

Universidade do Minho - Escola de Engenharia
Scripting no Processamento de Linguagem Natural

Aplicação de sugestão de filmes

Autores :
Diogo Costa(A78034)



Marco Silva(A79607)



Patrícia Barreira (A79007)



Versão 1.0
31 de Janeiro de 2019

Resumo

O presente relatório visa documentar o desenvolvimento de uma aplicação de sugestão de filmes. Primeiro é apresentado a forma como foi obtido e tratado os textos dos filmes, ou seja, através de *scraping* e guardando esse dados numa estrutura em ficheiros pickle. Numa segunda fase é apresentado o algoritmo tf-idf desenvolvido, bem como o responsável por sugerir filmes com base num filme. No capítulo seguinte são apresentadas métricas da aplicação desenvolvida, isto é, precision, recall e F1 score. Por fim é apresentada a interface para concretização desta aplicação.

Conteúdo

1	Introdução	3
2	Scrapping	4
2.1	Extrair o guião completo	4
2.2	Cálculo do sentimento	6
2.3	Dicionário: <code>words_from_movies</code>	7
2.4	Dicionário: <code>genres_from_movies</code>	8
3	Utilização TF-IDF	9
4	Apresentação de Resultados	16
5	Métricas de Desempenho	18
5.1	Lista dos filmes disponíveis	18
5.2	Sugestões do TASTEDIVE	18
5.3	Cálculo dos valores para todos os filmes	19
5.4	Cálculo das métricas	20
6	Conclusões	22

1 Introdução

O projeto desenvolvido, e que se encontra descrito neste documento, tem como principal objetivo realizar sugestões de filmes. Efetivamente, o processo será baseado no algoritmo TF-IDF aplicado ao guião completo dos filmes. O *output* é um conjunto das palavras mais significativas de um dado filme, que depois é usado na expectativa de que, conjuntos com palavras semelhantes coincidam com filmes parecidos.

Dado que apenas o algoritmo TF-IDF pode não ser suficiente para realizar as melhores sugestões, o processo foi ainda complementado com a comparação dos géneros dos vários filmes de modo a desempatar alguns casos do processo de sugestão.

Primeiramente, foi necessário recolher dados para alimentar o motor de sugestões. Para o efeito foi usado o site [IMSDb](#). Deste recolheu-se todos os guiões disponíveis através da biblioteca [BeautifulSoup](#). A próxima secção descreve em detalhe todo este processo.

2 Scrapping

2.1 Extrair o guião completo

O projeto descrito ao longo deste documento tem como objetivo sugerir filmes com base na aplicação do algoritmo TF-IDF ao guião destes. Desta forma, o primeiro passo é colecionar guiões de filmes de forma a criar uma base de dados passível de ser utilizada.

Assim sendo, recorreu-se ao sítio IMSDb onde é possível de se encontrar guiões de filmes que podem, após algum processamento, ser usados para alimentar o algoritmo TF-IDF e, consequentemente, sugerir filmes similares. Após a definição da fonte de informação utilizou-se o **Beautiful Soup** (BS) para realizar o *parse* do html de forma a extrair a informação necessária de forma simples e eficiente.

Desta forma, avaliou-se como era construídos os URL's do *site* e percebeu-se que era possível percorrer todos os filmes através de um menu ordenado alfabeticamente.

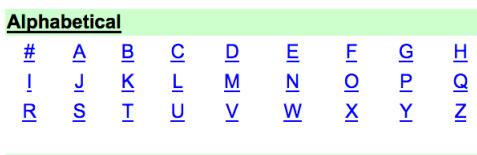


Figura 1: Pesquisa disponibilizado no site para o nome dos filmes

Sendo que cada uma das hiperligações da imagem resultavam na navegação para uma página com um formato fixo no URL.

```
https://www.imfdb.org/alphabetical/0  
https://www.imfdb.org/alphabetical/A  
https://www.imfdb.org/alphabetical/B  
...
```

Cada uma das páginas da respetiva letra do alfabeto tinham o seguinte formato:

A screenshot of the IMSDb website showing the 'A' page. The top navigation bar is red with the text "The Internet Movie Script Database (IMSDb)". Below it is a search bar with the placeholder "Search IMDB" and a "Go!" button. To the left is a sidebar with links for "Alphabetical" (with a grid of letters), "Genre" (Action, Adventure, Animation, etc.), "Sponsor" (Absolute Power), "TV Transcripts" (Ally McBeal), "International" (French scripts), and "Movie Software" (links to DVDRipper and WinX DVD Ripper). The main content area is titled "(A) Movie Scripts" and lists several movie titles with their descriptions and writers. Examples include "A Few Good Men" (1991-07 Revised draft) by Aaron Sorkin, "A Most Violent Year" (Undated Draft) by J.C. Chandor, and "A Scanner Darkly" (Undated Draft) by Charlie Kaufman.

Figura 2: Página com a lista dos filmes que começam por 'A'.

Na página de cada uma das letras encontra-se os *links* dos filmes iniciados por essa mesma letra. A título exemplificativo, ao aceder ao conteúdo do guião do filme *A Few Good Men*, percebe-se que existe um mapeamento direto entre o nome do *link* e o URL da página do guião do filme propriamente dito. Efetivamente, basta colocar um '-' entre as palavras que compõe o título do filme, prefixar o resultado com <https://www.imfdb.org/> e por fim concatenar com .html.

