01

Trabalhando com dados reais

Transcrição

Até o momento trabalhamos com dados de teste para simular a lógica de recomendação, estes dados presentes no arquivo *dados.csv* são de um exemplo que o próprio **Mahout disponibiliza**. Agora vamos utilizar dados reais!

No arquivo cursos.csv (https://github.com/alura-cursos/machine-learning-introducao-aos-sistemas-de-recomendacoes/blob/master/src/main/resources/cursos.csv) temos dados reais de avaliações de cursos. Este arquivo é bem diferente do anterior que estávamos utilizando por que a escala de avaliação vai de 0 a 10 e a quantidade de usuários e cursos também é muito maior, são 34.744 linhas e ele está no formato *CSV*.

> **Relembrando os passos!** Para efetivamente recomendarmos algo temos que: ler o arquivo de dados, criar o modelo de análise, criar as funções de proximidade e similaridade, criar o recomendador baseado no modelo e nas funções de análise para no fim, recomendarmos um curso. Se notar é o código que está escrito na classe `RecomendaProdutos`.

Antes de usarmos os dados reais vamos refatorar um pouco nosso projeto para simplificar algumas operações e tornar reusável partes de código que deixamos duplicadas nas classes.

Na classe `RecomendaProdutos` podemos inicialmente criar o recomendador da forma como já estávamos fazendo e utilizamos o *Builder* da aula anterior, passando apenas o modelo que é feito a partir da leitura do arquivo. Dessa forma, o que antes estava assim:

```
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, similar
```

Passa a ficar da seguinte forma:

```
Recommender recommender = new RecomendadorDeProdutosBuilder().buildRecommender(model);
```

Criamos o recomendador direto do *Builder* e com isso eliminamos a primeira duplicação de código. A segunda refatoração é porque a leitura dos dados não está muito clara, não sabemos se estamos lendo de produtos ou de cursos, além disso, nas classes `RecomendaProdutos` e `Avaliador` estamos duplicando a leitura do arquivo.

Para resolver essa questão, criaremos uma nova classe e nela inserimos um novo método que retorna o modelo de produtos especificamente. A classe se chamará `Recomendador` e o método `getModeloProdutos`.

```
public class Recomendador {

    public DataModel getModeloProdutos() throws IOException {
        File file = new File("dados.csv");
        return new FileDataModel(file);
    }
```

```
}
```

Dessa forma isolamos a criação do modelo e ele não fica espalhado em muitas partes do código. Agora nas classes `RecomendaProdutos` e `Avaliador` onde tínhamos a criação do modelo, teremos apenas uma linha que retorna o modelo pronto. O que estava assim em ambas as classes:

```
File file = new File("dados.csv");
FileDataModel model = new FileDataModel(file);
```

Agora fica da seguinte forma:

```
DataModel produtos = new Recomendador().getModeloProdutos();
```

Observe que mudamos o nome da variável de `model` para `produtos` justamente para que o código fique mais claro. Agora sabemos de vista que se trata de um modelo de produtos e não de um modelo sem descrição alguma. Com as classes que já temos, como fazemos para que possamos fazer recomendações de cursos ao invés de produtos?

Primeiro, criamos o método que cria o modelo pronto para que possamos utilizar na classe `Recomendador` da mesma forma que fizemos com os produtos:

```
public DataModel getModeloDeCursos() throws IOException {
    File file = new File("cursos.csv");
    return new FileDataModel(file);
}
```

E criamos a classe `RecomendaCursos` com o método `main` utilizando o `RecomendadorDeProdutosBuider` para criar nosso recomendador:

```
public class RecomendaCursos {
public static void main(String[] args) throws IOException, TasteException {
    DataModel cursos = new Recomendador().getModeloDeCursos();
    Recommender recommender = new RecomendadorDeProdutosBuilder().buildRecommender(cursos);

}
```

Já que estamos falando um pouco de refatoração, podemos melhorar um pouco mais a criação do modelo de análise, a classe `Recomendador` atualmente está assim:

```
public class Recomendador {

    public DataModel getModeloProdutos() throws IOException {
        File file = new File("dados.csv");
        return new FileDataModel(file);
    }

    public DataModel getModeloDeCursos() throws IOException {
        File file = new File("cursos.csv");
        return new FileDataModel(file);
```

```
    }

}
```

Com algumas modificações ela passa a ficar assim:

```
public class Recomendador {
    public DataModel getModeloProdutos() throws IOException {
        return getModelo("dados.csv");
    }

    private DataModel getModelo(String path) throws IOException {
        File file = new File(path);
        return new FileDataModel(file);
    }

    public DataModel getModeloDeCursos() throws IOException {
        return getModelo("cursos.csv");
    }
}
```

O que mudou foi: ao invés de ler o arquivo e retornar seu modelo duas vezes, fizemos isso uma vez só em um método privado e nos método públicos chamamos este método indicando apenas qual será o arquivo de dados.

