Implementação do Algoritmo SlopeOne em PySpark

Guilherme Begotti

São Paulo, Brasil

Abstract

Este breve artigo tem como objetivo introduzir um algoritmo de recomendação baseado em filtragem colaborativa, Slope One; que consiste, basicamente, na recomendação de itens fundamentada nas avaliações do conjunto de usuários aos itens analisadados. Além disso, será mostrado uma implementação deste algoritmo em PySpark e a discussão dos resultados obtidos.

Keywords: Algoritmos, Aprendizado de Máquina, Sistemas de Recomendação, Filtragem Colaborativa, Programação Funcional

1. Introdução

Este artigo, é um breve relatório sobre o trabalho realizado na disciplina de Inteligência na Web e Big Data que tem como requisitos aimplementação distribuída de um algoritmo em Haskell ou Spark que envolva dados ou seja escalável. Para isso, foi escolhido um algoritmo de recomendação por filtragem colaborativa, pois esse tipo de algoritmo apresenta resultados positivos no desenvolvimento prático [1] e requer grande quantidade de informações sobre o usuário e os itens que serão recomendados.

2. Desenvolvimento e Discussão

Brevemente, sistemas de recomendação pretendem, de forma eficiente e matemática, computar predições e através delas gerar recomendações de produtos a usuráios. O algoritmo Slope One, usado neste trabalho, se baseia em filtragem colaborativa de itens, criando recomendações através do cálculo de predições a partir da comparação entre avaliações de usuários ao itens das base de dados.

No artigo, Lemire et al. [2] descrevem três esquemas de algoritmos que almejam ser fáceis de implementar, que conseguissem receber novas avaliações

de itens e preservassem seus cálculos para processamento contínuo, fossem eficientes, também fizessem recomendações válidas e concretas à usuários pouco ativos em avaliações, e que tivessem acurácia significante ao longo da vida do sistema.

Antes mesmo de descrever a álgebra do algoritmo e seu funcionamento, deve-se denotar os testes feitos pelos autores [2], que mostraram boa acurácia dos algoritmos desenvolvidos por eles em comparação com outros esquemas usados comumente pela comunidade [2] e por isso foram escolhidos para implementação.

Isto posto, tem-se para a base dos cálculos envolvidos, nos três algoritmos, uma matriz com todas as notas, de todos os usuários para os produto. De tal forma que cada linha da matriz corresponde às notas de um usuário à uma quantidade itens, ou seja, o elemento da matriz n(i,j) contém a nota do usuário i para o item j. Para os casos omissos, onde o usário i não atribuiu notas a um item j, o elemento da matriz n(i,j) receberá um valor igual a 0.

A partir desta matriz podemos obter relações entre os dados, onde uma coluna da matriz mostra as notas dos diferentes i usuários para um único item j, e a linha define os diferentes j itens avaliados por um único usuário i. Assim, o algoritmo pretende gerar uma função matemática linear (como na equação 1, onde N(i,j) é a nota do usuário i a um determinado item j e P(i,j) é predição do usuário i para o item j) e predizer qual seria a nota dada por um usuário i ao item j ainda não avaliado, com a(i,j) igual a 0.

$$P(usuario2, item2) = N(usuario2, item1) - N(usuario1, item1) + N(usuario1, item2) \tag{1}$$

Usando como base a equação1, tem-se para a análise dos dados levando em consideração todos os usuários e suas avaliações, o cálculo da média das diferenças entra as notas dos usuários para gerar uma predição (definida na equação 2).

$$P(usurioi, itemj) = \{\sum (N(usurioi + 1, itemj) - N(usurioi, itemj) + N(usurioi, itemj + 1))/|usurios|\} \tag{2}$$

Com a álgebra explanada, o algoritmo escolhido entre os três desenvolvidos por Lemire et al. [2] é o Weighted Slope One, que além desta interpolação leva em consideração o desvio padrão entre cada par de item dos dados para

melhorar a predição. Para isso a programação foi dividida em duas fases, a primeira onde é computado os desvios padrões e as frequências de pares de itens avaliados em conjuntos e depois o cálculo das predições.

