# MOVIE RECOMMENDATION SYSTEM

**Omar Kashmar**

**M.Tech Aerospace Engineering**

**IIT Bombay**

***213011004***

***Self-Project***

***Aug'23***

## 1. Introduction

In this project, I aimed to develop a movie recommendation system utilizing techniques from machine learning. The goal was to create a system that suggests movies to users based on their preferences and viewing history. The recommendation system utilizes exploratory data analysis and cosine similarity for generating movie recommendations.

## 2. Data Collection and Preprocessing

- Describe the dataset used for this project, including its source and the features available.
- Explain the preprocessing steps performed on the dataset. This may include handling missing values, eliminating correlated variables, and any data transformations applied.

## 3. Exploratory Data Analysis (EDA)

- Summarize the key insights gained from the exploratory data analysis.
- Include visualizations or statistics that highlight patterns, distributions, or trends in the data.
- Discuss any interesting findings that influenced the design of the recommendation system.

## 4. Cosine Similarity for Movie Recommendation

- Explain the concept of cosine similarity and its relevance in recommendation systems.
- Detail how the movie features were vectorized to compute cosine similarity.
- Describe the algorithm used to calculate the similarity scores between movies.

## 5. Implementation

- Provide an overview of the implementation process, including the tools and libraries used.
- Share code snippets or pseudocode that illustrate how you implemented the recommendation system.

## Explanation:

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)**.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

## 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]))

```
from sklearn.metrics.pairwise import cosine_similarity
vec1 = [1,1,0,1,1]
vec2 = [0,1,0,1,1]
print(cosine_similarity([vec1, vec2]))
```

```
[[1.        0.8660254]
 [0.8660254 1.       ]]
```

---

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

The **similarity score**, in the context of this movie recommendation system, refers to a numerical value that quantifies the similarity or closeness between two items (movies in this case). In collaborative filtering-based recommendation systems, like the one I have developed using cosine similarity, the similarity score represents how alike two items are based on certain features or attributes.

In this project, I've used cosine similarity to calculate the similarity scores between movie vectors. Here's a brief explanation of how cosine similarity works and how the similarity score is calculated:

1. **Vector Representation**: Each movie is represented as a vector in a high-dimensional feature space, where each dimension corresponds to a particular feature or attribute. These features could include genres, actors, directors, release year, etc.
2. **Cosine Similarity Calculation**: Cosine similarity is a metric that measures the cosine of the angle between two non-zero vectors. It's used to determine the similarity between the directions of these vectors rather than their magnitudes. The formula for cosine similarity between two vectors A and B is:

    - A and B are the vectors being compared.
    - A_i and B_i represent the values of the corresponding dimensions (features) in the vectors.
    - The dot product of A and B calculates the sum of the products of their corresponding dimensions.
    - The magnitudes of the vectors are calculated and multiplied to the dot product to normalize the similarity value.

3. **Interpreting the Score**: The cosine similarity score ranges from -1 to 1, with 1 indicating that the two vectors are identical, 0 indicating no similarity, and -1 indicating complete dissimilarity (opposite directions). In the context of your recommendation system, higher cosine similarity

scores suggest that the movies are more similar, and lower scores suggest less similarity.

4. **Recommendation**: For each user or movie, the system calculates the cosine similarity scores with respect to all other movies. It then ranks the movies based on their similarity scores in descending order and suggests the top-rated ones as recommendations.

In this project, the similarity score is used to sort and rank movies based on their similarity to a user's preferences or viewing history. Movies with higher similarity scores to the user's preferences are recommended more strongly because they are considered more relevant based on their shared characteristics.

The similarity score is a fundamental concept in recommendation systems that aids in generating meaningful and relevant recommendations for users.

## 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')
```

movies_data.head()

## Converting the text data to feature vectors

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)
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 : ')
```

**From the result: [26]**

Enter your favourite movie name: Wild Wild West
Movies suggested for you:

1 . Wild Wild West
2 . Jack Ryan: Shadow Recruit
3 . Hamlet
4 . Much Ado About Nothing
5 . Henry V
6 . The Helix... Loaded
7 . Men in Black II

8 . The Road to El Dorado
9 . Men in Black
10 . Men in Black 3
11 . Fled
12 . Abraham Lincoln: Vampire Hunter
13 . My Week with Marilyn
14 . Star Trek: Generations
15 . Get Shorty
16 . Meet Dave
17 . Once Upon a Time in Mexico
18 . The Velocity of Gary
19 . The Pirates! In an Adventure with Scientists!
20 . Chairman of the Board
21 . Grown Ups 2
22 . Thor
23 . After the Sunset
24 . Blade Runner
25 . Pokémon: Spell of the Unknown
26 . Edtv
27 . My Favorite Martian
28 . Recess: School's Out
29 . Bandidas

## 6. Results

- Present the results of the **recommendation** system, including its **accuracy** and efficiency.
- Showcase how the system performed in suggesting movies to users.
- Include metrics or visualizations that demonstrate the quality of the recommendations.

## 7. Discussion

- **Analyze** the strengths and limitations of the **implemented** recommendation system.
- Discuss any challenges encountered during the project and how you addressed them.
- Reflect on the practicality and real-world applications of the recommendation system.

## 8. Conclusion

- Summarize the key takeaways from the project.
- Highlight the achievements and contributions of the recommendation system.
- Suggest possible future improvements or extensions to enhance the system's performance.
- Performed exploratory data analysis and eliminated **correlated variables** and imputed missing variables.
- Used **Cosine similarity** for the vectorised movie feature and shorted the movies based on the **similarity score**.