

Progetto "BookSuggestor"

Nome: Salvatore Alfio

Cognome: Sambataro

Matricola: 1000015834

Corso: Social Media Management

Docente: Antonino Furnari

Anno accademico: 2022/2023

Introduzione

Il progetto "BookSuggestor" consiste nella realizzazione di un sistema di raccomandazione che permette all'utente di ottenere consigli su libri da leggere, sulla base di un insieme di parole chiave fornite in input al sistema. Più nello specifico, il funzionamento del sistema è basato sull'utilizzo di un apposito dataset, che contiene diverse informazioni, ad esempio il titolo, la trama o l'autore, riguardo un'ampia varietà di libri.

Un altro obiettivo del progetto è quello di confrontare il funzionamento del sistema utilizzando due diverse tipologie di rappresentazione per il testo: "**Bag of Words**" e "**Word Embeddings**".

Infine, i risultati ottenuti dalle due varianti dell'algoritmo saranno analizzati e confrontati, usando la metrica di valutazione "**Mean Reciprocal Rank**", e saranno tratte le dovute conclusioni.

Dataset utilizzato

Al fine di mettere a disposizione degli utenti un sistema di raccomandazione che tenga conto di un numero quanto più alto possibile di libri, si è deciso di utilizzare un dataset già pronto.

Nello specifico, il dataset utilizzato è il dataset open-source "[CMU Book Summary](#) (<https://www.kaggle.com/datasets/ymaricar/cmu-book-summary-dataset>)" (fonte: [kaggle](#) (<https://www.kaggle.com/datasets>)).

Più nello specifico, per ogni libro si hanno a disposizione:

- Wikipedia ID
- Freebase ID
- Titolo
- Autore
- Data di pubblicazione
- Generi
- Riassunto della trama

Preprocessing dei dati

Il sistema raccomanda all'utente un certo insieme di libri sulla base della somiglianza tra le parole chiave inserite dall'utente e il contenuto effettivo del libro, in termini di trama.

Per il corretto funzionamento del sistema è necessario inizialmente effettuare una fase di preprocessing dei dati presenti all'interno del dataset. Nello specifico, le operazioni da svolgere sono:

- caricare i record presenti nel file in un'apposita struttura dati

- rimuovere eventuali record di libri per cui si hanno solo informazioni parziali (ad esempio, manca la trama, il titolo o l'autore)

Tra le librerie utilizzate per tali operazioni abbiamo:

- pandas**: utile per la realizzazione di strutture dati apposite per la memorizzazione dei dati raccolti
- numpy**: fornisce la possibilità di usare diverse tipologie di funzioni matematiche

In [1]:

```
import pandas as pd
import numpy as np

books = pd.read_csv("booksummaries.txt",
                     header=None, sep="\t",
                     names=["Wikipedia ID", "Freebase ID", "Title", "Author", "Pub date", "Genres"]

books = books.dropna(subset=['Title', 'Author', 'Summary', 'Genres'])
books = books.reset_index(drop=True)

print("Numero di libri disponibili: ", len(books))
```

Numero di libri disponibili: 12055

Una volta effettuato il preprocessing dei dati, le informazioni su ognuno dei libri presenti nel dataset sono memorizzate in un apposito dataframe, in cui ognuna delle righe ha la seguente forma:

In [2]:

```
books.head(n=1)
```

Out[2]:

	Wikipedia ID	Freebase ID	Title	Author	Pub date	Genres	Summary
0	620	/m/0hhv	Animal Farm	George Orwell	1945-08-17	{""/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":...}	Old Major, the old boar on the Manor Farm, ca...

Come è possibile osservare dal record di esempio, è necessario effettuare un'ulteriore pulizia dei dati riguardanti il genere dei libri.

Notiamo che i generi sono memorizzati sotto forma di **coppie "chiave-valore"** : l'estrazione dei soli nomi dei generi può essere effettuata grazie all'utilizzo della libreria "JSON".

In [3]:

```
import json

dictionary = json.loads(books.iloc[0]['Genres'])
print(dictionary, "\n")

genres = []
for key in dictionary:
    genres.append(dictionary[key])

print(genres)

{'/m/016lj8': 'Roman à clef', '/m/06nbt': 'Satire', '/m/0dwly': "Children's literature", '/m/014dfn': 'Speculative fiction', '/m/02xlf': 'Fiction'}

['Roman à clef', 'Satire', "Children's literature", 'Speculative fiction', 'Fiction']
```

Ripetiamo l'operazione appena vista per ognuno dei libri presenti nel dataframe:

In [4]:

```
genres = []

for genreList in books['Genres']:
    dictionary = json.loads(genreList)
    lista = []
    for key in dictionary:
        lista.append(dictionary[key])
    genres.append(",".join(lista))

books['Genres'] = genres
```

Adesso ognuno dei record avrà nel campo "Genres" solo i nomi dei generi di appartenenza:

In [5]:

```
books.iloc[0]["Genres"]
```

Out[5]:

```
"Roman à clef,Satire,Children's literature,Speculative fiction,Fiction"
```

Algoritmi utilizzati

Per poter implementare la ricerca di un libro sulla base di specifiche parole chiave, è necessario utilizzare un'apposita funzione di rappresentazione per il testo.

