolvidas técnicas mais avançadas para a construção de sistemas de recomendação. Uma dessas técnicas é a fatoração de matrizes.

A fatoração de matrizes consiste em se aprender a reconstruir uma dada matriz que contém aquilo que se deseja prever (no caso, ratings), utilizando para isso fatores latentes dos dados. Nesse trabalho, especificamente, será discutida uma implementação modificada do método de fatoração de matrizes funk-SVD, apresentado em [2], visando obter uma previsão de boa qualidade que minimize o erro(RMSE) em relação aos dados alvo.

2 DATASET

A base de dados utilizada para o treino da base é composta por 403.214 3-tuplas da forma <userid,itemid,rating> que representam dados históricos de avaliações de itens por usuários. As notas variam de 1-5.

Além disso, os dados a serem usados como alvos para o teste da base são 49.630 tuplas da forma <userid,itemid>, sendo que o rating é ocultado.

Finalmente, para o fim de se conseguir testar os parâmetros ótimos, a base de treino foi dividida em um split 70% treino e 30% teste.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

3 IMPLEMENTAÇÃO

Nessa seção serão discutidos detalhes da implementação do algoritmo.

3.1 Algoritmo

O algoritmo implementado, como discutido, foi o funk-svd modificado, visando conseguir melhores resultados de RMSE. A formulação básica do algoritmo funk-svd é tal que a matriz de ratings R (n_usuários x n_items) é fatorada em duas matrizes P(n_usuarios x k) e Q(n_items x k), onde k representa o número de fatores latentes. Como as matrizes P e Q não são inicialmente conhecidas, elas são inicializadas aleatoriamente e aprendidas por meio de um algoritmo de gradiente descendente:

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$P[u] = P[u] - \alpha(e_{ui} * Q[i] - \lambda * P[u])$$

$$Q[i] = Q[i] - \alpha(e_{ui} * P[u] - \lambda * Q[i])$$

Onde α é a taxa de aprendizado, λ a regularização e $\hat{r}_{ui}$ a previsão de rating pelo usuário u para um item i. Seguindo esse algoritmo, consegue-se otimizar P e Q ao longo das iterações, chegando a resultados satisfatórios. Porém, resta discutir como é feita essa previsão $\hat{r}_{ui}$. Na realidade, ela pode ser modificada de várias maneiras visando obter melhores resultados, sendo que em sua forma mais simples, ela é dada por $\hat{R} = PQ^T$, onde $\hat{R}$ é a matriz reconstruída com todos os ratings faltantes preenchidos.

Porém, pelos testes realizados(que serão discutidos na próxima seção), a qualidade de previsão do algoritmo com apenas essas características foi relativamente baixa, o que foi um dos motivadores para a adição de regularização, inspirada pelo livro [1]. Porém, a regularização por si só não melhorou o suficiente os resultados, então foi testado o uso de biases, como visto no livro [4]. Adicionar esses vieses de usuário e item mudaram não somente à previsão, como também o processo do gradiente descendente, que passou a ser:

$$\hat{r}_{ui} = \mu + bu[u] + bi[i] + \langle P[u], Q[i] \rangle$$

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$P[u] = P[u] - \alpha(e_{ui} * Q[i] - \lambda * P[u])$$

$$Q[i] = Q[i] - \alpha(e_{ui} * P[u] - \lambda * Q[i])$$

$$bu[u] = bu[u] - \alpha(e_{ui} - \lambda * bu[u])$$

$$bi[i] = bi[i] - \alpha(e_{ui} - \lambda * bi[i])$$

Onde bu e bi são os biases de usuário e item, respectivamente e μ é a media global das notas. Essa implementação obteve resultados melhores, que foram aqueles submetidos para a competição.

Algorithm	Test Error	Kaggle Error
funk-svd	0.72	1.35
funk-svd w/ regularization	0.69	1.29
funk-svd biased	0.67	1.24
NMF	0.75	1.52

Table 1: Resultados de cada algoritmo na base de teste e no kaggle

3.2 Estruturas de dados

O algoritmo inteiro foi modelado como uma classe SVD, cujos atributos são os hiper-parametros α, λ, k e número de iterações, além das matrizes P e Q^T, os vetores bu e bi e a média global μ. As funções relevantes para uso são fit(train) que faz o treino da base, predictTargets(targets) que faz a previsão dos ratings dos pares (usuario,item) dados e evaluate(test), que avalia o erro(RMSE) das previsões em relação à dados de teste.

