Exercício de Programação 2
Sistemas de Recomendação

Sheldon Goulart de Alcântara
alcantarash92@gmail.com
Vinícius Teodoro
vtcpires@gmail.com

Abstract

This report aims to present some variations of recommendation systems for scientific articles.

1 Sistemas de Recomendação

O sistema de recomendação é definido como uma estratégia de tomada de decisão para usuários em ambientes de informações complexas. Além disso, o sistema de recomendação foi definido a partir do ponto de vista do comércio eletrônico como uma ferramenta que ajuda os usuários a pesquisar através de registros de conhecimento relacionados ao interesse e preferência dos usuários. Definido como um meio de auxiliar e aumentar o processo social de usar recomendações de outros para fazer escolhas quando não há conhecimento pessoal suficiente ou experiência das alternativas. Os sistemas lidam com o problema da sobrecarga de informações que os usuários geralmente enfrentam, fornecendo-lhes recomendações de conteúdo e serviço personalizadas.

No trabalho em questão, foi adotado quatro estratégias para recomendação de artigos científicos:

- Aleatória.
- Mais popular.
- Baseada em Conteúdo.
- Filtragem colaborativa.

2 Aleatória

É recomendado um artigo independente da preferência do usuário.

```
import numpy as np
import pandas as pd
from random import randint

raw_data = pd.read_csv('raw-data.csv', encoding='iso-8859-1');
users = pd.read_csv('users.dat',  header=None)
users[0] = users[0].str.split(' ')

def checkifRandom(vetor, index, y): #fun   o para checar se j   existe
    no vetor
    x = 0
    for i in vetor[index]:
        if int(i) == y:
            x = 1
            break
```

```
    if x == 1:
        checkifRandom(users[0], index, randint(0, len(vetor)))
    else:
        print('The recommendation for the user', index, 'is the article', y)
        resultRandom.append(y) #coloca no result o resultado
    return

def suggest_randomly(users): #fun  o de sugest o aleat ria
    len_user = len(users[0])
    for index, x in enumerate(users[0]):
        y = randint(0, len_user)
        checkifRandom(users[0], index, y)
    return


    Out (10 primeiros leitores):

('The recommendation for the user', 0, 'is the article', 4747)
('The recommendation for the user', 1, 'is the article', 3653)
('The recommendation for the user', 2, 'is the article', 4201)
('The recommendation for the user', 3, 'is the article', 2188)
('The recommendation for the user', 4, 'is the article', 3072)
('The recommendation for the user', 5, 'is the article', 678)
('The recommendation for the user', 6, 'is the article', 3978)
('The recommendation for the user', 7, 'is the article', 2915)
('The recommendation for the user', 8, 'is the article', 2469)
('The recommendation for the user', 9, 'is the article', 4884)
```

Listing 1: Recommender System Random

3 Mais Popular

Considerando o artigo mais lido no conjunto de usuários, é definido um ranking de popularidade entre os documentos lidos, e o mais popular é recomendado ao usuário.

```
import numpy as np
import pandas as pd

raw_data = pd.read_csv('raw-data.csv', encoding='iso-8859-1' );
users = pd.read_csv('users.dat',  header=None)
users[0] = users[0].str.split(' ')

articles = {}
for k in range(len(raw_data)):
    articles[k+1] = 0

#Vetor para armazenar quantas vezes um artigo foi lido
for i in range(len(users)):
    for j in users[0][i]:
        if int(j) != 0:
            articles[int(j)] = articles[int(j)] + 1

#Ordenando o vetor de artigos lidos
aux = []
articles_sorted = sorted(articles, key=articles.get, reverse=True)# "ranking" dos artigos lidos
for r in articles_sorted:
    print r, articles[r]
    aux.append(r)

#Verifica se o usu rio j  leu o artigo e recomenda o pr ximo mais popular
def checkReadMostPopular(vetor, index, position):
```

```
    read = 0
    for i in vetor[index]:
        if int(i) == aux[position]:
            read = 1
            break
    if read == 1:
        position = position + 1
        checkReadMostPopular(users[0], index, position)
    else:
        vetor[index].append(aux[position])
        print('The recommendation for the user', index, 'is the article'
    , aux[position])

#Fun  o para recomendar o artigo mais popular
def suggest_mostPopular(users):
    for index, x in enumerate(users[0]):
        checkReadMostPopular(users[0], index, 0)
    return

Out(10 primeiros leitores):

('The recommendation for the user', 0, 'is the article', 10)
('The recommendation for the user', 1, 'is the article', 10)
('The recommendation for the user', 2, 'is the article', 10)
('The recommendation for the user', 3, 'is the article', 10)
('The recommendation for the user', 4, 'is the article', 10)
('The recommendation for the user', 5, 'is the article', 10)
('The recommendation for the user', 6, 'is the article', 10)
('The recommendation for the user', 7, 'is the article', 14)
('The recommendation for the user', 8, 'is the article', 10)
('The recommendation for the user', 9, 'is the article', 10)
```

Listing 2: Most Popular

4 Baseada em Conteúdo

Um sistemas de recomendação baseado em conteúdo funciona com dados que o usuário fornece, nesse caso os artigos lidos pelo mesmo. Com base no abstract dos artigos lidos, foi usado o valor de TF-IDF associado ao método do vizinho mais próximo (kNN).