Le rappresentazioni che saranno utilizzate per questo progetto sono due:

- **Bag of Words**
- **Word Embeddings**

Rappresentazione Bag of Words

La rappresentazione "**Bag of Words**" è una rappresentazione che permette di rappresentare un testo sulla base dell'insieme delle parole che sono in esso contenute. Lo svantaggio principale è che essa tiene conto esclusivamente del numero di occorrenze di ogni parola all'interno del testo in questione, mentre non si considera il significato e il contesto della singola parola.

Calcolo della rappresentazione Bag of Words tramite la libreria "scikit-learn"

Per calcolare il vettore Bag of Words di un testo, si è scelto di fare uso della libreria "**scikit-learn**", la quale mette a disposizione un apposito modulo chiamato "**CountVectorizer**" che permette di calcolare la rappresentazione Bag of Words sulla base di un insieme di testi, detto "**corpus di documenti**".

La prima cosa da fare è importare il modulo "CountVectorizer" e creare un oggetto di tipo "CountVectorizer" con l'apposita funzione:

In [6]:

```
from sklearn.feature_extraction.text import CountVectorizer  
count_vect = CountVectorizer()
```

Definizione del "vocabolario dei termini"

A questo punto, bisogna definire il corpus di documenti rispetto al quale calcolare la rappresentazione. In questo caso, i documenti saranno le trame dei singoli libri.

Successivamente, attraverso la funzione "`count_vect.fit(...)`" potremo creare il cosiddetto "**vocabolario dei termini**", cioè l'insieme di tutte le diverse parole presenti nei documenti del corpus.

In [7]:

```
summaries = books["Summary"].tolist()  
_ = count_vect.fit(summaries)
```

Visualizziamo il numero di termini del vocabolario e una parte di esso:

In [8]:

```
print("Dimensione del vocabolario: ", len(count_vect.vocabulary_), "\n")  
print(list(count_vect.vocabulary_.items())[:10])
```

Dimensione del vocabolario: 103311

```
[('old', 64934), ('major', 55659), ('the', 90890), ('boar', 11527), ('on',  
65121), ('manor', 56222), ('farm', 32178), ('calls', 14579), ('animals', 4  
890), ('for', 34266)]
```

Come possiamo notare, il vocabolario è un dizionario in cui le "chiavi" sono le diverse parole trovate, mentre il "valore" associato ad ognuno di essi è l'"ID" associato alcorrispondente termine.

Calcolo della rappresentazione di singoli testi

Una volta definito il vocabolario, possiamo calcolare il vettore Bag of Words dei riassunti delle trame dei diversi libri del dataset attraverso l'apposita funzione "`count_vect.transform(...)`", la quale dovrà essere chiamata passando come input le trame dei diversi libri.

Usiamo inoltre la libreria "`tqdm`" per creare una "barra di caricamento" per seguire l'andamento del calcolo delle diverse rappresentazioni.

In [9]:

```
from tqdm import tqdm

BOWSummaries = []

for i in tqdm(range(len(summaries))):
    BOWSummaries.append(np.array(count_vect.transform([str(summaries[i])])).todense()).f1
```

100%|██████████| 12055/12055 [00:08<00:00, 1458.44it/s]

Consideriamo il primo libro del dataset, e stampiamo il corrispondente vettore Bag of Words:

In [10]:

```
print("TRAMA:\n\n ",summaries[0], "\n\n#####\n")
print("Dimensione del vettore Bag of Words: ", len(BOWSummaries[0]) , "\n\n#####\n")
print("Vettore BOW:\n\n" , BOWSummaries[0])
```

TRAMA:

Old Major, the old boar on the Manor Farm, calls the animals on the farm for a meeting, where he compares the humans to parasites and teaches the animals a revolutionary song, 'Beasts of England'. When Major dies, two young pigs, Snowball and Napoleon, assume command and turn his dream into a philosophy. The animals revolt and drive the drunken and irresponsible Mr Jones from the farm, renaming it "Animal Farm". They adopt Seven Commandments of Animalism, the most important of which is, "All animals are equal". Snowball attempts to teach the animals reading and writing; food is plentiful, and the farm runs smoothly. The pigs elevate themselves to positions of leadership and set aside special food items, ostensibly for their personal health. Napoleon takes the pups from the farm dogs and trains them privately. Napoleon and Snowball struggle for leadership. When Snowball announces his plans to build a windmill, Napoleon has his dogs chase Snowball away and declares himself leader. Napoleon enacts changes to the governance structure of the farm, replacing meetings with a committee of pigs, who will run the farm. Using a young pig named Squealer as a "mouthpiece", Napoleon claims credit for the windmill idea. The animals work harder with the promise of easier lives with the windmill. After a violent storm, the animals find the windmill annihilated. Napoleon and Squealer convince the animals that Snowball destroyed it, although the scorn of the neighbouring farmers suggests that its walls were too thin. Once Snowball becomes a scapegoat, Napoleon begins purging the farm with his dogs, killing animals he accuses of consorting with his old rival. He and the pigs abuse their power, imposing more control while reserving privileges for themselves and rewriting history, villainising Snowball and glorifying Napoleon. Squealer justifies every statement Napoleon makes, even the pigs' alteration of the Seven Commandments of Animalism to benefit themselves. 'Beasts of England' is replaced by an anthem glorifying Napoleon, who appears to be adopting the lifestyle of a man. The animals remain convinced that they are better off than they were when under Mr Jones. Squealer abuses the animals' poor memories and invents numbers to show their improvement. Mr Frederick, one of the neighbouring farmers, attacks the farm, using blasting powder to blow up the restored windmill. Though the animals win the battle, they do so at great cost, as many, including Boxer the workhorse, are wounded. Despite his injuries, Boxer continues working harder and harder, until he collapses while working on the windmill. Napoleon sends for a van to take Boxer to the veterinary surgeon's, explaining that better care can be given there. Benjamin, the cynical donkey, who "could read as well as any pig", notices that the van belongs to a knacker, and attempts to mount a rescue; but the animals' attempts are futile. Squealer reports that the van was purchased by the hospital and the writing from the previous owner had not been repainted. He recounts a tale of Boxer's death in the hands of the best medical care. Years pass, and the pigs learn to walk upright, carry whips and wear clothes. The Seven Commandments are reduced to a single phrase: "All animals are equal, but some animals are more equal than others". Napoleon holds a dinner party for the pigs and the humans of the area, who congratulate Napoleon on having the hardest-working but least fed animals in the country. Napoleon announces an alliance with the humans, against the labouring classes of both "worlds". He abolishes practices and traditions related to the Revolution, and changes the name of the farm to "The Manor Farm". The animals, overhearing the conversation, notice that the faces of the pigs have begun changing. During a poker match, an argument breaks out between Napoleon and Mr Pilkington, and the animals realise that the faces of the pigs look like the faces of humans, and no one can tell the difference between them. The pigs Snowball, Napoleon, and Squealer adapt Old Major's ideas into an actual philosophy, which they formally name Animalism. Soon after, Napoleon and Squealer indulge in the vices of humans (drinking alcohol, sleeping in beds, trading). Squealer is employed to alter the Seven Commandments to account for this humanisation, an allusion to the Soviet go

vernment's revising of history in order to exercise control of the people's beliefs about themselves and their society. The original commandments are: # Whatever goes upon two legs is an enemy. # Whatever goes upon four legs, or has wings, is a friend. # No animal shall wear clothes. # No animal shall sleep in a bed. # No animal shall drink alcohol. # No animal shall kill any other animal. # All animals are equal. Later, Napoleon and his pigs secretly revise some commandments to clear them of accusations of law-breaking (such as "No animal shall drink alcohol" having "to excess" appended to it and "No animal shall sleep in a bed" with "with sheets" added to it). The changed commandments are as follows, with the changes bolded: * 4 No animal shall sleep in a bed with sheets. * 5 No animal shall drink alcohol to excess. * 6 No animal shall kill any other animal without cause. Eventually these are replaced with the maxims, "All animals are equal, but some animals are more equal than others", and "Four legs good, two legs better!" as the pigs become more human. This is an ironic twist to the original purpose of the Seven Commandments, which were supposed to keep order within Animal Farm by uniting the animals together against the humans, and prevent animals from following the humans' evil habits. Through the revision of the commandments, Orwell demonstrates how simply political dogma can be turned into malleable propaganda.

```
#####
```

Dimensione del vettore Bag of Words: 103311

```
#####
```

Vettore BOW:

[0 0 0 ... 0 0 0]

Notiamo che:

- la dimensione della rappresentazione è pari alla dimensione del vocabolario
- poichè il vocabolario è molto ampio, la maggior parte delle componenti del vettore avranno valore pari a 0

Calcolo della similarità tra parole chiave e trame

Dopo aver calcolato i vettori Bag of Words delle trame dei libri, dobbiamo adesso calcolare la rappresentazione delle keyword inserite in input dall'utente.