Assim sendo, está encontrado um mecanismo que permite de forma automática aceder a todos os URL's presentes no site IMSDb, o que permite iniciar a fase de *Scraping* propriamente dita. Desta forma, foi realizada uma função `get_movies_url` que está responsável por implementar em Python o algoritmo descrito a cima.

```
def get_movies_url():
    alphabet = "0ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    names_pages = []
    urls = []
    titles = []

    #reunião de todas as páginas de nomes de filmes
    for c in alphabet:
        print("Getting " + "https://www.imfdb.org/alphabetical/" + c)
        request = requests.get("https://www.imfdb.org/alphabetical/" + c)
        names_pages.append(request)

    for page in names_pages:
        page_soup = BeautifulSoup(page.text, "html.parser")
        results = page_soup.find_all('a')
        for result in results:
            if re.search(r'<p>', str(result.parent)) and
               'href' in result.attrs and 'title' in result.attrs:
                titles.append(result.text)
                title = result.text.replace(' ', '-')
                title = title.replace(':', '')
                url = 'https://www.imfdb.org/scripts/' + title + '.html' + '\n'
                urls.append(url.strip())
    return urls, titles
```

De uma forma mais detalhada, o primeiro ciclo é responsável por realizar um pedido GET, através da biblioteca Requests, a cada uma das páginas do menu do alfabeto. De seguida, o html é processado pelo BS, são procuradas todas as tags `a` em html, ou seja, todas as hiperligações que seja nomes de filmes. Foi verificado que tal acontecia quando eram precedidas por tags `p`, e tinham dois atributos, um `href` e um `title`. Assim sendo, quando estas condições estavam reunidas estavamo na presença do *link* necessário para compor o URL do guião do filme em questão. O resultado final é uma lista com os `urls` de todos os guões dos filmes disponibilizados no site IMSDb.

A fase seguinte passa por extrair (*scraping*) o guião presente num URL do conjunto processado anteriormente. Assim sendo, analisou-se o html das páginas dos diferentes guões e percebeu-se que o conteúdo útil se encontrava embutido numa tag `td` de html com um atributo `class` com um valor `scrtext`. Com base nesta informação construiu-se a função `scrap_full_script`:

```
def scrap_full_script(url):
    soup = BeautifulSoup(requests.get(url).text, "html.parser")
    full_script = soup.find_all('td', {'class': "scrtext"})[0].get_text()
    return clean_script(full_script)
```

De realçar que no final o texto é normalizado através da função `clean_script`, sendo que o resultado é uma string sem pontuação e com as palavras separadas por um único espaço.

A título exemplificativo, tem-se a seguinte transformação de uma porção do guião do Titanic:

My God.

His grip tightens on the diamond.

He looks up, meeting her gaze. Her eyes are suddenly infinitely wise and deep.

You look for treasures in the wrong place, Mr. Lovett. Only life is priceless, and making each day count.

His fingers relax. He opens them slowly. Gently she slips the diamond out of his hand. He feels it sliding away.

Then, with an impish little grin, Rose tosses the necklace over the rail. Lovett gives a strangled cry and rushes to the rail in time to see it hit the water and disappear forever.

my god his grip tightens on the diamond he looks up meeting her gaze her eyes are suddenly infinitely wise and deep you look for treasures in the wrong place mr lovett only life is priceless and making each day count his fingers relax he opens them slowly gently she slips the diamond out of his hand he feels it sliding away then with an impish little grin rose tosses the necklace over the rail lovett gives a strangled cry and rushes to the rail in time to see it hit the water and disappear forever

De facto, neste momento da execução tem-se acesso ao guião completo do filme já pronto a ser usado pelo algoritmo TF-IDF.

2.2 Cálculo do sentimento

Após o processamento do guião do filme revela-se possível calcular o respetivo sentimento. Para tal utilizou-se a biblioteca `nltk.sentiment.vader`. A utilização desta mostrou-se bastante fácil, bastando apenas inicializar o analisador e carregá-lo com o texto para que ele o analise.

```
nltk_sentiment = SentimentIntensityAnalyzer()  
score = nltk_sentiment.polarity_scores("texto que se quer analisar")
```

A pontuação dada pelo `polarity_scores` vem na forma de dicionário com as respetivas entradas:

- {'neg': [0.0..1.0]}: Representa a percentagem de **negatividade** presente no input recebido.
- {'neu': [0.0..1.0]}: Representa a percentagem de **neutralidade** presente no input recebido.
- {'pos': [0.0..1.0]}: Representa a percentagem de **negatividade** presente no input recebido.
- {'compound': [-1.0..1.0]}: Representa um **valor conjugado** de todos os outros.

De forma a perceber a evolução do sentimento ao longo do filme dividiu-se o guião do filme em 500 blocos de tamanho semelhante. Consequentemente, para cada um dos blocos foi calculado o sentimento respetivo e o valor correspondente à chave 'compound' foi acrescentado à lista `block_scores`.

```

for i in range(num_blocks-1):
    curr_block = full_script[i*block_size:(i+1)*block_size]
    block_score = nltk_sentiment.polarity_scores(curr_block)
    block_scores.append(block_score.get('compound'))
curr_block = full_script[i*block_size:]
block_score = nltk_sentiment.polarity_scores(curr_block)
block_scores.append(block_score.get('compound'))

```

Assim sendo, tem-se o que é necessário para construir um gráfico de forma a poder visualizar a variação do sentimento ao longo do filme. De facto, no eixo do *xx* tem-se o identificador de cada bloco do guião do filme. No eixo do *yy* encontram-se os valores dos sentimentos respetivos. Foi ainda adicionada um correlação linear entre os diferentes pontos de forma a permitir uma melhor percepção da evolução do sentimento ao longo do tempo.

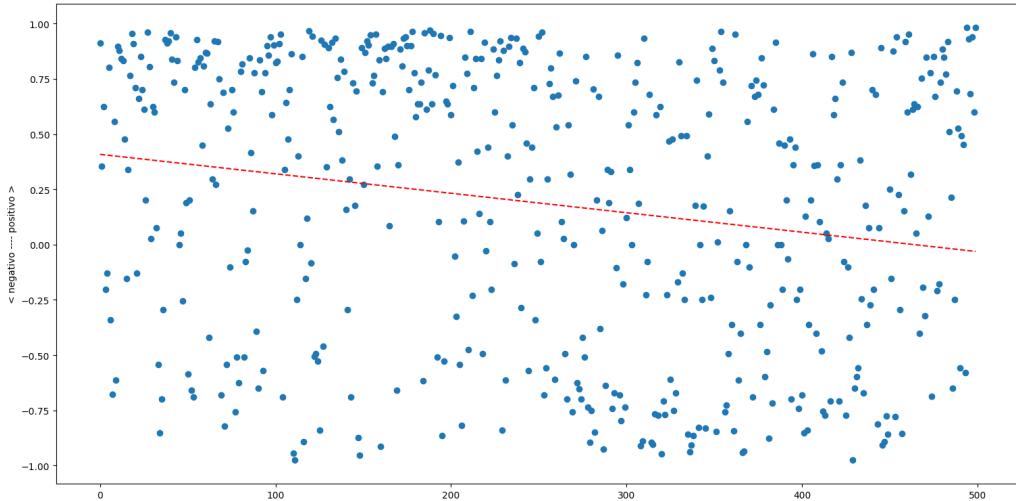


Figura 3: Gráfico com os resultados da análise de sentimentos ao filme Titanic.

2.3 Dicionário: words_from_movies

A estrutura de dados `words_from_movies` é um dicionário em que a chave é o título de um filme (em minúsculas com as palavras separadas por um único espaço) e o valor é uma string com o guião normalizado do filme [2.1]. Este dicionário é criado na função `save_full_scripts` e, consequentemente, guardado num ficheiro `pickle` quando terminado. Para utilizar esta funcionalidade basta correr o comando: `./scraping.py -f`.

```

# Guardar estrutura de dados num ficheiro pickle
def save_obj(obj, name):
    with open(name + '.pkl', 'wb') as fp:
        pickle.dump(obj, fp, pickle.HIGHEST_PROTOCOL)

def save_full_scripts(list_urls):
    words_from_movies = {}
    for i, url in enumerate(list_urls, start=1):

```

```

try:
    # Extração do guião completo
    full_script = scrap_full_script(url)
    (...)

    # Inserção no dicionário
    words_from_movies[title] = full_script
    (...)

except IndexError:
    (...)

print('Saving words_from_movies to file...')
save_obj(words_from_movies, 'dict_movies_list_words')

```

A gravação do pré-processamento dos guiões é guardada em ficheiro de forma a otimizar a utilização da ferramenta de sugestão de filmes. De facto, este algoritmo é bastante demorado devido ao número de filmes que é necessário processar (cerca de 1200) e desta forma torna-se quase imediata a sua utilização no próximo estágio da aplicação.

2.4 Dicionário: genres_from_movies

À semelhança da estrutura mencionada anteriormente também o `genres_from_movies` é um dicionário. Contudo, a chave é o título de um filme (em minúsculas com as palavras separadas por um único espaço) e o valor é uma lista dos géneros desse filme.

Primeiramente, foi necessário arranjar a lista dos géneros para um dado filme. Após alguma pesquisa percebeu-se que estes estavam disponíveis no próprio *site* do IMSDb.



Figura 4: Lista de géneros do filmes Titanic.

Seguidamente, analisou-se o HTML e constatou-se que a informação relevante se encontrava numa *tag table* com o atributo `class` e o valor `script_details`.

```
script_details = soup.find_all('table', {'class': "script-details"})[0]
```

Posteriormente, retirou-se todos os links presentes na tabela e extraiu-se aqueles que estavam relacionados com géneros, ou seja, os que continham a string "/genre/" no `href`.

```

(...)

links = script_details.find_all('a')
for link in links:
    href = link['href']
    match = re.match(r'/genre/(.*)', href)
    if match:

```

```

        if title not in genres_from_movies:
            genres_from_movies[title] = []
            genres_from_movies[title].append(match.group(1))

```

Por fim, após o dicionário estar preenchido guardou-se a estrutura num ficheiro pickle com o objetivo de otimizar a sua utilização. Para utilizar esta funcionalidade basta correr o comando: `./scraping.py -g`.

```

# Guardar estrutura de dados num ficheiro pickle
def save_obj(obj, name):
    with open(name + '.pkl', 'wb') as fp:
        pickle.dump(obj, fp, pickle.HIGHEST_PROTOCOL)

def save_genres(titles):
    (...)

print('Saving genres_from_movies to file...')
save_obj(genres_from_movies, 'dict_movies_list_genres')

```

De facto, esta estrutura de dados tem como objetivo melhorar a escolha dos filmes sugeridos.

3 Utilização TF-IDF

A sigla TF-IDF que significa *term frequency inverse document frequency*, representa um algoritmo que tem como objetivo extraer de um corpus as palavras mais importantes de cada documento. Para isso, há primeiro uma análise da frequência de cada palavra no corpus, e depois comparado essa frequência com a frequência da palavra no documento. Isto traduz-se em 3 cálculos:

- $TF(t) = (\text{Número de vezes que o termo } t \text{ aparece num documento}) / (\text{Número total de termos no documento})$
- $IDF(t) = \log_{10}(\text{Número total de documentos} / \text{Número de documentos que contêm o termo } t)$
- $TF-IDF(t) = TF(t) * IDF(t)$

Com o objetivo de perceber realmente como funciona o algoritmo, este foi desenvolvido passo a passo.

Primeiro é necessário ler os ficheiros pickles gerado pela primeira fase(scraping) para obter a informação dos filmes a analisar:

- dict_movies_list_words.pkl
- dict_movies_list_genres.pkl

Para a concretização do algoritmo foi necessário criar um dicionário com a ocorrência de cada palavra de cada filme. Isto é:

```

dict =
{'titulo_filme_1': {'palavra_1': #ocorrências , 'palavra_2': #ocorrências ...},
'titulo_filme_2': {'palavra_1': #ocorrências , 'palavra_2': #ocorrências ...},
...}

```

Para isso, utilizou-se o dicionário criado na leitura do ficheiro dict_movies_list_words.pkl, ou seja:

```

def buildWordCountDict(dict_movies):
    dict = {}
    for movie,words in dict_movies.items():
        dict[movie] = {}
        for word in words.split():
            if word in dict[movie]:
                dict[movie][word] = dict[movie][word] + 1
            else:
                dict[movie][word] = 1
    return dict

```

Assim, pode-se então realizar os cálculos necessários para concretizar o TF-IDF. Começando então pelo calculo do TF este será guardado num dicionário com o seguinte formato:

```

dict = {'titulo_filme_1': {'palavra_1': tf_value , 'palavra_2': tf_value ...},
        'titulo_filme_2': {'palavra_1': tf_value , 'palavra_2': tf_value ...},
        ...}

