01_movielens_cluestering

October 25, 2016

1 Movielens - Cluesterização

Projeto da disciplina de Data Mining
PESC - Programa de Engenharia de Sistemas e Computação
COPPE / UFRJ
Autor: Rafael Lopes Conde dos Reis
E-mail: condereis@cos.ufrj.br

1.1 1 - Resumo

O trabalho consiste em analisar a aplicação da técnica K-means para clusterizar os filmes da base do MovieLens. Deve-se observar os clusters gerados para diferentes valores de k (15-30), assim como antes e depois de executar redução de dimensionalidade, com PCA (10-15 dimensões).

1.2 2 - Pacotes Utilizados

```
In [22]: import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.cluster import KMeans
         from sklearn.decomposition import PCA

         %matplotlib inline
```

1.3 3 - Pré-processamento

A base fornecida pelo MovieLens contém 4 colunas (userId, movieId, rating e timestamp), como pode ser visto abaixo. O pré-processamento irá gerar uma matriz *usuários x filmes* tendo todas as avaliações faltantes substituidas pela média da avaliação do filme.

```
In [2]: raw_data = pd.read_csv('../data/movielens/ratings.csv')
        raw_data.head()
```

```
Out[2]:    userId  movieId  rating   timestamp
        0       1       31     2.5  1260759144
        1       1     1029     3.0  1260759179
```

```
2       1     1061     3.0  1260759182
3       1     1129     2.0  1260759185
4       1     1172     4.0  1260759205
```

```
In [3]: # Dataframe com colunas identificadas pelos ids dos filmes
        ratings_df = pd.DataFrame(columns=raw_data.movieId.sort_values().unique())

        for index, row in raw_data.iterrows():
            ratings_df.set_value(row.userId, row.movieId, row.rating)
```

O resultado é uma matriz repleta de NaNs. Esses valores serão substituidos pela nota média do filme.

```
In [12]: # Substitui NaNs pela média da coluna
         ratings_df.fillna(ratings_df.mean(), inplace=True)
         ratings_df.head()
```

```
Out[12]:       1         2         3         4         5         6         7
         1.0  3.87247  3.401869  3.161017  2.384615  3.267857  3.884615  3.283019
         2.0  3.87247  3.401869  3.161017  2.384615  3.267857  3.884615  3.283019
         3.0  3.87247  3.401869  3.161017  2.384615  3.267857  3.884615  3.283019
         4.0  3.87247  3.401869  3.161017  2.384615  3.267857  3.884615  3.283019
         5.0  3.87247  3.401869  4.000000  2.384615  3.267857  3.884615  3.283019

              8       9        10       ...    161084  161155  161594  161830  161
         1.0     3.8    3.15  3.45082   ...       2.5     0.5     3.0     1.0     1
         2.0     3.8    3.15  4.00000   ...       2.5     0.5     3.0     1.0     1
         3.0     3.8    3.15  3.45082   ...       2.5     0.5     3.0     1.0     1
         4.0     3.8    3.15  4.00000   ...       2.5     0.5     3.0     1.0     1
         5.0     3.8    3.15  3.45082   ...       2.5     0.5     3.0     1.0     1

              161944  162376  162542  162672  163949
         1.0     5.0     4.5     5.0     3.0     5.0
         2.0     5.0     4.5     5.0     3.0     5.0
         3.0     5.0     4.5     5.0     3.0     5.0
         4.0     5.0     4.5     5.0     3.0     5.0
         5.0     5.0     4.5     5.0     3.0     5.0

         [5 rows x 9066 columns]
```

```
In [13]: # Salva o dataframe em arquivo para facilitar o acesso
         ratings_df.to_csv('../data/movielens/input_matrix.csv', index=False)
```

1.4 4 - Clusterização sem redução de dimensionalidade

O primeiro passo é aplicas k-means a matriz de dados completa, sem nenhuma forma de redução de dimensionalidade.