Supponendo che le keyword siano memorizzate in un'apposita variabile, calcoliamo la relativa rappresentazione BOW:

In [11]:

```
keywords = "story of a group of farm animals who rebel against their human farmer, hopin  
BOWkeywords = np.array(count_vect.transform([keywords]).todense()).flatten()
```

A questo punto, per cercare libri le cui trame sono simili alle keyword inserite in input, una soluzione possibile è calcolare una **misura di distanza** tra la rappresentazione delle keyword e le rappresentazioni dei singoli testi.

In questo caso, la misura di distanza implementata è la **similarità del coseno**, definita come:

$$\text{cossim}(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

Il calcolo della similarità può essere effettuato attraverso due apposite funzioni della libreria "NumPy":

- **dot()**: calcolo del prodotto scalare tra due vettori
- **norm()**: calcolo della norma di un vettore

Calcoliamo quindi le diverse similarità tra le keyword e i libri come segue:

In [12]:

```
from numpy import dot
from numpy.linalg import norm

similarities = list()

for i in tqdm(range(len(BOWSummaries))):
    title = books.iloc[i]["Title"]
    author = books.iloc[i]["Author"]
    similarities.append( tuple ((title , author , (dot(BOWSummaries[i], BOWKeywords)) / (norm(BOWSummaries[i]) * norm(BOWKeywords))))))

similarities[0:10]
```

100% |██████████| 12055/12055 [00:06<00:00, 1722.36it/s]

Out[12]:

```
[('Animal Farm', 'George Orwell', 0.6654636523305504),
 ('A Clockwork Orange', 'Anthony Burgess', 0.5723264722039985),
 ('The Plague', 'Albert Camus', 0.5963706247825356),
 ('A Fire Upon the Deep', 'Vernor Vinge', 0.619436717447886),
 ('All Quiet on the Western Front',
  'Erich Maria Remarque',
  0.5752185890586847),
 ('A Wizard of Earthsea', 'Ursula K. Le Guin', 0.5632036628427579),
 ('Blade Runner 3: Replicant Night', 'K. W. Jeter', 0.47267155030848435),
 ('Blade Runner 2: The Edge of Human', 'K. W. Jeter', 0.4539543919769876),
 ('Crash', 'J. G. Ballard', 0.47943886564996496),
 ('Children of Dune', 'Frank Herbert', 0.5473379265047649)]
```

Per concludere, ordiniamo i diversi libri sulla base del coefficiente di similarità calcolato, e restituiamoci all'utente i libri più simili in base alle keyword inserite.

A titolo di esempio, restituiamo i primi 3 libri con il coefficiente di similarità più alto:

In [13]:

```
similarities = sorted(similarities, key = lambda x: x[2] , reverse = True)

print("Libri più simili alle parole chiave date in input:\n")
for i in range(3):
    print(i+1,' ',similarities[i][0] , '' , ', similarities[i][1] , ' (Coeff.Similarità
```

Libri più simili alle parole chiave date in input:

```
1 ) " Animal Farm " , George Orwell (Coeff.Similarità: 0.66546365233055
04 )
2 ) " Snowball's Chance " , John Reed (Coeff.Similarità: 0.664795002420
5023 )
3 ) " The Alteration " , Kingsley Amis (Coeff.Similarità: 0.65349842644
17577 )
```

Possiamo anche notare che i risultati "meno simili" all'input sono dei libri totalmente diversi da quello che l'utente ha richiesto: essi infatti molto probabilmente avranno dei coefficienti di similarità prossimi o uguali a 0:

In [14]:

```
similarities = sorted(similarities, key = lambda x: x[2] , reverse = False)

print("Libri meno simili alle parole chiave date in input:\n")
for i in range(3):
    print(i+1,' ',similarities[i][0] , '' , ', similarities[i][1] , ' (Coeff.Similarità
```

Libri meno simili alle parole chiave date in input:

```
1 ) " The Kennel Murder Case " , S. S. Van Dine (Coeff.Similarità: 0.0
)
2 ) " Slavers " , Chris Pramas (Coeff.Similarità: 0.0 )
3 ) " Deathstalker " , Simon Green (Coeff.Similarità: 0.0 )
```

Approccio alternativo: rappresentazione "Word Embeddings"

Oltre a rappresentare un testo con la rappresentazione "Bag of Words", possiamo usare un'altra tipologia di rappresentazione, chiamata "**Word Embeddings**".

Nello specifico, la rappresentazione Word Embeddings è una rappresentazione che tiene conto della semantica e del contesto delle singole parole, in maniera tale che **parole usate in maniera simile abbiano una rappresentazione simile**.

Data una serie di testi, in Python è possibile calcolare la rappresentazione Word Embeddings in diversi modi: in questo caso, si farà uso della libreria "SpaCy".

La prima cosa da fare è caricare uno dei modelli offerti dalla libreria: in questo caso, si è scelto di usare il modello "**en_core_web_md**".