Então para cada usuário, deve-se buscar as notas de cada item, computando cada uma das notas e a frequência com que um par de itens é avaliado em conjunto por um mesmo usuário, junto do desvio padrão entre a avaliação dos dois itens. Ainda nessa fase, no final, deve-se iterar por cada um dos desvios e dividí-los pela frequência dos pares, como representado no trecho de código em Python Spark, abaixo.

```
for ratings in self.data.values():
    for (i, first) in ratings.items():
        self.freqs.setdefault(i, {})
        self.devs.setdefault(i, {})
        for (j, second) in ratings.items():
            if i != j:
                self.freqs[i].setdefault(j, 0)
                self.freqs[i][j] += 1
                self.devs[i].setdefault(j, 0.0)
                self.devs[i][j] += first - second

for (i, ratings) in self.devs.items():
    for j in ratings:
        ratings[j] /= self.freqs[i][j]
```

Em seguida, para cada item avaliado pelo usuário e todos os items que o usuário não avaliou deve-se calcular a predição da nota pela nota de outros usuários e sua frequência, descrito no código a seguir.

```
for (i, first) in userRatings.items():
    for (j, second) in self.devs.items():
        if j not in userRatings and i in self.devs[j]:
            auxFreq = self.freqs[j][i]
            recs.setdefault(j, 0.0)
            freqs.setdefault(j, 0)
            recs[j] += (second[i] + first) * auxFreq
            freqs[j] += auxFreq

recommendations = sc.parallelize([(x, y / freqs[x]) for (x,
    y) in recs.items()])
```

```
        recommendationsSorted = recommendations.sortBy(lambda xy:
            xy[1], False)
```

E executando esse códigos em uma base de dados (descrita na seção 1) conseguimos obter predições para qualquer usuário. Porém, como dito na seção 1, este projeto tinha como objetivo apresentar o uso do paradigma funcional de programação para uso em grandes bases, mas esta implementação usa também o paradigma imperativo. Infelizmente, não foi possível em tempo hábil, conseguir o mesmo processamento nas RDDS para otimização de performance. Algumas dificuldades foram encontradas na programação deste algoitmo com essas estruturas, pois não é possível calcular outra estura, como os desvios e as frequências, dentro da estrutura principal de dados usando o MapReduce, que era determinante para o funcionamentodo algoritmo.

Assim, foi impossível testar na base dados MovieLens [3], pois todo processamento estourava a memória usada na máquina local, pois também não foi possível usar servidores remotos para o processamento. Para remediar esta falha outro algoritmo foi buscado, chamado Alternate Least Squares (ALS) da biblioteca MlLib, e implementado de forma funcional, como apresentado a seguir.

```
evalsFormatedRDD = (
    evalsRDD
    .map(lambda x: x.split())
    .map(lambda x: Rating(x[0], x[1], float(x[2])))
)

trainModel = ALS.train(evalsFormatedRDD, 10, 5)
testData = evalsFormatedRDD.map(lambda x: (x[0], x[1]))
preds = trainModel.predictAll(testData).map(lambda x: ((x[0],
    x[1]), x[2]))
```

Mesmo implementando este algoritmo de forma funcional, não foi possível processar a base de dados de 700mb disponibilizada [3], também por estouro de memória. E isso, pode ter acontecido pois os algoritmos foram testados em uma única máquina e não em vários servidores como deveria ser feito na prática.

Todavia, os algoritmos foram usados em porções pequenas da base de dados e foi possível gerar predições condizentes e mostrar que os algoritmos

funcionam.

3. Conclusão

Ainda que todos os objetivos deste trabalho não tenham sido conquistados, parte da implementação foi feita almejando o paradigma funcional e a análise de uma grande quantidade de dados. O trabalho desenvolvido, foi usado dentro da empresa onde trabalho e foi bem recebido, gerando novas oportunidades.

Tem-se como pretensão, para minha monografia, o uso de dados para predição de comportamento do usuário, então as lições aprendidas, tanto no desenvolvimento deste projeto e ao longo do curso, serão muito úteis no futuro. E agora o paradigma funcional, também entrou como um dos itens, que serão usados nos próximos trabalhos.

[1] G. Linden, B. Smith, J. York, Amazon.com recommendations: Item-to-item collaborative filtering, IEEE Internet Computing 7 (2003) 76–80.

[2] D. Lemire, A. Maclachlan, Slope one predictors for online rating-based collaborative filtering 5 (2007).

[3] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, ACM Trans. Interact. Intell. Syst. 5 (2015) 19:1–19:19.