P e Q^T foram modelados como arrays da biblioteca numpy, de dimensões (n_usuarios x k) e (k x n_itens) respectivamente. Inicialmente esses arrays foram iniciados com os valores sugeridos pelo professor [5] ($\frac{5*rand()}{\sqrt{5}}$), o que levou a resultados ruins. Isso foi substituído pela inicialização em valores aleatórios entre 0 e 1 posteriormente, o que levou a resultados melhores. Por fim, buscando melhorar o algoritmo e inspirado pela documentação de bibliotecas que implementam o algoritmo[1], a inicialização dessas matrizes foi feita de acordo com uma distribuição normal de média 0 e desvio 0.1. bu e pu foram simplesmente inicializados como arrays de zeros de tamanho n_usuários e n_itens respectivamente.

3.3 Complexidade

Nota-se que em cada iteração do gradiente descendente(l) são avaliadas todas as ratings da base de treino(s). Além disso, avaliar uma rating da base de treino é O(k) (produto interno de P[u] e Q[i]), atualizar P[u] e Q[i] também é O(k) (soma de vetores de tamanho k), por fim, atualizar bu[u] e bi[i] é O(1). Assim, tem-se que são feitas l*s iterações de operações que são no máximo O(k), assim, o treino do algoritmo tem complexidade total O(lsk).

Por outro lado, cada previsão é O(k), como visto pela formula de $\hat{r}_{ui}$, porém, queremos fazer s previsões. Assim, a complexidade de fazer todas previsões pedidas é O(sk). Por fim, completar a matriz R é dominado pela complexidade do produto PQ^T que é O(n_users*k*n_itens).

Quanto à complexidade de espaço, ela é dominada pelo espaço gasto por P e Q e, assim ela é O(n_users*k + n_itens*k).

4 RESULTADOS

Por meio dos testes, estabeleceu-se que com 30 iterações, α = 0.1, λ = 0.02 e k=100, com o algoritmo proposto (funk-svd biased), o erro era minimizado. A tabela 1 mostra o erro de alguns dos modelos testados.

[1]https://surprise.readthedocs.io/en/stable/matrix_factorization.html

5 DISCUSSÃO

É interessante de se notar como o uso da regularização por si só já foi a diferença entre o funk-svd ser superado pelo item average (para esse caso) e ele ser melhor que esse benchmark. Apesar disso, o fator de regularização ótimo era bem pequeno ao se adicionar os biases (0.02 com biases e 0.1 sem biases), isso talvez se deva a um maior número de parâmetros a ser otimizado, o que levou a menos overfitting com menor necessidade de regularização. Outro possível motivo é que no modelo com biases, os valores de P e Q são naturalmente menores (pois a previsão depende também dos biases e média), o que reduz a necessidade da regularização. De fato, o modelo de biases mesmo com λ = 0 já apresentou resultados melhores que todos os demais (1.26 de erro no kaggle).

Outro fator interessante de se notar é como o valor ótimo de k varia para os diferentes modelos, sendo que o modelo com biases funcionava bem com k's maiores, enquanto o com regularização funcionava melhor com k's médios(15-20) e o sem nada funcionava melhor com k's pequenos(5-10). Isso pode sugerir que o modelo com biases é melhor em considerar recomendações não triviais.

Adicionalmente, nota-se que o alpha para todos os modelos foi relativamente elevado, o que pode sugerir que (caso não houvessem restrições de tempo), deveria ser aumentado o número de épocas.

Além disso, nota-se que o modelo NMF(non-negative matrix factorization) baseado em [3] provavelmente teve algum erro de implementação, visto que seu erro oscilava entre alto e baixo dependendo da iteração.

Por fim, destaca-se que foi possível obter resultados satisfatórios na implementação do sistema de recomendação proposto, conseguindo superar os benchmarks propostos sem superar o limite de tempo.

REFERENCES

[1] Charu C Aggarwal et al. 2016. *Recommender systems*. Vol. 1. Springer.
[2] Simon Funk. 2006. Netflix update: Try this at home.
[3] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.
[4] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.
[5] Rodrygo L. T. Santos. 2022. Slides de sistemas de recomendação. Accesso em: 22-09-2022.