Esso può essere caricato in Python con le seguenti istruzioni:

In [15]:

```
import spacy  
nlp = spacy.load('en_core_web_md')
```

A questo punto, per calcolare i Word Embedding delle trame dei diversi libri del dataset, si dovranno svolgere le seguenti operazioni:

- **Pre-processing sulle trame dei libri:** applicazione di una pipeline di NLP che prevede operazioni di tokenizzazione, rimozione delle Stop Words e della punteggiatura, e lemmatizzazione del testo, attraverso le apposite funzioni messe a disposizione dalla libreria "SpaCy"
- **Calcolo del vettore Word Embedding delle singole parole di un documento**
- **Calcolo del vettore Word Embedding di un intero documento:** ciò può essere fatto calcolando la media degli embedding delle parole presenti nel documento in questione

N.B. A differenza dell'algoritmo basato sulla rappresentazione *Bag of Words*, in questo caso consideriamo solo un ristretto sotto-insieme del dataset, in quanto in caso contrario il calcolo degli embedding richiederebbe troppo tempo.

In [16]:

```
# consideriamo solo i primi 200 Libri  
  
booksSample = books.head(n=200)
```

Pre-processing sulle trame

Attraverso le funzioni messe a disposizione dalla libreria "SpaCy", effettuiamo un lavoro di pre-procesing sulle trame dei libri.

Per quanto riguarda la rimozione delle Stop Words, all'interno di SpaCy è già implementato un vocabolario che contiene le possibili Stop Words:

In [17]:

```
stop_words = spacy.lang.en.stop_words.STOP_WORDS
```

```
print(stop_words)
```

```
{'made', 'fifty', "'re", 'forty', 'seems', 'who', 'into', 'call', 'everywh  
ere', 'elsewhere', 'eight', 'yourselves', 'mostly', 'whether', 'also', 'ha  
s', 'several', 'further', 'ever', 'beyond', 'although', 'due', 'themselve  
s', 'as', 'your', 'why', 'being', 'hundred', 'no', 'over', 'and', 'did',  
'n't', 'more', 'must', 'nowhere', 'again', 'becoming', "'ve", 'since',  
'm', 'in', 'latter', 'nobody', 'former', 'have', 'meanwhile', 'how', 'her  
eupon', 'get', 'such', 'nevertheless', 'both', 'much', 'n't', 'hereafter',  
'hereby', 'side', 'per', 'others', 'towards', 'next', 'thru', 'here', 'whe  
rein', 'nor', 'unless', 'ten', 'less', 'really', 'namely', 'either', 'thro  
ughout', 'they', 'be', 'two', 'put', 'ourselves', 'whenever', 'it', 'afte  
r', 'thereupon', 'noone', 'those', "n't", 'everything', 'each', 'full', 'o  
n', 'under', 'doing', 'yours', 'becomes', 'else', 'somewhere', "'ll", 'whi  
ch', 'with', 'if', 'out', 'last', 'now', 'nine', 'sometimes', 'our', 'ther  
eafter', 'three', 'anyone', 'he', 'therein', 'rather', 'five', 'together',  
'beside', 'part', 'toward', 'anyway', 'hence', 'against', 'across', 'excep  
t', 'their', 'his', 'my', 'thence', 'somehow', 'not', 'most', 'anyhow', 'm  
ine', "'d", 'seeming', 'quite', 'will', 'still', "'s", 'what', 'own', 'whi  
ther', 'the', 'say', 'do', 'go', 'third', 'other', 'ours', 'himself', 'b  
y', 'latterly', 'below', 'whereby', 'serious', 'had', 'give', 'four', 'tha  
n', 'at', 'moreover', 'are', 'anything', 'beforehand', 'back', 'hers', 'be  
sides', 'various', "'re", 'few', 'only', 'many', 'make', 'via', 'seemed',  
'off', 'first', 'eleven', 'herein', 'you', 'using', 'cannot', 'its', 'sho  
w', 'was', 'alone', 'from', 'an', "'m", 'though', 'please', 'yourself', 'a  
nother', 'onto', 'perhaps', "'ve", 'can', 'does', "'ll", 'through', 'nothi  
ng', 'within', 'however', 'too', 'seem', 'she', 'used', 'i', 'bottom', 'ma  
y', 'were', 'while', 'something', 'see', 'any', "'ve", 'should', 'often',  
'same', 'everyone', 'never', "'s", 'but', 'to', 'whereas', 'upon', 'anywhe  
re', 'would', 'just', 'least', 'therefore', 'them', 'when', 'afterwards',  
'll', 'thereby', 'me', 'whence', 'indeed', 'before', 'could', 'of', 'arou  
nd', 'whoever', 'herself', 'am', 'fifteen', "'s", 'been', 'all', 'us',  
'd', 'until', 're', 'twelve', 'we', 'formerly', 'whereupon', 'a', 'itsel  
f', 'keep', 'during', 'whose', 'about', 'sixty', 'very', 'became', 'or',  
're', 'her', 'even', 'twenty', 'almost', 'take', 'for', 'whereafter', 'so  
meone', 'once', 'well', 'that', 'name', 'this', 'always', 'down', 'along',  
'six', 'some', 'among', 'front', 'whatever', 'otherwise', 'empty', 'whol  
e', 'there', 'without', 'become', 'him', 'wherever', 'so', 'neither',  
"'m", 'then', 'above', 'is', 'up', 'already', 'move', 'enough', 'because',  
'myself', 'amount', 'between', 'one', 'where', 'might', 'none', 'yet', 'c  
a', 'whom', 'every', 'thus', 'sometime', 'these', 'top', 'done', "'d", 'am  
ongst', 'regarding', 'behind'}
```

