Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação

Trabalho Prático 1

Algoritmos de Recomendação

Disciplina: SCC0284 – Sistemas de Recomendação
Prof. Dr. Marcelo G. Manzato

Aluno
Felipe Scrochio Custódio

NUSP
9442688

Índice

Implementação

O trabalho foi implementado em linguagem **Python 3**, em ambiente Linux.

```
.
├──── assignment1.pdf
├──── assignment1.py
├──── csv
│    ├──── movies_data.csv
│    ├──── submit_file.csv
│    ├──── test_data.csv
│    ├──── train_data.csv
│    └──── users_data.csv
├──── plots
│    ├──── initial_ratings.png
│    ├──── rmse.png
│    ├──── time_baseline.png
│    ├──── time_elapsed.png
│    ├──── time_itemCF.png
│    └──── time_rfrec.png
├──── requirements.txt
├──── results
│    ├──── baseline.csv
│    ├──── itemCF.csv
│    └──── rfrec.csv
```

A implementação está na raiz, no arquivo *assignment1.py*.
A base de dados é carregada da pasta csv.
A pasta plots contém gráficos para auxiliar na visualização dos resultados dos algoritmos e da base de dados.

Pacotes utilizados estão listados no arquivo *requirements.txt.*

```
seaborn==0.8.1
pandas==0.20.3
matplotlib==2.0.2
numpy==1.14.1
ipython==6.3.1
progressbar2==3.37.1
scikit_learn==0.19.1
termcolor==1.1.0
```

A instalação dos pacotes necessários pode ser feita com o seguinte comando:

```
pip install -r requirements.txt
```

Para execução:

```
python assignment1.py
```

A tela inicial oferece um menu para escolher qual algoritmo rodar. O script irá rodar para todos os casos de teste em *test_data.csv*. Em seguida, irá fazer a análise de desempenho de todos os algoritmos, rodando para uma certa porção da base de treino *train_data.csv*, armazenando os tempos de execução e o RMSE entre a avaliação e a predição. Os gráficos gerados são armazenados na pasta *plots*.

```
Recommender Systems - Assignment 1

Choose recommendation algorithm:
        1 - Item-Item Collaborative Filtering
        2 - Baseline
        3 - RF-Rec
        4 - Bayes
        5 - SVD
        6 - SGD
        7 - Code profiling only
:::
```

Para a análise de desempenho, é pedido um número de casos de teste. Esse número será usado para fazer o *split* da base de treino.

Os resultados documentados foram obtidos com toda a base de treino, com as submissões no Kaggle. Algoritmos implementados seguindo as anotações feitas em sala de aula e os slides disponibilizados.

Algoritmos implementados:

- Filtragem colaborativa baseada em vizinhança de itens
- Baseline
- RF-Rec

Algoritmos não implementados ou com problemas:

- Bayes
    - Implementado, porém extremamente custoso e lento
- SVD
    - Esqueleto da implementação, método de decomposição utilizado (*np.linalg.svd*) não rodou para a matriz (usuários x filmes) inteira.
- SVD com Gradiente Descendente Estocástico
    - Não implementado

Desempenho

Filtragem colaborativa baseada em vizinhança de itens

Escolha do K

A matriz de avaliações é bem esparsa. No mapa de calor da figura 1 (usuários x filmes), podemos ver que existem longas seções de itens com poucas avaliações, e vários usuários que não assistiram quase nenhum filme. Isso leva a um problema: com um valor de K muito elevado, os valores de similaridade são extremamente baixos. Muitas vezes, os valores eram tão baixos que os somatórios levavam a valores flutuantes extremamente baixos e até mesmo com erros, resultando em divisões por zero ou números negativos.

