Curso 03 - Exercícios de fixação
Quinzena 03 - Prof. Jose Fernando Rodrigues Junior

1) O Apache Mahout Taste é um subprojeto que provê funcionalidades de recomendação por meio da técnica de Filtragem Colaborativa (*Collaborative Filtering)*. Em sua forma mais comum, esta técnica se baseia na identificação de usuários com gostos semelhantes com a suposição de que os itens consumidos por tais usuários são de interesse mútuo. Considerando o código Java a seguir, visto em aula:

```
public static void main(String[] args) throws Exception {
  DataModel model = new FileDataModel(new File("Dados.csv"));
  UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
  UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);
  Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
  List<RecommendedItem> recommendations = recommender.recommend(1,2);
}
```

a) Identifique os 3 elementos necessários à realização da Filtragem Colaborativa.

Modelo de dados:
`DataModel model = new FileDataModel(new File("Dados.csv"));`

Medida de Similaridade:
`UserSimilarity similarity = new PearsonCorrelationSimilarity(model);`

Vizinhança:
`UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);`

b) Altere o código para a definição de uma vizinhança de tamanho 3, e para a recuperação de 10 itens recomendados.

```
UserNeighborhood neighborhood = new NearestNUserNeighborhood(3, similarity, model);

List<RecommendedItem> recommendations = recommender.recommend(1,10);
```

c) Pesquise a API Taste em http://mahout.apache.org/docs/0.13.0/api/docs/mahout-mr/org/apache/mahout/cf/taste/impl/similarity/AbstractItemSimilarity.html e altere o código dado para usar a função de similaridade Distância Euclideana.

```
import org.apache.mahout.cf.taste.impl.similarity.EuclideanDistanceSimilarity;
UserSimilarity similarity = new EuclideanDistanceSimilarity(model);
```

d) Pesquise a API Taste em https://mahout.apache.org/docs/0.13.0/api/docs/mahout-mr/org/apache/mahout/cf/taste/neighborhood/UserNeighborhood.html e altere o código para usar a vizinhança baseada em threshold, de modo que todos os usuários com similaridade até 0.5 sejam considerados para a definição da Filtragem Colaborativa.

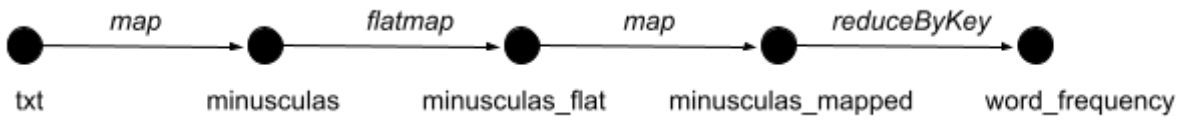
```
import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.5, similarity, model);
```

2) Considere o código Spark visto em aula:

```
import pyspark
sc = pyspark.SparkContext('local[*]')
txt = sc.textFile('file:////usr/share/doc/python3/copyright')
def minuscula_limpa(palavra):
    palavra = palavra.lower()
    simbolos = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
    for ch in simbolos:
        palavra = palavra.replace(ch, '')
    return palavra

minusculas = txt.map(minuscula_limpa)
minusculas_flat = minusculas.flatMap(lambda line: line.split(" "))
minusculas_mapped = minusculas_flat.map(lambda word: (word, 0) if word == "" else
(word, 1))
word_frequency = minusculas_mapped.reduceByKey(lambda a,b:a +b)
word_frequency_sorted = word_frequency.sortBy(lambda a:a[1], ascending=False)
word_frequency_sorted_collected = word_frequency_sorted.collect()
```

a) Escreva o grafo acíclico que descreve seu *lineage* Spark relativo à action reduceByKey:



b) Adicione uma nova transformação para retirar as stop_words, i.e., as palavras que aparecem frequentemente, mas que não auxiliam no processo analítico. Considere a seguinte lista de palavras:

```
stop_words = ('the', 'or', 'of', 'to', 'and', 'in',  'a', 'by',  'for', 'as')
```

Após criar o RDD minusculas_flat:

```
no_stop_words = minusculas_flat.filter(lambda x: x not in stop_words)
minusculas_mapped = no_stop_words.map(lambda word: (word, 0) if word == "" else
(word, 1))
```