Calcolo della rappresentazione "Word Embedding" delle trame

Come già detto, il calcolo della rappresentazione word embedding di un intero documento prevede dei passi ben precisi:

- estrazione delle parole dal testo
- calcolo della rappresentazione Word Embedding delle singole parole
- calcolo della rappresentazione Word Embedding dell'intero documento come media delle rappresentazioni delle parole in esso contenute

In [18]:

```
plotEmbeddings = []

for i in tqdm(range(len(booksSample))):
    plot = nlp(booksSample.iloc[i]['Summary'])
    # Tokenizzazione, rimozione delle stop word e Lemmatizzazione
    tokens = [token.lemma_ for token in plot if not token.is_stop and not token.is_punct]
    wordEmbeddings = []
    for t in tokens:
        wordEmbeddings.append(nlp(t).vector)
    emb = np.mean(wordEmbeddings, 0)
    plotEmbeddings.append(emb)
```

100% |██████████| 200/200 [04:33<00:00, 1.37s/it]

Adesso, così come fatto nel caso di Bag of Words, possiamo calcolare le similarità tra la rappresentazione delle keyword fornite in input dall'utente e le rappresentazioni delle trame dei singoli libri. Anche in questo caso, usiamo come misura di similarità la **similarità del coseno**.

In [19]:

```
keywords = "story of a group of farm animals who rebel against their human farmer, hopin

keywords = nlp(keywords)
tokens = [token.lemma_ for token in keywords if not token.is_stop and not token.is_punct]
wordEmbeddings = []
for t in tokens:
    wordEmbeddings.append(nlp(t).vector)
keywords = np.mean(wordEmbeddings, 0)

similarities = []

for i in range(len(booksSample)):
    similarities.append((booksSample.iloc[i]['Title'], booksSample.iloc[i]['Author'], d
sortedSimilarities = sorted(similarities, key=lambda similarities: similarities[2], rev
print("Libri più simili alle parole chiave date in input:\n")
for i in range(3):
    print(sortedSimilarities[i][0], " , ", sortedSimilarities[i][1], "(coeff. similari
```

Libri più simili alle parole chiave date in input:

Animal Farm , George Orwell (coeff. similarità: 0.89561796)
Ishmael , Daniel Quinn (coeff. similarità: 0.8559113)
The Memory of Earth , Orson Scott Card (coeff. similarità: 0.8425473)

Valutazione dei risultati

Al fine di valutare le performance del sistema, usiamo il cosiddetto "**Mean Reciprocal Rank**".

Il Mean Reciprocal Rank (MRR) è un indice statistico che permette di valutare un algoritmo che produce una lista di possibili risposte ad una query, le quali sono ordinate per "probabilità di correttezza".

In questo caso, una query equivale alla ricerca di libri simili a delle parole chiave, mentre la probabilità di correttezza è data dal coefficiente di similarità tra le rappresentazioni delle keyword inserite in input e le trame dei libri.

Formalmente, il Mean Reciprocal Rank è definito come:

$$MRR = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

dove Q è l'insieme delle query, mentre

$$rank_i$$

è la "posizione" del "risultato atteso" dalla query i -esima nella lista dei risultati restituita dall'algoritmo, ordinata secondo le probabilità di correttezza.

Per effettuare il processo di valutazione, eseguiamo le seguenti operazioni:

- definiamo un insieme di 20 libri, rispetto ai quali scegliamo un insieme di parole chiave che li descrivono: avremo quindi 20 query
- eseguiamo l'algoritmo usando ogni volta uno degli insiemi di parole chiave definiti al passo precedente, e calcoliamo per ognuno dei libri il rispettivo "Rank", cioè la "posizione" di tale libro nella lista dei risultati restituiti dall'algoritmo
- calcoliamo il MRR dell'algoritmo

N.B. per una valutazione dei risultati più coerente, consideriamo anche per Bag of Words la stessa frazione del dataset usata nel caso di Word Embeddings.