TF-IDF consiste nume medida estatística que tem o intuito de indicar a importância de uma palavra de um documento em relação a uma coleção de documentos.

Sendo assim, TF conta o número de vezes que um determinado termo ocorre em cada documento e soma esse valor. O número de vezes que um termo ocorre em um documento é a frequência do termo (Term Frequency - TF). Enquanto IDF é incorporado para diminuir o peso dos termos que ocorrem mais frequentemente no conjunto de textos selecionados, ao mesmo tempo que aumenta o peso daqueles que ocorrem raramente[1].

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
import numpy

raw_data = pd.read_csv('raw-data.csv', encoding='iso-8859-1', index_col='
    doc.id' );
users = pd.read_csv('users.dat',  header=None)
users[0] = users[0].str.split(' ')

tfidf_vectorizer = TfidfVectorizer()
raw_data['tfidf']  = list(tfidf_vectorizer.fit_transform(raw_data['raw.
    abstract']))
```

```

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
raw_data['tfidf']  = list(tfidf_vectorizer.fit_transform(raw_data['raw.
    abstract']))

tfidf_matrix = tfidf_vectorizer.fit_transform(raw_data['raw.abstract'])
nbrs         = NearestNeighbors(n_neighbors=2).fit(tfidf_matrix)

def get_similar_articles(doc_id):#Retorna o primeiro artigo mais similar
    ao artigo lido
    row                = raw_data.index.get_loc(doc_id)
    distances, indices = nbrs.kneighbors(tfidf_matrix.getrow(row))
    names_similar      = pd.Series(indices.flatten()).map(raw_data.
    reset_index()['doc.id'])
    return list(names_similar)[1:2]

def otherArticles(vetArtLidos):#Cria uma lista com os artigos similares
    aos artigos lidos
    listarticles = []
    for i in vetArtLidos:
        if (int(i) != 0):
            similar_articles = get_similar_articles(int(i))
            listarticles.append(similar_articles)
    return list(set([y for x in listarticles for y in x]))

#Fun  o para checar se o artigo recomendado j  foi lido pelo usu rio
def checkReadBasedContent(vetor, index, articles, position):
    read = 0
    for i in vetor[index]:
        if int(i) == articles[position]:
            read = 1
            break
    if read == 1:
        position = position + 1
        checkReadBasedContent(users[0], index, articles, position)
    else:
        vetor[index].append(articles[position])
        print('The recommendation for the user', index, 'is the article'
    , articles[position])

    return

#Fun  o para recomendar o artigo baseado em conte do
def suggest_basedContent(users):
        for index, x in enumerate(users[0]):
            userArticles = otherArticles(x)
            checkReadBasedContent(users[0], index, userArticles, 0)
        return

Out(10 primeiros leitores):

('The recommendation for the user', 0, 'is the article', 13703)
('The recommendation for the user', 1, 'is the article', 9131)
('The recommendation for the user', 2, 'is the article', 6337)
('The recommendation for the user', 3, 'is the article', 803)
('The recommendation for the user', 4, 'is the article', 12480)
('The recommendation for the user', 5, 'is the article', 1920)
('The recommendation for the user', 6, 'is the article', 13504)
('The recommendation for the user', 7, 'is the article', 2209)
('The recommendation for the user', 8, 'is the article', 1408)
('The recommendation for the user', 9, 'is the article', 11265)
```

Listing 3: Content Based

5 Filtragem Colaborativa

A técnica de fatoração matricial é capaz de caracterizar tanto itens quanto usuários através de vetores de variáveis latentes inferidas de padrões de avaliação dos itens. Uma alta correspondência entre as variáveis latentes de um item e um usuário levam a uma recomendação[3].

Essa técnica está fortemente relacionada à decomposição de valores singulares (SVD, do inglês Singular Value Decomposition)

A idéia por trás de uma abordagem de SVD clássica é decompor uma matriz de avaliações P de dimensão em um produto de três matrizes,

$$P = A\Sigma B^T \tag{1}$$

onde A possui dimensão N x N, Σ possui dimensão NxM e B possui dimensão MxM. As matrizes A e B são ortogonais. Σ é uma matriz diagonal com k entradas diferentes de zero. Assim sendo, as dimensões efetivas das matrizes A, Σ e B são, NxK, KxK e KxM, respectivamente. Estas k entradas diagonais da matriz Σ são todas positivas com $\zeta_1 \geq \zeta_2 \geq ...\zeta_k \geq 0$. As colunas de A e B são, respectivamente, chamadas de autovetores a esquerda e a direita de P.