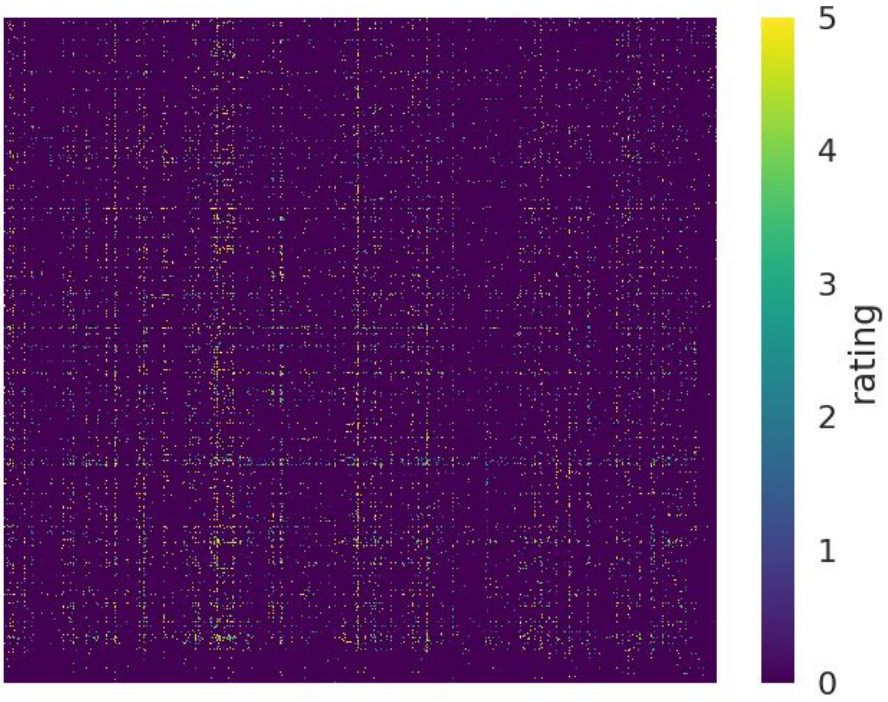

Gráfico 1: Mapa de calor das avaliações iniciais.

Foram feitos testes com K = 100, K = 50, K = 30 e K = 20. Infelizmente, para valores acima de 20, os problemas citados acima aconteceram para muitos casos de teste, fazendo a execução ficar muito mais demorada também. O K com os melhores resultados foi **K = 20**.

Resultados

RMSE: 1.08589

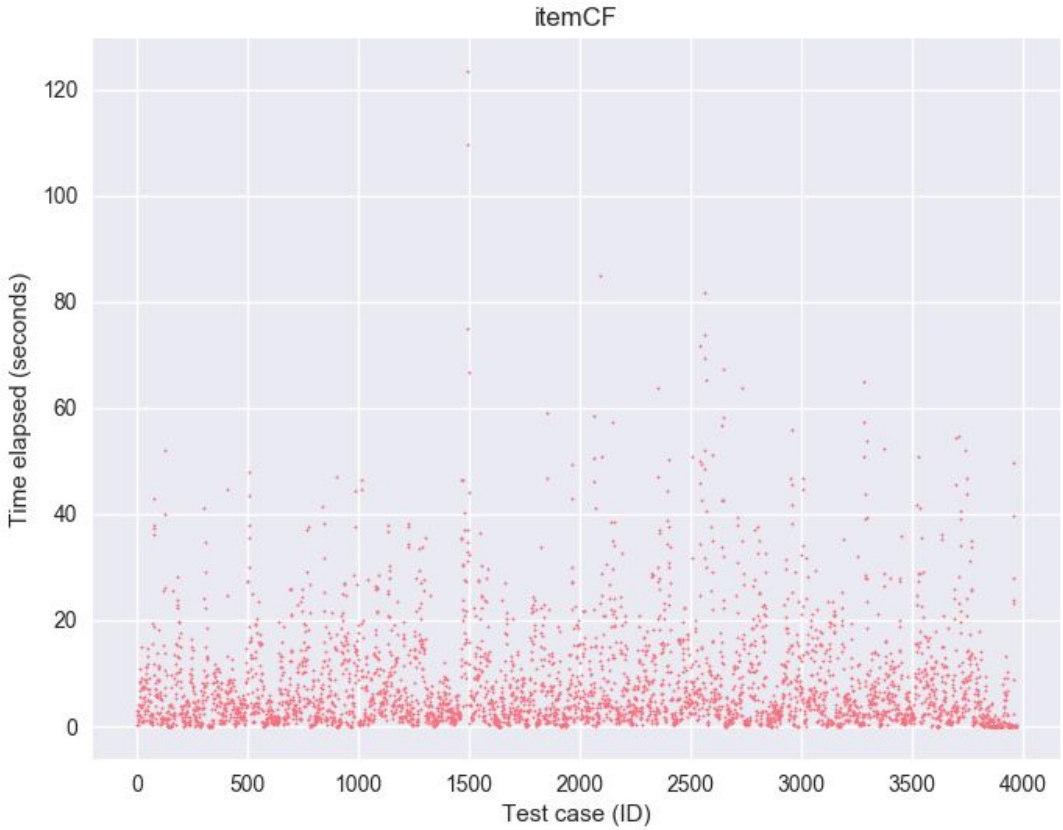

Gráfico 2: Tempos de execução para a FC baseada em vizinhança de itens.