Definizione dell'insieme dei libri utili ad effettuare la valutazione

Consideriamo i primi 20 libri del dataset:

In [20]:

```
subset = books.head(n=20)
subset
```

Out[20]:

	Wikipedia ID	Freebase ID	Title	Author	Pub date	Genres	Sur
0	620	/m/0hhyy	Animal Farm	George Orwell	1945-08-17	Roman à clef,Satire,Children's literature,Spec...	Old b the Farr
1	843	/m/0k36	A Clockwork Orange	Anthony Burgess	1962	Science Fiction,Novella,Speculative fiction,Ut...	te li near Er
2	986	/m/0ldx	The Plague	Albert Camus	1947	Existentialism,Fiction,Absurdist fiction,Novel	Pla c ir
3	2080	/m/0wkt	A Fire Upon the Deep	Vernor Vinge	NaN	Hard science fiction,Science Fiction,Speculati...	The pos arou n
4	2152	/m/0x5g	All Quiet on the Western Front	Erich Maria Remarque	1929-01-29	War novel,Roman à clef	Th te s Bäu
5	2890	/m/011zx	A Wizard of Earthsea	Ursula K. Le Guin	1968	Children's literature,Fantasy,Speculative fict...	G you or one k
6	4081	/m/01b4w	Blade Runner 3: Replicant Night	K. W. Jeter	1996-10-01	Science Fiction,Speculative fiction	Liv Dec actir cc
7	4082	/m/01b56	Blade Runner 2: The Edge of Human	K. W. Jeter	1995-10-01	Science Fiction,Speculative fiction	Bec s r at ev
8	6020	/m/01t5z	Crash	J. G. Ballard	1973	Speculative fiction,Fiction,Novel	Th tl the e na
9	6628	/m/01y92	Children of Dune	Frank Herbert	1976	Science Fiction,Speculative fiction,Children's...	Nine Er Mu we

	Wikipedia ID	Freebase ID	Title	Author	Pub date	Genres	Sur
10	6629	/m/01y9j	Candide, ou l'Optimisme	Voltaire	1759-01	Satire,Bildungsroman,Picaresque novel	C cc ej ch:
11	6630	/m/01yb0	Chapterhouse Dune	Frank Herbert	1985-04	Science Fiction,Speculative fiction,Children's...	situat des G Th
12	6921	/m/01_mr	Carmilla	Sheridan Le Fanu	1872	Gothic fiction	pre by Le as
13	7817	/m/025zx	The Cider House Rules	John Irving	1985	Fiction	gr orph wher
14	7923	/m/026l0	Dracula	Bram Stoker	1897	Science Fiction,Speculative fiction,Horror,Inv...	The is epi forn Th
15	8237	/m/0297f	Don Quixote	Miguel de Cervantes	1605	Parody,Children's literature,Psychological nov...	, Q proto
16	8567	/m/02cxx	Dune Messiah	Frank Herbert	1969	Science Fiction,Speculative fiction,Children's...	yeai the des i "Sta
17	8757	/m/02fck	Darwin's Dangerous Idea	Daniel Dennett	1995	Philosophy,Science	M P Darv Dea Herc stc
18	9000	/m/02h3j	Death of a Hero	Richard Aldington	NaN	Fiction	
19	10861	/m/02y0f	The Trial	Franz Kafka	1925	Fiction,Absurdist fiction,Novel	t bi th fin&

Definizione delle parole chiave per ognuno di essi

Definiamo per ognuno dei libri scelti un insieme di frasi che ne descrive la trama:

In [21]:

```
summaryHardCoded = []
```

```
summaryHardCoded.append("story of a group of farm animals who rebel against their human .")
summaryHardCoded.append("In search of strong emotions, a boy commits many criminal acts.")
summaryHardCoded.append("Story about rats and a plague that breaks out. At first, everyone")
summaryHardCoded.append("story in an universe with different physical laws. The galaxy i")
summaryHardCoded.append("story during the World War I, when a young German soldier, afte")
summaryHardCoded.append("Story of a young mage who lives in an island. He is very powerf")
summaryHardCoded.append("Living on Mars, a man called Deckard is acting as a consultant .")
summaryHardCoded.append("The main character, Decard, refuges into a house with a woman,")
summaryHardCoded.append("story about car-crash sexual fetishism: its protagonists become")
summaryHardCoded.append("A man arrived in the desert. With his children still being infa")
summaryHardCoded.append("The protagonist was born in the castle of a Baron in Westphalia")
summaryHardCoded.append("story about struggles of the Bene Gesserit Sisterhood against t")
summaryHardCoded.append("In an isolated castle deep in the Styrian forest, Laura leads a")
summaryHardCoded.append("A young orphan grows up in an orphanage run by a pro-abortion d")
summaryHardCoded.append("A man about to get married goes to Transylvania to conclude the")
summaryHardCoded.append("Story of the life and insightful journey of a Spanish man who s")
summaryHardCoded.append("After the death of Baron Harkonnen and the defeat of the Sardau")
summaryHardCoded.append("Book which analyze the Charles Darwin theory about the evolutio")
summaryHardCoded.append("When word comes that the protagonist was killed in the war, his")
summaryHardCoded.append("Story of a young man who finds himself caught up in the mindles")
```

Calcoliamo adesso il MRR dell'algoritmo, considerando prima i testi rappresentati con Bag of Words, e successivamente con Word Embeddings.