```
import pandas as pd
import numpy as np
from scipy.sparse.linalg import svds

users = pd.read_csv('users.dat',  header=None)
raw_data = pd.read_csv('raw-data.csv', encoding='iso-8859-1' );

article_df = pd.DataFrame(raw_data, columns = ['doc.id', 'raw.title', '
    raw.abstract'])

article_df = article_df.rename(index=str, columns={"doc.id": "ArticleID",
     "raw.title": "Title", "raw.abstract":"Abstract"})

users[0] = users[0].str.split(' ')
ratings = pd.read_csv('ratings.csv')

def fillRatings(ratings):
    columns = ['userId','articleId', 'rating']
    for index, user in enumerate(users[0]):
        for article in user:
            df = pd.DataFrame([[index, article ,5.0], [index, article
    ,5.0]], columns=columns)
            frames = [ratings, df]
            ratings = pd.concat(frames, ignore_index=True)
    return ratings

new_rate = fillRatings(ratings)
ax = new_rate.drop_duplicates()
ratings_df =  pd.read_csv('ratings.csv')

ratings_df = ratings_df.append(ax, ignore_index=True)
ratings_df=ratings_df.rename(columns = {'userId':'UserID', 'articleId':'
    ArticleID','rating':'Rating' })

article_df['ArticleID'] = article_df['ArticleID'].apply(pd.to_numeric)

R_df = ratings_df.pivot(index = 'UserID', columns ='ArticleID', values =
    'Rating').fillna(0)

R = R_df.as_matrix()
user_ratings_mean = np.mean(R, axis = 1)
R_demeaned = R - user_ratings_mean.reshape(-1, 1)

U, sigma, Vt = svds(R_demeaned, k = 50)
sigma = np.diag(sigma)
```

```

all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) +
    user_ratings_mean.reshape(-1, 1)
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = R_df.
    columns)

def recommend_article(predictions_df, userID, article_df,
    original_ratings_df, num_recommendations=1):

    # Pega e classifica as predi  es do usu rio
    user_row_number = userID - 1 # Assumindo que o ID do usu rio se
    inicia em 1
    sorted_user_predictions = predictions_df.iloc[user_row_number].
    sort_values(ascending=False)

    # Pega os dados do usu rio e mescla com as inform  es do artigo.
    user_data = original_ratings_df[original_ratings_df.UserID == (userID
    )]
    user_full = (user_data.merge(article_df, how = 'left', left_on = '
    ArticleID', right_on = 'ArticleID').
                    sort_values(['Rating'], ascending=False)
                )

    # Recomenda os artigos melhores avaliados e que n o foram lidos pelo
     usu rio
    recommendations = (article_df[~article_df['ArticleID'].isin(user_full
    ['ArticleID'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how =
     'left',
               left_on = 'ArticleID',
               right_on = 'ArticleID').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations, :-1]
                      )

    return user_full, recommendations

for index, user in enumerate(users[0]):
    already_rated, predictions = recommend_article(preds_df, index+1,
    article_df, ratings_df, 1)
    print('The recommendation for the user', index, 'is the article:',
    predictions)

Out (10 primeiros leitores):

The recommendation for the user 0 is the article:   ArticleID
                                                    Title  \
0         1  The metabolic world of Escherichia coli is not...

The recommendation for the user 1 is the article:   ArticleID
                                                    Title  \
0         1  The metabolic world of Escherichia coli is not...

The recommendation for the user 2 is the article:   ArticleID
                                                    Title  \
0         1  The metabolic world of Escherichia coli is not...

The recommendation for the user 3 is the article:   ArticleID
                                                    Title  \
0         1  The metabolic world of Escherichia coli is not...

The recommendation for the user 4 is the article:   ArticleID
                                                    Title  \
0         1  The metabolic world of Escherichia coli is not...
```

```
The recommendation for the user 5 is the article:    ArticleID
                                                  Title  \
0           1  The metabolic world of Escherichia coli is not...

The recommendation for the user 6 is the article:    ArticleID
                                                  Title  \
0           1  The metabolic world of Escherichia coli is not...

The recommendation for the user 7 is the article:    ArticleID
                                                  Title  \
0           1  The metabolic world of Escherichia coli is not...

The recommendation for the user 8 is the article:    ArticleID
                                                  Title  \
0           1  The metabolic world of Escherichia coli is not...

The recommendation for the user 9 is the article:    ArticleID
                                                  Title  \
0           1  The metabolic world of Escherichia coli is not...
```

Listing 4: Collaborative Filtering

6 References

[1] Beginners Guide to learn about Content Based Recommender Engines. *https* : *//www.analyticsvidhya.com/blog/*2015/08/*beginners − guide − learn − content − based − recommender − systems/*

[2] Content-based filterin. *http* : *//recommender − systems.org/content − based − filtering/*

[3] Role of Matrix Factorization Model in Collaborative. *https* : *//arxiv.org/pdf/*1503.07475*.pdf*

[4] Modelos de Fatoração Matricial.*https* : *//www.maxwell.vrac.puc − rio.br/*19273/19273$_4$*.pdf*