```

Para a concretização deste calculo precisa-se de ter o número ocorrência de cada palavra em cada filme e o número total de palavras do filme. Esta informação facilmente é obtida através do dicionário criado na função *buildWordCountDict*, isto é, basta aceder *dict[filme][palavra]* e *len(dict[filme])*, respetivamente. Isto traduz-se em código em:

```

def computeTF(wordDict):
    tfDict = {}
    for movie,words in wordDict.items():
        tfDict[movie] = {}
        totalMovieWords= len(wordDict[movie])
        for word,freq in words.items():
            tfDict[movie][word] = freq / totalMovieWords
    return tfDict

```

Agora, é necessário calcular IDF, e para isso é preciso obter o número total de filmes e o número de filmes que têm cada palavra. O primeiro valor facilmente é obtido através do dict criado na função *buildWordCountDict*, fazendo *len(dict)*. Já para o segundo valor a sua obtenção não é direta e foi necessário criar um dicionário que para cada palavra associa o número de filmes onde está aparecia. O algoritmo completo traduz-se em:

```

def computeIDF(wordDict):
    idfDict = {}
    totalMovies= len(wordDict)
    wordOnDocuments = {}

    #calculo para cada palavra, o número de filmes onde está aparece
    for movie, words in wordDict.items():
        for word in words:
            if word in wordOnDocuments:
                wordOnDocuments[word] = wordOnDocuments[word] + 1
            else:
                wordOnDocuments[word] = 1

```

```

#calculo do valor IDF
for movie,words in wordDict.items():
    idfDict[movie] = {}
    for word in words:
        idfDict[movie][word] = math.log10(totalMovies / wordOnDocuments[word])
return idfDict

```

O resultado será também guardado num dicionário com o seguinte formato:

```

dict = {'titulo_filme_1': {'palavra_1': idf_value, 'palavra_2': idf_value ...},
        'titulo_filme_2': {'palavra_1': idf_value, 'palavra_2': idf_value ...},
        ...}

```

Neste momento tem-se então o valor tf e idf para cada palavra de cada filme. Para concretização do passo final, basta então fazer a multiplicação dos dois valores de cada dicionário.

```

def computeTFIDF(wordDict, tfDict, idfDict):
    tfidfDict = {}
    for movie, words in wordDict.items():
        tfidfDict[movie] = {}
        for word in words:
            tfidfDict[movie][word] = tfDict[movie][word] * idfDict[movie][word]
    return tfidfDict

```

O resultado será também guardado num dicionário com o seguinte formato:

```

dict =
{'titulo_filme_1': {'palavra_1': tf-idf_value, 'palavra_2': tf-idf_value ...},
 'titulo_filme_2': {'palavra_1': tf-idf_value, 'palavra_2': tf-idf_value ...},
 ...}

```

Juntando todos os passos numa única função que retorna o dicionário com o valor tf-idf para cada palavra de cada filme, temos então:

```

def buildTFIDF(dict_movies):
    wordDict = buildWordCountDict(dict_movies)
    tfDict = computeTF(wordDict)
    idfDict = computeIDF(wordDict)
    tfidfDict = computeTFIDF(wordDict, tfDict, idfDict)
    return tfidfDict

```

Tendo então calculado tf-idf para cada palavra de cada filme, agora é necessário aceder rapidamente às que possuem um maior valor de tf-idf de cada filme. Portanto foi criado um dicionário ordenado por ordem decrescente pelo valor tf-idf de cada palavra. Terá o seguinte formato:

```

dict =
{'titulo_filme_1': [('palavra_1', tf-idf_value), ('palavra_2', tf-idf_value), ...],
 'titulo_filme_2': [('palavra_1', tf-idf_value), ('palavra_2', tf-idf_value), ...],
 ...}

```

Assim facilmente se consegue extrair as palavras mais importante de cada filme. Por exemplo, caso se pretenda obter as 25 palavras mais importantes, basta extrair os 25 primeiros tuplos da lista de cada filme. Foi tomada a decisão de guardar este dicionário em ficheiro pickle para que seja mais eficiente a utilização da aplicação, não sendo necessário estar sempre a calcular este dicionário a cada pesquisa.

Resta então explicar o algoritmo usado para fazer match do filme pesquisado com os filmes que serão dados como sugeridos.

Fixado um número para o top de palavras mais importantes de cada filme, por exemplo 10, o primeiro passo será então, para cada filme comparar o seu top de palavras mais importantes com o top do filme que foi pesquisado. O número de palavras que fizeram match nesta comparação é guardado numa lista de tuplos com o seguinte formato:

```
suggestFilms =
[('titulo_filme_1', #palavras_iguais_entreEste_filme_e_pesquisado), ...]
```

De seguida esta lista é ordenada de forma decrescente de acordo com o segundo valor do tuplo. Este ordenação permite ter à cabeça os filmes com maior número de matches de palavras do top comparando com o top do filme pesquisado. Por exemplo, caso seja pesquisado o filme *Batman* e o top inclua as 100 palavras do filme com maior valor tf-idf, teremos o seguinte output:

```
[('batman 2', 30), ('the dark knight rises', 20),
('angel eyes', 8), ('the cincinnati kid', 8),
('the fabulous baker boys', 8), ('hardrain', 8), ('limitless', 8),
('se7en', 8), ('under fire', 8), ('bruce almighty', 7)]
```

Caso o top seja composto por 25 palavras, teremos então:

```
[('batman 2', 10), ('the dark knight rises', 5), ('bruce almighty', 3),
('ace ventura pet detective', 2), ('affliction', 2),
('croupier', 2), ('detroit rock city', 2), ('king kong', 2),
('legend', 2), ('natural born killers', 2)]
```

Isto diz que, analisando o primeiro resultado, em 100 palavras com o maior valor de tf-idf de cada filme, 30 fazem match entre o filme Batman e Batman2, 20 fazem match entre *The Dark Knight Rises* e assim sucessivamente. Para o segundo output, o mesmo princípio, num top formado por 25 palavras, 10 fazem match entre o *Batman* e *batman2*, 5 fazem match entre *Batman* e *The Dark Knight Rises*, e assim sucessivamente.

