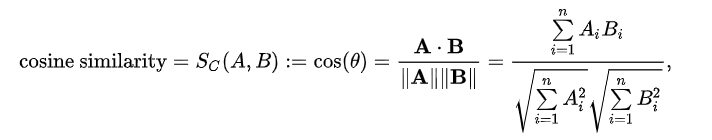
MOVIE RECOMMENDATION

Cosine similarity is the cosine of the angle between two n-dimensional vectors in an n-dimensional space. It is **the dot product of the two vectors divided by the product of the two vectors' lengths (or magnitudes)**.



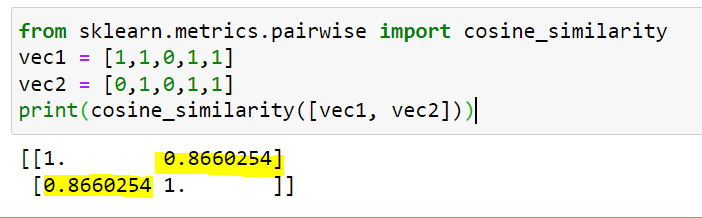
**Example**

from sklearn.metrics.pairwise import cosine\_similarity

vec1 = [1,1,0,1,1]

vec2 = [0,1,0,1,1]

print(cosine\_similarity([vec1, vec2]))



vec1 \* vec2 = 1\*0+1\*1+0\*0+1\*1+1\*1 = 0+1+0+1+1= 3

SQRT(1\*1+1\*1+0\*0+1\*1+1\*1) = SQRT(1+1+0+1+1) = SQRT(4) = 2

SQRT(0\*0+1\*1+0\*0+1\*1+1\*1) = SQRT(0+1+0+1+1) = SQRT(3) = 1.73

Cosine Similarity = 3/ (2\*1.73) = 3/ 3.46 = 0.8670

**Import Libraries**

import numpy as np

import pandas as pd

import difflib

from sklearn.feature\_extraction.text import TfidfVectorizer

# TfidfVectorizer - This is used to convert text data into numerical values

from sklearn.metrics.pairwise import cosine\_similarity

import os

**Check the current working directory**

display (os.getcwd())

**Change the current working directory and read data**

os.chdir('C:\\Noble\\Training\\Acmegrade\\Data Science\\Projects\\PRJ Movie Recommendation\\')

movies\_data =pd.read\_csv('movies.csv')

display (movies\_data.head())

**Display the shape**

display (movies\_data.shape)

**Selecting the relevant features for recommendation**

selected\_features = ['genres','keywords','tagline','cast','director']

print(selected\_features)

**Display the info**

display (movies\_data.info())

**Check for Null Values**

display (movies\_data.isna().sum())

**Display the selected columns**

display (movies\_data[selected\_features].head())

**Check Null Values in selected columns**

display (movies\_data[selected\_features].isna().sum())

**Replacing the null values with null string**

for feature in selected\_features:

movies\_data[feature] = movies\_data[feature].fillna('')

display (movies\_data.head())

**Check Null Values in selected columns**

display (movies\_data[selected\_features].isna().sum())

**Combining all the 5 selected features**

combined\_features = movies\_data['genres']+' '+movies\_data['keywords']+' '+movies\_data['tagline']+' '+movies\_data['cast']+' '+movies\_data['director']

display (combined\_features)

**Converting the text data to feature vectors/Bag of Words**

This is to find cosine similarity

# Vector shape is (4803, 17318). This is based on the number of distinct words. All the words will be converted to their equivalent numbers.

vectorizer = TfidfVectorizer()

feature\_vectors = vectorizer.fit\_transform(combined\_features)

display (feature\_vectors.shape)

display (pd.DataFrame(feature\_vectors.toarray()))

**Print TF-IDF Values**

TF-IDF stands for Term Frequency Inverse Document Frequency of the records. It can be defined as calculating how relevant a word in a series or corpus is to a text.

**TF-IDF (Term Frequency-Inverse Document Frequency)**

***Term Frequency:****In Document d, the frequency represents the number of instances of a given word t*

*tf(t,d) = count of t in d / number of words in d*

***Inverse Document Frequency***

*idf (t) =1 + log e [ n / df(t) ]*

*where*

*n = Total number of documents available*

*t = term for which idf value has to be calculated*

*df(t) = Number of documents in which the term t appears*

**TF-IDF   = tf(t, d) \* idf(t),** (Multiply tf and idf values)

The sklearn calculated TF-IDF values might not match with Manually calculated values since **smoothing**  and l2 normalization are applied in Sklearn calculated values

print (feature\_vectors)

**Getting the similarity scores using cosine similarity**

similarity = cosine\_similarity(feature\_vectors)

print (similarity )

**Print Shape Cosine Similarity**

display(similarity.shape)

**Enter the movie name to get Similarity**

movie\_name = input(' Enter your favourite movie name : ')

**Creating a list with all the movie names given in the dataset**

list\_of\_all\_titles = movies\_data['title'].tolist()

print(list\_of\_all\_titles)

**Length of the List**

print (len(list\_of\_all\_titles))

**Finding the close match for the movie name given by the user**

find\_close\_match = difflib.get\_close\_matches(movie\_name, list\_of\_all\_titles)

print(find\_close\_match)

**Display the close match**

close\_match = find\_close\_match[0]

print(close\_match)

**Finding the index of the movie with the title**

index\_of\_the\_movie = movies\_data[movies\_data.title == close\_match]['index'].values[0]

print(index\_of\_the\_movie)

**Get the similarity row for the selected index**

These are the similarity values for the movie entered by the user

similarity\_score = list(enumerate(similarity[index\_of\_the\_movie]))

print(similarity\_score)

**Length of Similarity Score**

len(similarity\_score)

**Sorting the movies based on their similarity score**

Display the index and similarity rating as a tuple

This list is sorted based on the similarity score

sorted\_similar\_movies = sorted(similarity\_score, key = lambda x:x[1], reverse = True)

print(sorted\_similar\_movies)

**Print the name of similar movies based on the index – Top 30**

print('Movies suggested for you : \n')

i = 1

for movie in sorted\_similar\_movies:

index = movie[0]

title\_from\_index = movies\_data[movies\_data.index==index]['title'].values[0]

if (i<30):

print(i, '.',title\_from\_index)

i+=1

**Consolidated code**

movie\_name = input(' Enter your favourite movie name : ')

list\_of\_all\_titles = movies\_data['title'].tolist()

find\_close\_match = difflib.get\_close\_matches(movie\_name, list\_of\_all\_titles)

close\_match = find\_close\_match[0]

index\_of\_the\_movie = movies\_data[movies\_data.title == close\_match]['index'].values[0]

similarity\_score = list(enumerate(similarity[index\_of\_the\_movie]))

sorted\_similar\_movies = sorted(similarity\_score, key = lambda x:x[1], reverse = True)

print('Movies suggested for you : \n')

i = 1

for movie in sorted\_similar\_movies:

index = movie[0]

title\_from\_index = movies\_data[movies\_data.index==index]['title'].values[0]

if (i<30):

print(i, '.',title\_from\_index)

i+=1