MRR con rappresentazione Bag of Words

Prima di tutto, è necessario calcolare la rappresentazione Bag of Words delle "query":

In [22]:

```
queryBOW = []
```

```
for i in range(20):
    queryBOW.append(np.array(count_vect.transform([summaryHardCoded[i]]).todense()).flat
```

Calcoliamo adesso l'MRR:

In [23]:

```
totRank = 0.0

for i in tqdm(range(20)):    # fisso la query
    similarities = []
    for j in range(len(booksSample)):    # scorro tutti i 200 libri del dataset
        title = booksSample.iloc[j]["Title"]
        similarities.append( tuple ((title , (dot(BOWSummaries[j], queryBOW[i]))/(norm(B
sortedSimilarities = sorted(similarities, key=lambda similarities: similarities[1] ,
rank = 1
for j in range(len(sortedSimilarities)):
    if sortedSimilarities[j][0] != booksSample.iloc[i]["Title"]:
        rank = rank + 1
    else:
        break

totRank = totRank + (1/rank)

MRR = totRank / 20

print("MRR con Bag of Words = ",MRR, "\n")
```

100% |██████████| 20/20 [00:01<00:00, 11.58it/s]

MRR con Bag of Words = 0.4186654306963017

MRR con rappresentazione Word Embeddings

Effettuiamo adesso lo stesso procedimento utilizzando la rappresentazione Word Emeddings. Calcoliamo prima di tutto la rappresentazione Word Embeddings di tutte le query:

In [24]:

```
queryEMB = []

for i in range(20):
    tokens = [token.lemma_ for token in nlp(summaryHardCoded[i]) if not token.is_stop and token.pos_ != "PUNCT"]
    wordEmbeddings = []
    for t in tokens:
        wordEmbeddings.append(nlp(t).vector)
    queryEMB.append(np.mean(wordEmbeddings,0))
```

Infine, calcoliamo il MRR:

In [25]:

```
totRank = 0.0

for i in tqdm(range(20)):    # fisso la query
    similarities = []
    for j in range(len(booksSample)):    # scorro gli embedding dei libri
        title = booksSample.iloc[j]["Title"]
        similarities.append( tuple ((title , (dot(plotEmbeddings[j], queryEMB[i]))/(norm

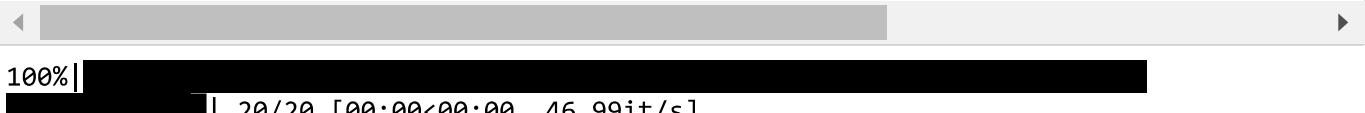
sortedSimilarities = sorted(similarities, key=lambda similarities: similarities[1] ,

rank = 1
for j in range(len(sortedSimilarities)):
    if sortedSimilarities[j][0] != booksSample.iloc[i]["Title"]:
        rank = rank + 1
    else:
        break

totRank = totRank + (1/rank)

MRR = totRank / 20

print("MRR con Word Embeddings = ",MRR,"\\n")
```



100% | 20/20 [00:00<00:00, 46.99it/s]

MRR con Word Embeddings = 0.5143330912308823

Conclusioni

Dai risultati e dalle misurazioni delle performance dell'algoritmo basate sulla misura "MRR", possiamo concludere che **le performance del sistema sono superiori se i testi sono rappresentati attraverso la rappresentazione Word Embeddings.**

Ciò intuitivamente può essere dovuto alle proprietà di tale rappresentazione, la quale, come già detto, tiene conto non solo della presenza di specifici termini nel testo da analizzare, ma anche e soprattutto del **modo con cui tali termini sono utilizzati nel testo**, dando quindi la possibilità di effettuare delle raccomandazioni più precise rispetto a quanto è possibile fare utilizzando invece la rappresentazione Bag of Words.

Bisogna inoltre considerare che l'analisi delle performance è effettuata soltanto su una porzione ridotta del dataset: con molta probabilità, effettuando tali analisi rispetto all'intero dataset e con un maggior numero di query, la differenza di performance tra le due varianti dell'algoritmo diventerebbe ancora più marcata.