Com vista a melhorar as sugestões, os resultados que tenham o mesmo match será analisado o género do filme. Isto é, se houver algum filme que esteja fora do top 10, mas que tenha o mesmo número de match e o género do filme esteja mais de acordo com o filme que foi pesquisado, então é feita a troca. Assim, está a análise é feita tendo por base um conjunto de alternativas com o mesmo valor do match e esse conjunto é dado pelos restantes valores(os restantes filmes que não foram sugeridos) da lista criada anteriormente, *suggestFilms*.

No exemplo do *Batman*, estão a ser sugeridos 10 filmes. O objetivo então é, para os filmes com o mesmo match e que tenham alternativas, verificar-se se as alternativas são melhores de acordo com o género. Este conjunto de alternativas, só segue para o menor match dos filmes sugeridos, que no caso do *Batman* com 100 palavras, é o 7 e com 25 é o 2. Iremos analisar o 25, uma vez que pode haver mais que uma substituição. O que é feito a seguir, é calcular o géneros dos filmes que estão na lista que têm o match igual a 2. Depois é comparado o género desses filmes, com o género do filme pesquisado, e ordenado segundo o número de matches de forma decrescente. A seguir é feito o teste para verificar se é possível fazer então a substituição. Este teste é composto por:

- Existir alternativas para efetuar a troca, ou seja, existir mais filmes com o match igual a 2 para além dos sugeridos
- O match entre géneros tem que ser maior na alternativa do que no filme que escolhido como sugerido.

Com exemplo do *Batman*, temos então:

```
Géneros do filme Batman: Action, Crime, Fantasy, Thriller
Géneros dos filmes alternativos com o mesmo match do filme pesquisado:
- Next Friday: Comedy
- The Woodsman: Drama

Filmes sugeridos:
- Batman 2
- The Dark Knight Rises
- Bruce Almighty
- Ace Ventura Pet Detective
- Affliction
- Croupier
- Detroit Rock City
- King Kong
- Natural Born Killers
```

Pelo resultados apresentados em cima verificamos que, há dois filmes que têm o match igual a 2 e portanto foi calculo o género desses filmes. De seguida foi feita a comparação desse género com o género do filme pesquisado. Verificou-se então que não havia nenhuma correspondência. Desta forma, os filmes que estavam na sugestão inicial são mantidos, uma vez que apresentam melhores resultados(maior tf-idf).

Apresenta-se de seguida uma situação de troca, para a pesquisa do filme *Death at a Funeral*:

```
Filmes sugeridos antes de tratar do género:
[('hesher', 14), ('after.life', 13), ('ted', 12),
('12 and holding', 11), ('american beauty', 10), ('angel eyes', 10),
('the anniversary party', 10), ('as good as it gets', 10),
('blue velvet', 10), ('the limey', 10)]
```

```
Géneros do filme pesquisado: Comedy
Géneros dos filmes alternativos com o mesmo match do filme pesquisado:
- Mini's First Time: Comedy, Crime, Drama
- Martha Marcy May Marlene: Drama, Thriller
- Obsessed: Drama, Thriller
- The Perks of Being a Wallflower: Drama, Romance
- Rear Window: Thriller, Mystery
- A Walk to Remember: Drama, Romance
- What Lies Beneath: Horror.Mystery, Thriller, Drama
```

```
Filmes sugeridos:
- Hesher
- After.life
- Ted
- 12 and holding
- Mini's First Time
- Angel Eyes
```

- The Anniversary Party'
- As Good As It Gets
- Blue Velvet
- The Limey

Neste exemplo, existiam 7 filmes que apresentavam o mesmo match no tf-idf(10) para além dos apresentados nas sugestões. O primeiro filme com este match, é *American Beauty* e o seu género é Drama. Uma vez que o filme pesquisado é de comédia, e há uma alternativa que tem como género, é feita a troca do filme *American Beauty* por *Mini's First Time*. Como não há mais nenhum filme alternativo que tenha como género comédia, não é realizada mais nenhuma troca.

Outro exemplo, para o filme *Wanted* temos os seguintes resultados:

Filmes sugeridos antes de tratar do género:

```
[('priest', 10), ('hollow man', 9),
('nightmare on elm street the final chapter', 9),
('hard rain', 8), ('the mechanic', 8), ('rise of the guardians', 8),
('a most violent year', 7), ('blood simple', 7), ('bodyguard', 7),
('chronicle', 7)]
```

Géneros do filme pesquisado: Action, Crime, Thriller

Géneros dos filmes alternativos com o mesmo match do filme pesquisado:

- Hancock: Action, Comedy, Crime, Drama, Fantasy, Thriller
- Ronin: Action, Crime, Thriller
- The King of Comedy: Comedy, Crime, Drama
- Limitless: Mystery, Sci-Fi, Thriller
- Memento: Drama, Mystery, Thriller
- Cinema Paradiso: Drama, Romance
- Garden State: Comedy, Drama, Romance
- Hesher: Drama
- Starman: Adventure, Drama, Romance, Sci-Fi

Filmes sugeridos:

- Priest
- Hollow Man
- Nightmare on Elm Street The Final Chapter
- Hard Rain
- The Mechanic
- Rise of the Gardian
- Hancock
- Ronin
- Bodyguard
- Chronicle

Neste exemplo, o menor match é 7 e surgem 9 alternativas possíveis com o mesmo match de tf-idf. Tendo em conta o género, verifica-se que *Hancock* é constituído pelos 3 géneros do filme pesquisado, enquanto que *A Most Violent Year* é constituído por Crime, Drama, Thriller. Consequentemente *Hancock* apresenta melhores resultados a nível de género, sendo então substituído. A próxima alternativa disponível, *Ronin* tem match total de géneros, enquanto que o próximo filme dos sugeridos, *Blood Simple* tem 2 matches em 3(Crime, Drama, Thriller), sendo assim substituído. A próxima alternativa disponível, *The King of Comedy* tem Crime em comum com o filme pesquisado e o próximo filme sugerido, *Bodyguard* tem como género Drama, Romance e Thriller. Ou seja, um em comum tal como o filme alternativo. Portanto não é feita a substituição. O ultimo filme a verificar é *Chronicle* que tem como géneros, drama, sci-fi e thriller. Tendo então um em

comum como o filme pesquisado, não será substituído porque os filmes alternativos apresentam também apenas 1 gênero em comum.