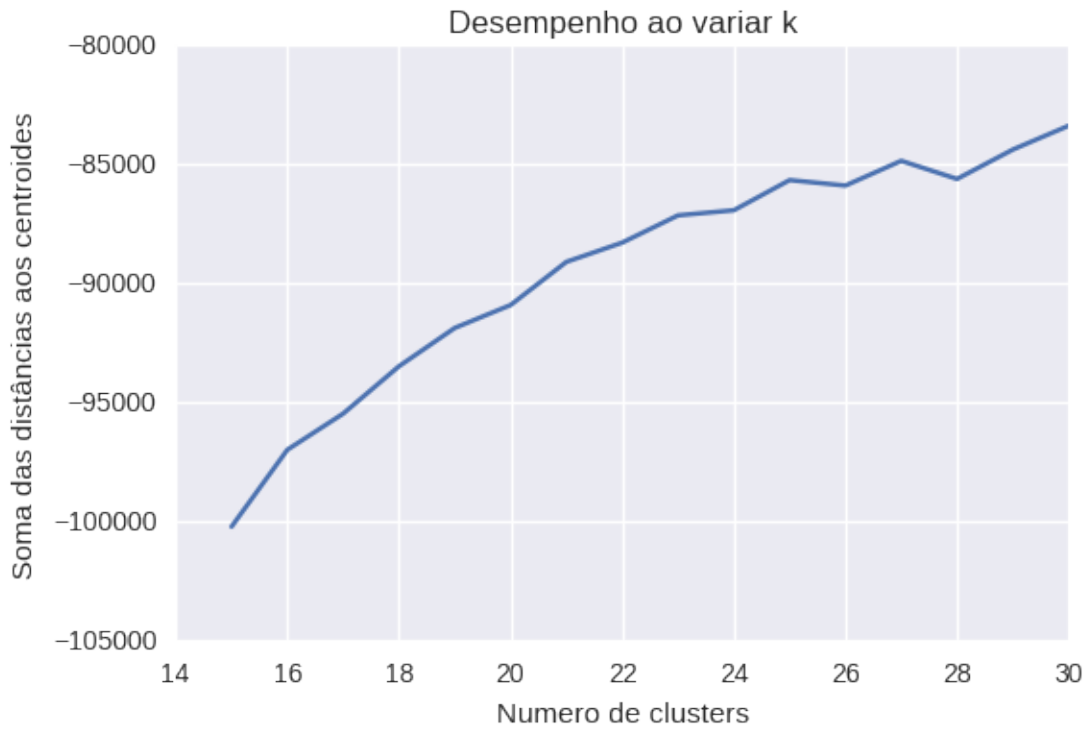
```
In [14]: ratings_df = pd.read_csv('../data/movielens/input_matrix.csv')
```

Serão treinados 15 modelos diferentes, variando o número de clusters entre 15 e 30. Para cada modelo foram feitas 10 inicializações (n_init=10) e escolhida a melhor, para tentar encontrar um modelo mais próximo do mínimo global, uma vez que k-means apenas garante convergência para um mínimo local. A inicialização usada foi a k-means++, que inicia os centroides afastados uns dos outros, o que tende a garantir melhores resultados que uma inicialização aleatória [1].

```
In [24]: score_list = []
         for k in range(15,31):
             kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10, n_jobs=-1).
             score_list.append(kmeans.score(ratings_df.transpose()))
```

```
In [31]: plt.plot(range(15,31), score_list)
         plt.title('Desempenho ao variar k')
         plt.xlabel('Numero de clusters')
         plt.ylabel(u'Soma das distâncias aos centroides')
```

```
Out[31]: <matplotlib.text.Text at 0x7fc8975d8910>
```




```
In [32]: import os
         os.environ['PATH']
```

```
Out[32]: '/home/rafael/virtualenvs/datascience/bin:/home/rafael/virtualenvs/datasci
```

```
In [ ]:
```

```
In [ ]:
```

1.5 Referências

[1] - "k-means++: The advantages of careful seeding" Arthur, David, and Sergei Vassilvitskii, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2007)

In [ ]: