Python/ Deep Learning project report

Finding Similar Movies based on their summary

Team-11

Siva Sameer Krishna Yarlagadda-44

Sai Nagarjuna Kolla-18

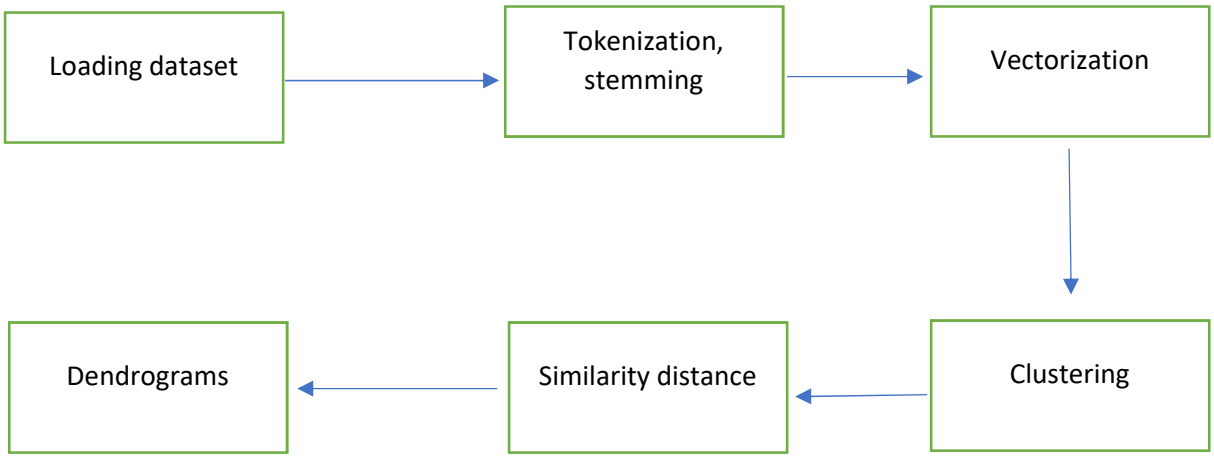
Motivation: Finding similar movies based on reviews and summaries but not by their genre.

If we consider Action movies, they are again of different classes like super heroes, battles, fighting, racing etc. So searching movies based on same genre might not be a good choice.

Dataset used: IMDB dataset

Dataset contains 5 columns movie_rank, movie_title, movie_genre. imdb_plot and wiki_plot

Architecture:



Loading the dataset:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('punkt')


movies = pd.read_csv("movies.csv")
print(movies.keys())
#print(movies)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Index(['rank', 'title', 'genre', 'wiki_plot', 'imdb_plot'], dtype='object')
```

Concatenating the movie summaries:

We have 2 summaries, IMDB and Wikipedia. But they might not be in same context. Since some features which goes missing in one movie summary might present in another one, there is a need to concatenate both of them.

Columns are in the form of Object data type, so they are converted to string type before concatenation.

```
movies["plot"] = movies["wiki_plot"].astype(str) + "\n" + movies["imdb_plot"].astype(str)

print(movies.head())
print(movies['plot'])
```

```
   rank                    title                                     genre  \
0     0             The Godfather                   [u' Crime', u' Drama']
1     1  The Shawshank Redemption                   [u' Crime', u' Drama']
2     2          Schindler's List  [u' Biography', u' Drama', u' History']
3     3               Raging Bull    [u' Biography', u' Drama', u' Sport']
4     4                Casablanca        [u' Drama', u' Romance', u' War']

                                        wiki_plot  \
0  On the day of his only daughter's wedding, Vit...
1  In 1947, banker Andy Dufresne is convicted of ...
2  In 1939, the Germans move Polish Jews into the...
3  In a brief scene in 1964, an aging, overweight...
4  It is early December 1941. American expatriate...

                                        imdb_plot  \
0  In late summer 1945, guests are gathered for t...
1  In 1947, Andy Dufresne (Tim Robbins), a banker...
2  The relocation of Polish Jews from surrounding...
3  The film opens in 1964, where an older and fat...
4  In the early years of World War II, December 1...

                                             plot
0  On the day of his only daughter's wedding, Vit...
1  In 1947, banker Andy Dufresne is convicted of ...
2  In 1939, the Germans move Polish Jews into the...
3  In a brief scene in 1964, an aging, overweight...
4  It is early December 1941. American expatriate...
```

Tokenization:

Concatenated Text is converted into sentences and then into word tokens. Word tokens are filtered by removing special characters.

```
s_tokens = [s for s in nltk.sent_tokenize("""In late summer 1945, guests
are gathered for the wedding reception of Don Vito Corleone's daughter Connie (Talia Shire) and Carlo Rizzi (Gianni Russo).""")]

w_tokens = [w for w in nltk.word_tokenize(s_tokens[0])]

import re

filtered_w_tokens = [w for w in w_tokens if re.search('[a-zA-Z]', w)]

print(filtered_w_tokens)
```

```
['In', 'late', 'summer', 'guests', 'are', 'gathered', 'for', 'the', 'wedding', 'reception', 'of', 'Don', 'Vito', 'Corleone', "'s", 'daughter', 'Connie', 'Talia
```

Stemming:

Same word may be in different forms, so breaking down the words to their root form using Stemming. Tokens before stemming and after stemming are displayed.

```
from nltk.stem.snowball import SnowballStemmer

snowball = SnowballStemmer("english")

print("before stemming:", filtered_w_tokens)

stemmed_w_tokens = [snowball.stem(w) for w in filtered_w_tokens]

print("after stemming:", stemmed_w_tokens)
```

```
before stemming: ['In', 'late', 'summer', 'guests', 'are', 'gathered', 'for', 'the', 'wedding', 'reception', 'of', 'Don', 'Vito', 'Corleone', "'s", 'daughter',
after stemming: ['in', 'late', 'summer', 'guest', 'are', 'gather', 'for', 'the', 'wed', 'recept', 'of', 'don', 'vito', 'corleon', "'s", 'daughter', 'conni', 't
```

We have to repeatedly perform tokenization and stemming in successive manner for large amount of text. So its better to combine these tasks as a single function.

```
def tokenizing_and_stemming(text):

    w_tokens = [w for s in nltk.sent_tokenize(text) for w in nltk.word_tokenize(s)]

    filtered_w_tokens = [w for w in w_tokens if re.search('[a-zA-Z]', w)]

    stemmed_w_tokens = [snowball.stem(w) for w in filtered_w_tokens ]

    return stemmed_w_tokens

stemmed_words = tokenizing_and_stemming("""In late summer 1945, guests
are gathered for the wedding reception of Don Vito Corleone's daughter Connie (Talia Shire) and Carlo Rizzi (Gianni Russo).""")
print(stemmed_words)
```

```
['in', 'late', 'summer', 'guest', 'are', 'gather', 'for', 'the', 'wed', 'recept', 'of', 'don', 'vito', 'corleon', "'s", 'daughter', 'conni', 'talia', 'shire',
```

Vectorization:

Converting the words into numbers by **TF-IDF vectorizer** in order to retrieve the some kind of meaning from them.

TF-IDF is used because it considers the word frequency in current document as well as in other documents and adjust the weight factor. It simply helps in identifying the unique words which are specific to the given document.

min_df: ignores the words which have frequency less than 0.25

max_df: ignores the words which have frequency more than 0.75

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vector = TfidfVectorizer(min_df=0.25, max_df=0.75, ngram_range=(1,3), stop_words='english', use_idf=True, tokenizer=tokenize_and_stem, ngram_range=(1,3))
```

Transforming the text into numeric form with tf-idf object created above.

```
tfidf_data = tfidf_vector.fit_transform([y for y in movies['plot']])

print(tfidf_data.shape)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:301: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokeni
  'stop_words.' % sorted(inconsistent))
(100, 393)
```

Clustering:

To find out how one movie is closely associated with another, K-means clustering is used.

6 clusters are taken. After each movie is assigned to its cluster, a separate column is created clusters which movie corresponds to. Total number of movies under each cluster are displayed.

```
from sklearn.cluster import KMeans

kmeans = KMeans(6)

kmeans.fit(tfidf_data)

kmeans_clusters = km.labels_.tolist()

movies["cluster"] = kmeans_clusters

print(movies['cluster'].value_counts())
```

```
3    39
1    23
2    17
0     9
5     7
4     5
Name: cluster, dtype: int64
```

Similarity between vectors is calculated by Cosine similarity.

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_distance = 1 - cosine_similarity(tfidf_data)
```

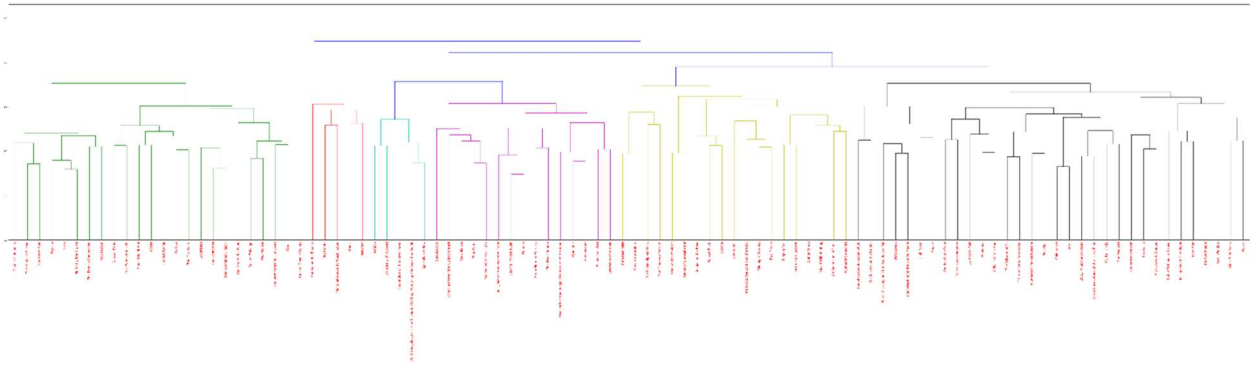
Based on similarity distance calculated above, to visualize the similarity, dendrograms are used which represents nodes in the hierarchical form.

Dendrograms are created based on complete linkage. Nodes on same level are considered similar to each other. Displaying the movie titles, movie id's according to leaves from (left to right) in dendrogram.

```
mergings = linkage(similarity_distance, method='complete')

dendrogram_ = dendrogram(mergings, labels=[x for x in movies["title"]], leaf_rotation=90, leaf_font_size=16)

fig = plt.gcf()
x = [lbl.set_color('r') for lbl in plt.gca().get_xmajorticklabels()]
fig.set_size_inches(108, 21)
labels=[x for x in movies["title"]]

print(dendrogram_.keys())
print(dendrogram_['leaves'])
print(dendrogram_['ivl'])


plt.show()
```

```
dict_keys(['icoord', 'dcoord', 'ivl', 'leaves', 'color_list'])
[94, 90, 97, 12, 76, 98, 63, 86, 7, 26, 27, 75, 16, 14, 84, 59, 0, 11, 6, 47, 58, 44, 77, 99, 89, 64, 69, 9, 82, 48, 57, 24, 29, 31, 88, 5, 4, 19, 38, 62, 3
['Double Indemnity', 'The Maltese Falcon', 'The Third Man', 'Psycho', 'Fargo', 'North by Northwest', 'The French Connection', 'Pulp Fiction', 'Citizen Kane',
```

Dendrogram plot:



References:

1. https://www.imdb.com/list/ls055592025/
2. http://kavita-ganesan.com/extracting-keywords-from-text-tfidf/#.XNg1yI5KjIU
3. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html
4. https://www.datacamp.com/community/tutorials/hierarchical-clustering-R