RF-Rec

Resultados

RMSE: 1.12583

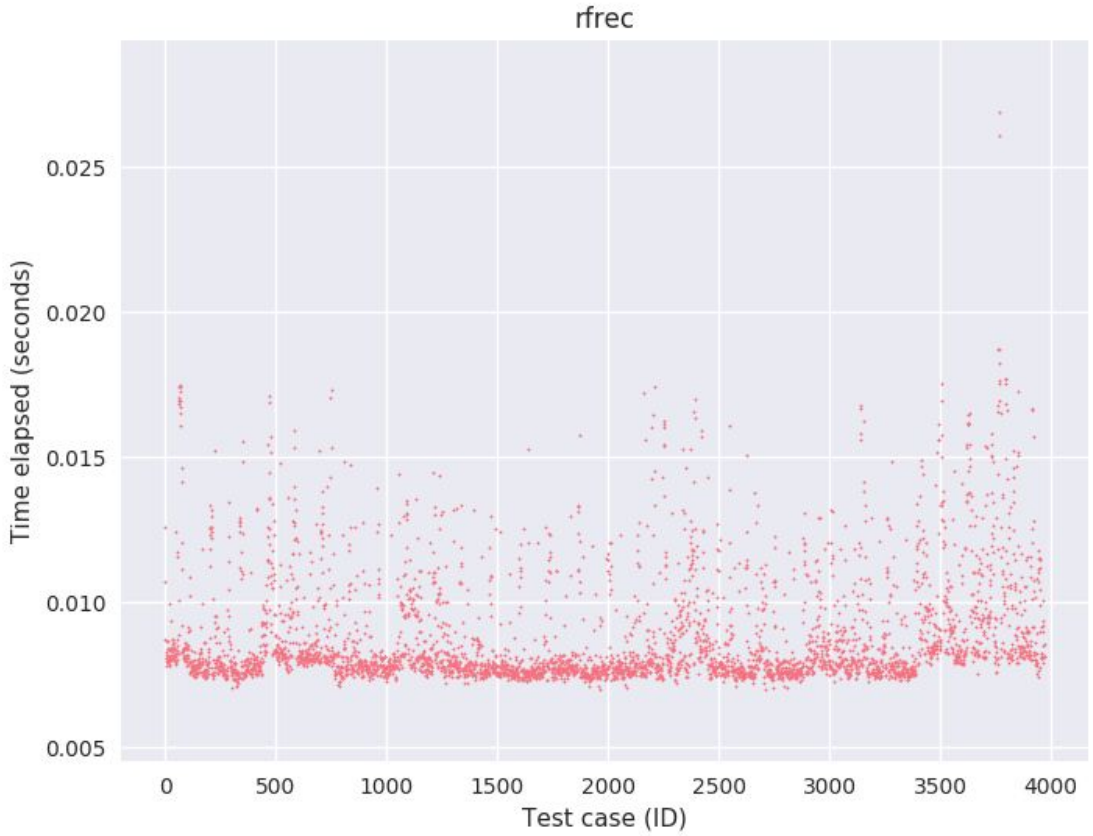

Gráfico 3: Tempos de execução para o RF-Rec.