O algoritmo final do match é então:

```

def match_count(mostImportantWords,movieWords):
    mostImportantWordsFirstTuple= [t[0] for t in mostImportantWords]
    movieWordsFirstTuple = [t[0] for t in movieWords]
    equalElem = set(mostImportantWordsFirstTuple) & set(movieWordsFirstTuple)
    return len(equalElem)

def match_genre_count(suggestFilms,lower_value,movieRequest):
    match = []
    for t in suggestFilms[nFilms:]:
        if t[1]!=lower_value:
            break
        else:
            match.append((t[0],
                          len(set(genresDict[movieRequest])
                               & set(genresDict[t[0]]))))
    match = sorted(match, key=lambda tup: tup[1], reverse = True)
    return match

def genres_ok(suggestFilms, movieRequest):
    lower_value = suggestFilms[nFilms][1]
    match_genre = match_genre_count(suggestFilms,lower_value,movieRequest)
    for n,t in enumerate(suggestFilms[:nFilms]):
        if t[1]==lower_value:
            if len(match_genre)>0 and match_genre[0][1]>0 and
               len(set(genresDict[t[0]])) &
               set(genresDict[movieRequest])) < match_genre[0][1]:
                suggestFilms[n]=match_genre[0]
                del match_genre[0]
    return suggestFilms

def match(movieRequest):
    suggestFilms = []
    if movieRequest in orderDict:
        mostImportantWords= orderDict[movieRequest] [:topWords]
        for movie, words in orderDict.items():
            if movieRequest != movie :
                movieWords = words[:topWords]
                suggestFilms.append((movie,
                                     match_count(mostImportantWords,movieWords)))
    suggestFilms = sorted(suggestFilms,
                           key=lambda tup: tup[1],
                           reverse = True)
    suggestFilms = genres_ok(suggestFilms,movieRequest)
    suggestFilmsGenres = []
    for movie in suggestFilms[:nFilms]:
        suggestFilmsGenres.append((movie[0],genresDict[movie[0]]))
    return suggestFilmsGenres
else :
    return []

```

4 Apresentação de Resultados

Uma vez que, os resultados a apresentar envolvem não só texto mas também imagens, implementou-se então um pequeno *website* que tratará não só de apresentar os resultados mas também da recolha do *input* do utilizador e invocação dos métodos necessários.

Para a implementação do *website* escolheu-se a linguagem *python* uma vez que através da biblioteca *flask* é possível implementar de uma forma fácil e rápida um *website*. Para além disso, por forma a facilitar a criação das páginas *html* foi utilizada uma linguagem de template chamada *Jinja2*. Com esta linguagem é possível assim não só inserir dados dinamicamente mas também implementar tanto verificação de condições como ciclos diretamente no ficheiro *html*.

Relativamente ao *website*, este encontra-se organizado em duas páginas principais, a página principal e a página de apresentação de resultados.

Para a página principal, desenvolveu-se inicialmente uma caixa de pesquisa para que o utilizador pudesse introduzir o nome do filme sobre o qual pretendia obter sugestões. Para além disso, mais abaixo existe ainda uma lista dos filmes para os quais é suportada a sugestão de filmes. Esta limitação surge da limitação do próprio *website* consultado para análise de filmes.

Assim, a pesquisa tanto pode ser feita através da caixa de pesquisa como através da lista de títulos que contêm hiperligações para as respetivas pesquisas.

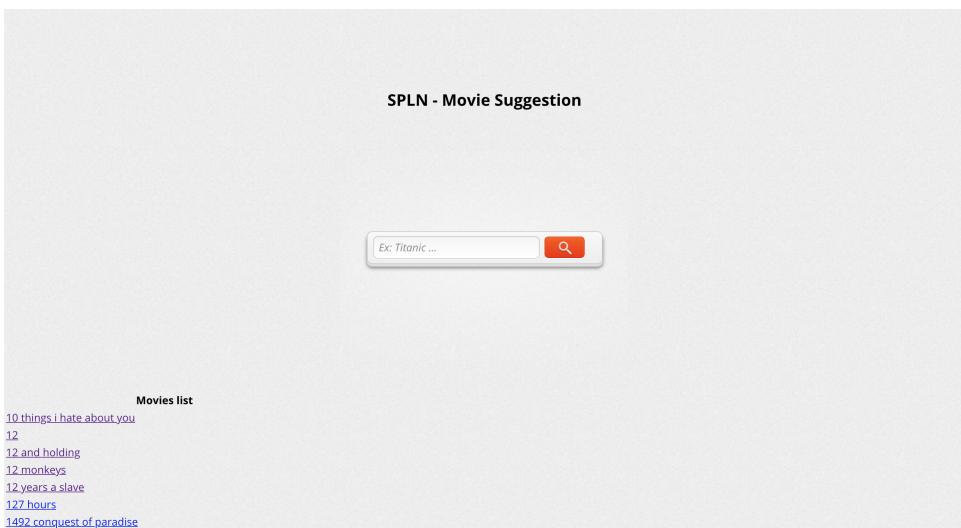


Figura 5: Página principal do *website* de sugestões.

```
@app.route('/')
def search():
    return render_template('index.html', movies = orderDict)
```

Aqui pode-se observar a passagem da informação dos filmes disponíveis para sugestões à função *render_template* de modo a que a página seja construída dinamicamente.

Para a página de apresentação do filme, são mostrados apenas alguns elementos característicos do filme, título, capa e a lista de filmes sugeridos. Foi adicionada também uma barra no topo que permite uma navegação fácil para a página inicial após uma pesquisa. É dada a possibilidade observação em maior pormenor da imagem da análise do sentimento da sugestão carregando em cima da imagem a observar.

