02

# Criando avaliador

### Transcrição

Já vimos o problema da taxa de erro e agora precisamos criar um avaliador que fará os testes para a gente. Portanto, criaremos uma nova classe chamada avaliador e nela teremos o método `main`.

```
public class Avaliador{
  public static void main(String[] args){

  }
}
```

Já sabemos que nosso avaliador precisa considerar a média da diferença dos valores absolutos. Então criaremos um objeto da classe `AverageAbsoluteDifferenceRecommenderEvaluator`:

```
RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
```

Com este objeto em mãos, podemos pedir para que ele faça as avaliações que já vimos como realizar manualmente. Porém ele precisa de algumas informações para funcionar.

```
evaluator.evaluate(buider, null, model, 0.9, 1.0)
```

A primeira informação é um construtor de recomendações, que ainda não temos, mas iremos criar. O segundo é um construtor de modelos, mas para este há uma alternativa no terceiro argumento no qual podemos passar o modelo pronto e os últimos dois argumentos que são a quantidade da amostra em porcentagem que desejamos utilizar para treinar o algoritmo e também testá-lo.

O modelo a gente já sabe como criar. Podemos apenas copiar da classe `RecomendaProdutos`.

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

E o Builder? Vamos criá-lo! Primeiro precisamos criar uma classe que implemente a interface `RecommenderBuilder` e esta classe se chamará `RecomendadorDeProdutosBuilder`.

```
public class RecomendadorDeProdutosBuilder implements RecommenderBuilder {
  public Recommender buildRecommender(DataModel model) throws TasteException {
    return null;
  }
}
```

Esta classe tem um método `buildRecommender` que precisa retornar um recomendador, ou seja, o outro trecho de código que já fizemos na classe `RecomendaProdutos`. Assim teremos na classe `RecomendadorDeProdutosBuilder` o seguinte:

```
public class RecomendadorDeProdutosBuilder implements RecommenderBuilder {
  public Recommender buildRecommender(DataModel model) throws TasteException {
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
    UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, sim
    return recommender;
  }
}
```

E na classe `Avaliador` teremos:

```
public class Avaliador{
  public static void main(String[] args) throws IOException, TasteException{
    File file = new File("dados.csv");
    FileDataModel model = new FileDataModel(file);

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
    RecommenderBuilder builder = new RecomendadorDeProdutosBuilder();
    double erro = evaluator.evaluate(buider, null, model, 0.9, 1.0);
    System.out.println(erro);
  }
}
```

Note que o avaliador retorna a taxa de erro que calculamos anteriormente. É importante notar que o cálculo do número pode variar de acordo com a implementação do algoritmo. Ao executarmos isso obtemos o resultado de 0.298873662948660084.

O valor 0.29 é um número bem menor que 1.18. Considerando isso, a probabilidade de avaliarmos bem uma recomendação melhorou bastante. Caso o recomendador chute um valor 4 para sua avaliação, a variação do valor real estará entre 3.71 e 4.29.

Concluímos que um avaliador é importantíssimo para testar se o algoritmo está próximo ou não baseado na sua taxa de erro.