Método Baseline

Resultados

RMSE: 1.06439

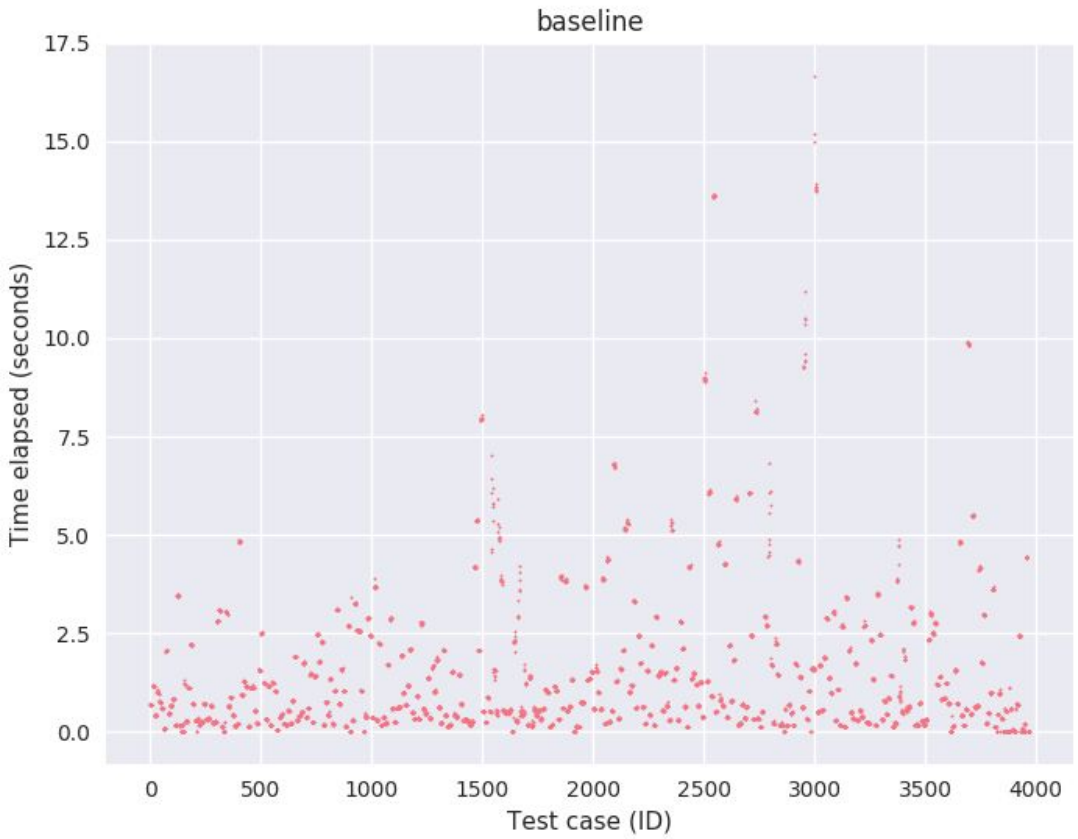

Gráfico 4: Tempos de execução para o Baseline.

Discussão

Problemas e decisões de projeto

A grande maioria das operações foram implementadas puramente com loops e condições, sem utilizar operações vetoriais do Python ou das bibliotecas como Numpy e SciPy (que são altamente otimizadas para operações em larga escala). Isso deixou a performance terrível e praticamente inviável.

Inicialmente, leio os arquivos .csv utilizando um *DataFrame* da biblioteca Pandas. As operações feitas pelo próprio Pandas são extremamente rápidas, porém com um alto custo de memória: todas as vezes que o Pandas executa alguma operação, ele cria uma cópia dos dados.

Optei por utilizar uma matriz de usuários x itens, por achar mais intuitiva e didática. Era assim que estava visualizando os algoritmos até então ao estudar para a disciplina. Porém, todas as operações acabaram ficando extremamente custosas, dado o tamanho da matriz e o tempo para percorrer todas às avaliações. Para resolver isso deveria ter utilizado todo o potencial de um *DataFrame* Pandas ou matrizes do NumPy e seus métodos. Colegas que fizeram utilizando esses métodos tiveram resultados em poucos segundos, ao invés de horas.

Filtragem colaborativa baseada em vizinhança de itens

O gargalo desse algoritmo com certeza é o cálculo de similaridades. Para todos os itens, é calculada a similaridade de todos os outros que o usuário avaliou. Calcular todas as similaridades previamente seria a melhor solução, gerando uma matriz de todos os itens para todos os itens. Porém, o custo em Python dessa operação se tornou inviável, calculando a similaridade sem métodos especiais de bibliotecas como Numpy ou Scipy. Portanto, optei por calcular durante a execução, para cada caso de teste, as similaridades necessárias. Caso fosse utilizar esse algoritmo novamente, geraria a matriz de similaridades antes de fazer as predições, utilizando alguma ferramenta que faça isso para um grande volume de dados de forma rápida, como a biblioteca MyMediaLite, implementada em C#.

Baseline

O método Baseline apresentou problemas em poucos casos de teste, retornando uma predição menor do que 1, em 14 testes dos 3970. Demorou cerca de 1h30m para rodar todos os casos de teste e gerar o csv de submissão. Imagino que a demora seja devido a muitas referências, como os *slices* de linhas e colunas para pegar as notas apenas do usuário ou do item.

RF-Rec

De todos os algoritmos, o RF-Rec foi o que rodou mais rápido. Por outro lado, seu RMSE foi mais elevado. Foi a primeira submissão no Kaggle, devido à simplicidade da implementação (comparado com os outros algoritmos).

Bayes

Por calcular todas as probabilidades de cada nota sempre para cada filme que o usuário avaliou, o tempo de execução ficou extremamente alto, não conseguindo obter resultados para submeter no Kaggle. Uma possível melhoria seria fazer esses cálculos previamente e não a cada iteração. A segunda melhoria seria a mesma dos outros algoritmos, utilizar extensivamente métodos de bibliotecas como NumPy e SciPy.