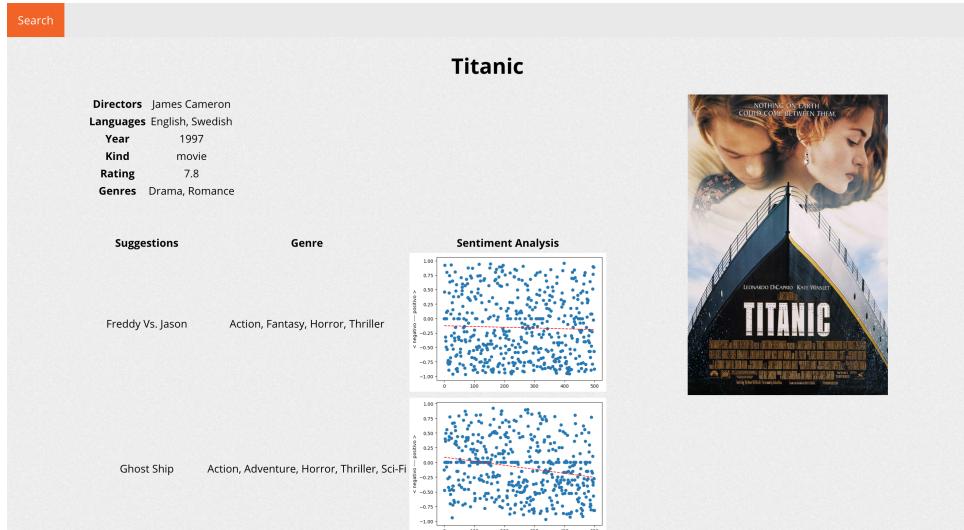


Figura 6: Página de apresentação das informações de um filme.

Pode-se observar aqui, os dois modos de pesquisa, cada um com um método distinto de passagem do título do filme a pesquisar do *html* para o *python*.

```
@app.route('/movie/<id>')
def movie(id):
    movie_name_original = id
    html_info = {}

    movies = IMDb_access.search_movie(movie_name_original)
    (...)

    return render_template('movie.html', movie_name = movie_name,
                           html_infos = html_info, cover_url = cover_url,
                           suggestions = suggestions, suggestion_error = suggestion_error)
```

Finalmente, para obter alguma informação sobre o filme a pesquisar, recorreu-se a uma biblioteca chamada *imdbPY* que permite obter inúmeras informações sobre um determinado filme. Veja-se então abaixo um exemplo de utilização desta biblioteca.

```
movies = IMDb_access.search_movie(movie_name_original)

if movies != []:
    movie_infos = IMDb_access.get_movie(movies[0].getID())

    directors_names = []
    try:
        directors = movie_infos['directors']
        for director in directors:
            directors_names.append(director['name'])
    except:
        directors_names = []

    html_info['Directors'] = ', '.join(directors_names)

    try:
```

```

        html_info['Languages'] = ', '.join(movie_infos['languages'])
    except:
        html_info['Languages'] = ''

    try:
        html_info['Year'] = movie_infos['year']
    except:
        html_info['Year'] = ''

```

5 Métricas de Desempenho

5.1 Lista dos filmes disponíveis

As métricas de desempenho desenvolvidas têm como objetivo avaliar se as sugestões realizadas fazem sentido quando comparadas com outros motores disponíveis. Neste caso, utilizou-se o site TASTEDIVE para realizar a comparação, visto este ser um dos motores mais populares para o efeito.

A primeira fase envolveu conseguir a lista de todos os filmes a que a aplicação desenvolvida tinha acesso para que as sugestões de cada um pudessem ser avaliadas. De facto, dado que é necessário a existência do ficheiro `dict_movies_list_words.pkl` para a aplicação funcionar, carregou-se este ficheiro e extraiu-se a lista das chaves do dicionário, uma vez que estas representam os títulos de todos os filmes disponíveis.

```

def load_available_movies():
    try:
        words_from_movies = load_obj('dict_movies_list_words')
        return list(words_from_movies.keys())
    except:
        print('ERROR - First use scrapping.py')
        exit(1)

```

5.2 Sugestões do TASTEDIVE

De seguida, era necessário preparar o mecanismo para que se conseguisse obter as sugestões por parte do TASTEDIVE. Após alguma investigação percebeu-se que o site disponibiliza uma API que permite de forma simplificada realizar *queries* ao motor de sugestões. Desta forma, a sintaxe utilizada nas interrogações é a seguinte:

```

https://tastedive.com/api/similar?q="movie:<nome_do_filme>"&
                                type=movies&
                                limit=10&
                                k=<access_key>

```

De uma forma mais detalhada, os vários componentes do URL tem o seguinte significado:

- `q="movie:<nome_do_filme>"`: pesquisa na categoria filmes pelo `<nome_do_filme>`.
- `type=movies`: indica que se pretende que as sugestões sejam apenas filmes.
- `limit=10`: o resultado deve ser de no máximo 10 filmes.
- `k=<access_key>`: a chave de acesso à API.

A título exemplificativo, quando é realizada um interrogação para o filme Titanic o resultado obtido é o seguinte:

```
{'Similar': {'Info': [{"Name': 'Titanic', 'Type': 'movie'}], 'Results': [{"Name': 'Home Alone', 'Type': 'movie'}, {"Name": "Boo", "Type": "movie"}, {"Name": "Ajab Prem Ki Ghazab Kahani", "Type": "movie"}, {"Name": "Ghost", "Type": "movie"}, {"Name": "2012", "Type": "movie"}, {"Name": "Breaking Dawn", "Type": "movie"}, {"Name": "Alvin And The Chipmunks", "Type": "movie"}, {"Name": "Twilight", "Type": "movie"}, {"Name": "The Smurfs", "Type": "movie"}, {"Name": "Baby's Day Out", "Type": "movie"}]}}
```

De facto, como é possível visualizar no exemplo a cima, o resultado produzido pela API vem no formato JSON. Assim sendo, utilizou-se a biblioteca em Python `json` para realizar o parse através da função `loads` e, consequentemente, adicionar o título do filme à lista dos filmes sugeridos `suggested_movies`.

```
td_data = json.loads(request.text)
for movie_entry in td_data['Similar']['Results']:
    suggested_movies.append(movie_entry['Name'].lower())
```

Desta forma, toda a funcionalidade descrita em cima é implementada através da função `tastedive_suggested` e permite obter a lista dos filmes sugeridos pelo TASTEDIVE.