> Para estas e outras técnicas de refatoração recomendamos o curso Refatorando na prática (https://cursos.alura.com.br/course/refatorando-na-pratica-com-java)

Agora, vamos começar a pedir recomendações diretamente. Por exemplo, 3 cursos para o usuário 1.

```
List<RecommendedItem> recommendations = recommender.recommend(1, 3);
for (RecommendedItem recommendation : recommendations) {
    System.out.println(recommendation);
}
```

Como resultado teremos:

```
RecommendedItem[item:426, value:10.0]
RecommendedItem[item:302, value:10.0]
RecommendedItem[item:396, value:9.767327]
```

Você pode notar que dessa vez as recomendações demoraram mais um pouco do que o que estávamos acostumados, isto é, estamos analisando agora 34.744 preferências em um número de 4.637 usuários. Agora, que cursos foram recomendados para o usuário 1?

Se abrirmos o arquivo `cursos.csv` veremos, por exemplo, que o usuário 1, fez o curso 246, que é o curso de Jogos com Android. Outro curso que este mesmo usuário fez foi o 158, que é o curso de JAX-RS, ou seja, de serviços web baseados em REST. Apenas dois cursos foram feitos pelo usuário 1, mas será que isso é suficiente para analisarmos as recomendações para ele?

A única informação que temos é que o usuário 1 fez cursos da plataforma Java e Android. Um para jogos e outro para serviços web. Mas que cursos nosso recomendador sugeriu?

O curso 426 é o curso de **Threads** que é de Java e que esta diretamente relacionado tanto ao mundo web quanto ao Android. O curso 302 por outro lado, é de 3D, que a primeira vista poderia não fazer sentido, mas faz, pois o usuário fez um curso de jogos e os conhecimentos em 3D podem ajudá-lo a criar interfaces deste tipo. Por último, tivemos o curso 396 que é sobre certificação Oracle, neste caso, não temos uma relação próxima das áreas de conhecimento.

Para irmos mais além, vamos pedir 6 recomendações para ver as demais sugestões que o algoritmo fará para este usuário que tem pouquíssimas informações em nossa base de dados. Nestes casos adicionais o algoritmo sugeriu:

```
RecommendedItem[item:302, value:10.0]
RecommendedItem[item:426, value:10.0]
RecommendedItem[item:429, value:9.767327]
RecommendedItem[item:428, value:9.767327]
RecommendedItem[item:396, value:9.767327]
RecommendedItem[item:439, value:9.652684]
```

Estes itens estão além dos anteriores, mostra mais dois cursos de certificação Oracle e um de expressões regulares. O de certificação pode ter sido recomendado por causa de alguma relação entre quem faz cursos Java, também faz da certificação, enquanto o de expressões regulares é um curso muito feito porque várias linguagens utilizam os conceitos dessa técnica.

Note que nosso recomendador funciona mesmo com poucas informações sobre o usuário, já que o usuário 1 tem apenas duas preferências salvas. Vamos analisar o usuário 15, por exemplo, que fez 4 cursos. São eles: 156) Curso de ASP.NET MVC, 133) Curso de RAZOR, 230) Curso de WCF e 178) Projeto Completo .NET.

O usuário 15 fez bastante cursos da plataforma Microsoft, todos os seus cursos até o momento são dessa categoria: Microsoft e Web. Pedimos 6 recomendações para o usuário 15, vejamos os resultados:

```
RecommendedItem[item:208, value:10.0]
RecommendedItem[item:284, value:10.0]
RecommendedItem[item:175, value:10.0]
RecommendedItem[item:158, value:10.0]
RecommendedItem[item:236, value:9.888109]
RecommendedItem[item:120, value:9.6]
```

O curso 208 é de Orientação a Objetos com PHP e tem alguma relação de conceito com os da Microsoft, porém com PHP. O 284 é de Spring MVC que é equivalente aos cursos de Web da Microsoft, porém na plataforma Java. O 175 é de Android, que não está relacionado aos demais. Os demais são 158, 236 e 120 que são cursos de JAX-RS, Laravel e PHP e MySQL.

O que percebe-se é que para o primeiro caso, o algoritmo recomendou cursos relacionados que poderiam ajudar o aluno a se aprofundar nos assuntos ligados a jogos e web, já no segundo caso o recomendador sugere cursos análogos ao que o aluno já fez porém em outras plataformas.