5.3 Cálculo dos valores para todos os filmes

Após a realização dos mecanismos que permitem ter os filmes disponíveis e forma de obter as sugestões do TASTEDIVE, basta comparar os resultados.

```
def test_suggest_engine_precision(movies):
    for movie in movies:
        control = tastedive_suggested(movie) # Sugestões do TASTEDIVE
        testing = [m[0] for m in match(movie)] # Sugestões do projeto
        if control and testing:
            update_values(control, testing) # Comparaçao
```

Efetivamente, da comparação dos resultados são extraídas três valores: `true_positive`, `false_positive` e `false_negative`. Estes serão usados para calcular a `precision`, `recall` e `f1_score`. Neste contexto os valores extraídos têm o seguinte significado:

- `true_positive`: simboliza o número de filmes que foram sugeridos por ambas as ferramentas.
- `false_positive`: representa o número de filmes que foram apenas sugeridos pela ferramenta a ser desenvolvida.
- `false_negative`: representa o número de filmes que foram sugeridos apenas pelo site TASTEDIVE.

Na prática a função que calcula estas variáveis chama-se `update_values`.

```

def update_values(control, testing):
    now_true_positive = calc_now_true_positive(control, testing)
    TRUE_POSITIVE += now_true_positive
    FALSE_NEGATIVE += len(control) - now_true_positive
    FALSE_POSITIVE += len(testing) - now_true_positive

```

Contudo, o cálculo do `true_positive`, isto é, a comparação dos resultados não se revelou tarefa fácil uma vez que o site IMSDb e o TASTEDIVE muitas vezes diferem no nome que dão ao mesmo filme. Por exemplo, para o filme "The Lord of the Rings: The Return of the King" os sites identificam o filme da seguinte forma:

- IMSDb: "Lord of the Rings: Return of the King"
- TASTEDIVE: "The Lord Of The Rings: The Return Of The King"

Com o objetivo de minimizar estas discrepâncias utilizou-se a função `similar` que retorna uma percentagem de parecência.

```

def similar(a, b):
    return SequenceMatcher(None, a, b).ratio()

```

Assim sendo, quando a função é chamada com os filme do senhor dos anéis obtem-se o seguinte resultado:

```

>>> similar("The Lord Of The Rings: The Return Of The King",
           "Lord of the Rings: Return of the King")
0.8048780487804879

```

Desta forma, colocou-se o `threshold` a 0.8 de forma a minimizar o risco de assinalar os filmes como diferentes quando estes são iguais. No fim deste processo tem-se os valores do `true_positive`, `false_positive` e `false_negative` prontos a serem usados para calcular as métricas propriamente ditas.

5.4 Cálculo das métricas

O cálculo das métricas revelou-se bastante direto uma vez que já se tinha os valores para realizar as operações. Foram feitos 4 testes, variando o número de palavras que constituem o top do tf-idf. Foi realizado o calculo para 25,50,75 e 100 palavras.

Para 25:

$$precision = \frac{true_positive}{true_positive + false_positive} = \frac{189}{189 + 9181} = 0.0201$$

$$recall = \frac{true_positive}{true_positive + false_negative} = \frac{189}{189 + 8995} = 0.0205$$

$$f1_score = 2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.201 * 0.0205}{0.0201 + 0.0205} = 0.0203$$

Para 75:

$$precision = \frac{252}{252 + 9028} = 0.02715$$

$$recall = \frac{252}{252 + 8752} = 0.02798$$

$$f1_score = 2 * \frac{0.02715 * 0.02798}{0.02715 + 0.02798} = 0.02756$$

Para 100:

$$precision = \frac{280}{280 + 9080} = 0.0299$$

$$recall = \frac{280}{280 + 8884} = 0.0305$$

$$f1_score = 2 * \frac{0.0299 * 0.0305}{0.0299 + 0.0305} = 0.0302$$

Foi escolhido usar o 100, uma vez que é o que apresenta melhores resultados. Importa referir que, apesar de terem sido calculadas as métricas em cima mencionadas, as duas últimas perdem alguma importância uma vez que tem em conta os **false_negative**, ou seja, os filmes sugeridos pelo TASTEDIVE que não o foram pelo sistema de sugestões desenvolvido. Efetivamente, dado que a base de dados do site TASTEDIVE é muito maior que a que foi usada do site IMSDb, torna-se impossível sugerir alguns dos filmes, pois simplesmente estes não existem.

Por fim, após analisar os resultados obtidos estes revelaram-se bastante baixos. Por um lado, devido ao facto de a comparação de igualdade entre filmes ser baseada no título dos mesmos, pois sites dão títulos ligeiramente diferentes ao mesmo filme. Por outro lado, a discrepância pode estar ligada à mentalidade com que as próprias sugestões são feitas, pois ao usar o algoritmo TF-IDF está-se a dar bastante prioridade ao conteúdo do próprio filme, o que pode não se verificar de forma tão demarcada nas escolhas realizadas pelo site de referência.

6 Conclusões

Com a realização deste trabalho prático, foi de facto possível perceber o elevado nível de complexidade envolvido na sugestão de filmes apenas com base nos seus diálogos. Apesar dos resultados relativamente baixos de precisão relativamente a outras ferramentas de sugestão de filmes, os resultados obtidos mostraram-se adequados, não sendo completamente descabidos.

Por forma a melhorar esta precisão, sugere-se a utilização da sinopse também como parte da caracterização de um filme uma vez que este condensa de uma forma mais objetiva o assunto do filme, o que não acontece com a versão completa, podendo assim levar a sugestões erradas. Desta forma, sugere-se a complementação do algoritmo *tf-idf* com a análise de descrições mais condensadas do filme em questão bem como a consideração de análises feitas por outros utilizadores. Uma vez que o grande foco seria a utilização do algoritmo *tf-idf*, foi dada a este o maior peso na sugestão mas, seria interessante o teste de outras combinações dos elementos característicos referidos anteriormente.