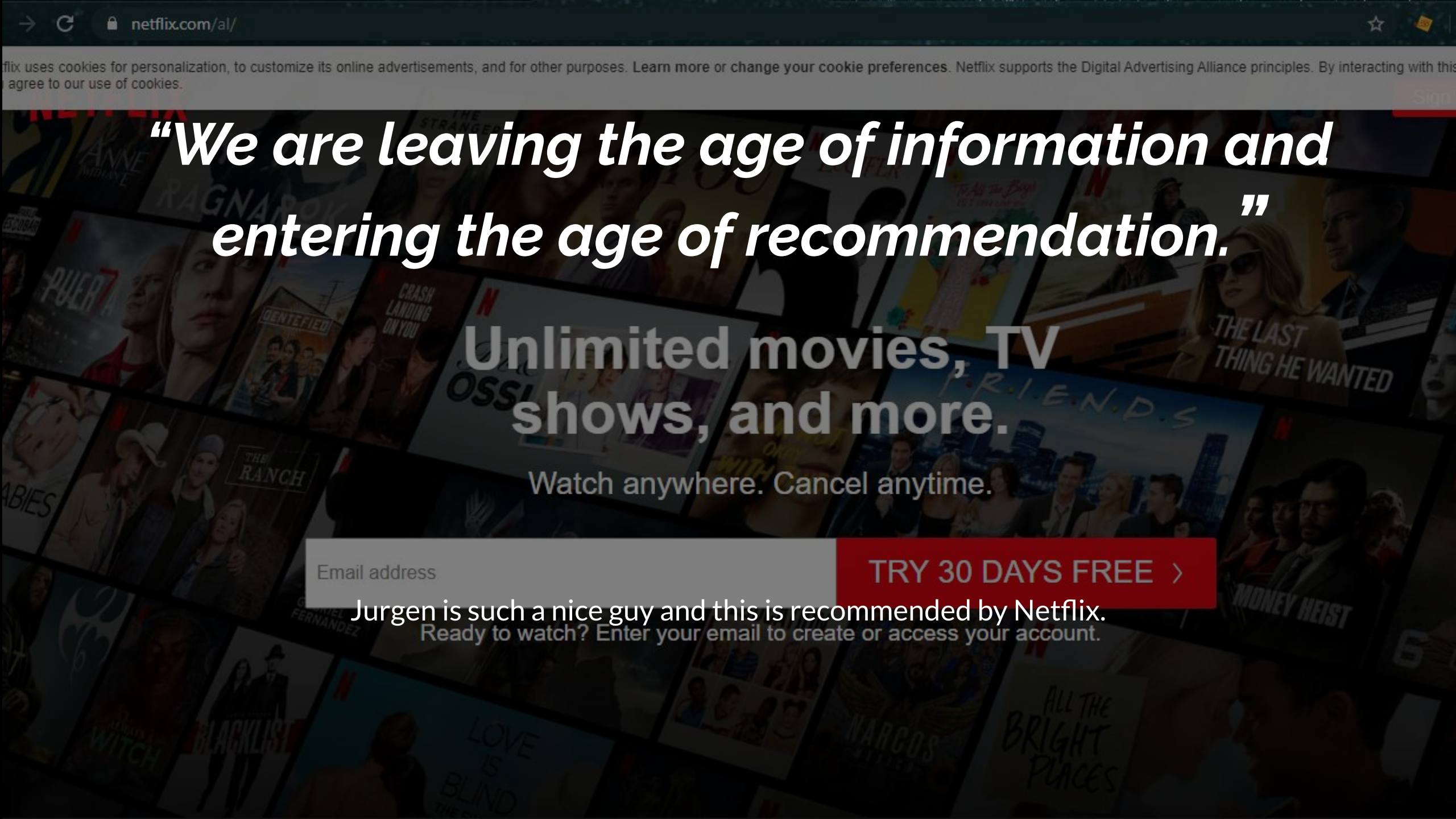


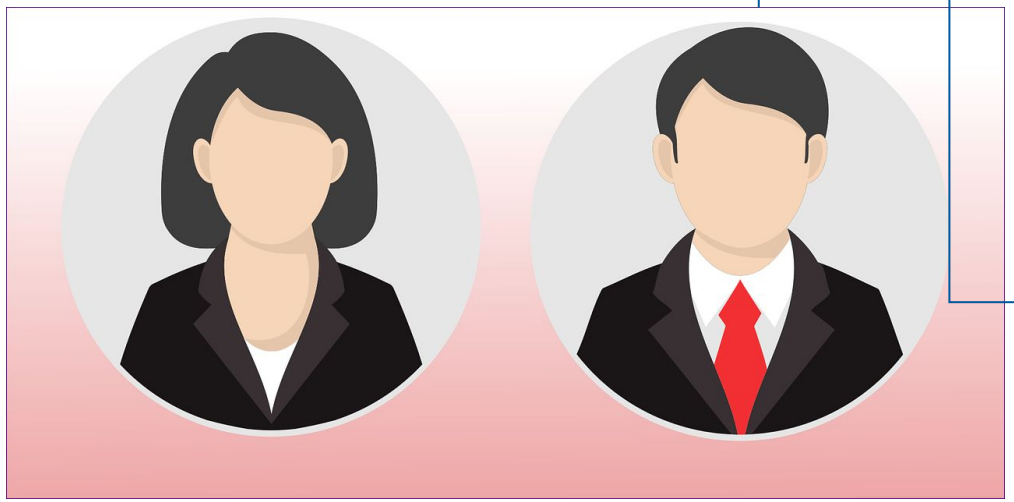
Jurgen is such a nice guy and this is recommended by Netflix.

Recommenation Systems - Netflix

When you create your Netflix account, or add a new profile in your account, we ask you to **choose a few titles that you like**. We use these titles to “jump start” your recommendations. Choosing a few titles you like is optional. If you choose to forego this step then we will start you off with a diverse and popular set of titles to get you going.

- Basically , the recommendations systems match the users with the items

Matching the users with their items of interest



Brands

同盟车 | SIMWOOD | PLAYBOY | THE COOLMIND | SELECTED | YEMEKE

View More

Color

- [ ] Red
- [ ] White
- [ ] Blue
- [ ] Green
- [ ] Black

View More

Set Fitness Suit Clothing Shorts Sp...

US $9.65 - 12.85

Shipping: US $1.79

★ 5.0 2536 Sold

Shop5619064 Store

NEGIZBER Polo-Shirt Short-Sleeve ...

US $7.99

Shipping: US $4.49

★ 4.7 2612 Sold

NEGIZBER Trend Store

Men 'S Slim Long Trench Coat Win...

New User Deal

US $16.98

Shipping: US $7.37

★ 3.3 57 Sold

Global Outlet Store

Trying to provide to the users what is most relevant to them

Ali Express

Trying to give to the users a better experience

During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems that could interest them) have taken more and more place in our lives. From e-commerce (suggest to buyers through online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.

In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy or anything else depending on industries).

Why are so important?

Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors.

As a proof of the importance of recommender systems, we can mention that, a few years ago, Netflix organised a challenges (the “Netflix prize”) where the goal was to produce a recommender system that performs better than its own algorithm with a prize of 1 million dollars to win.

Recommendation System

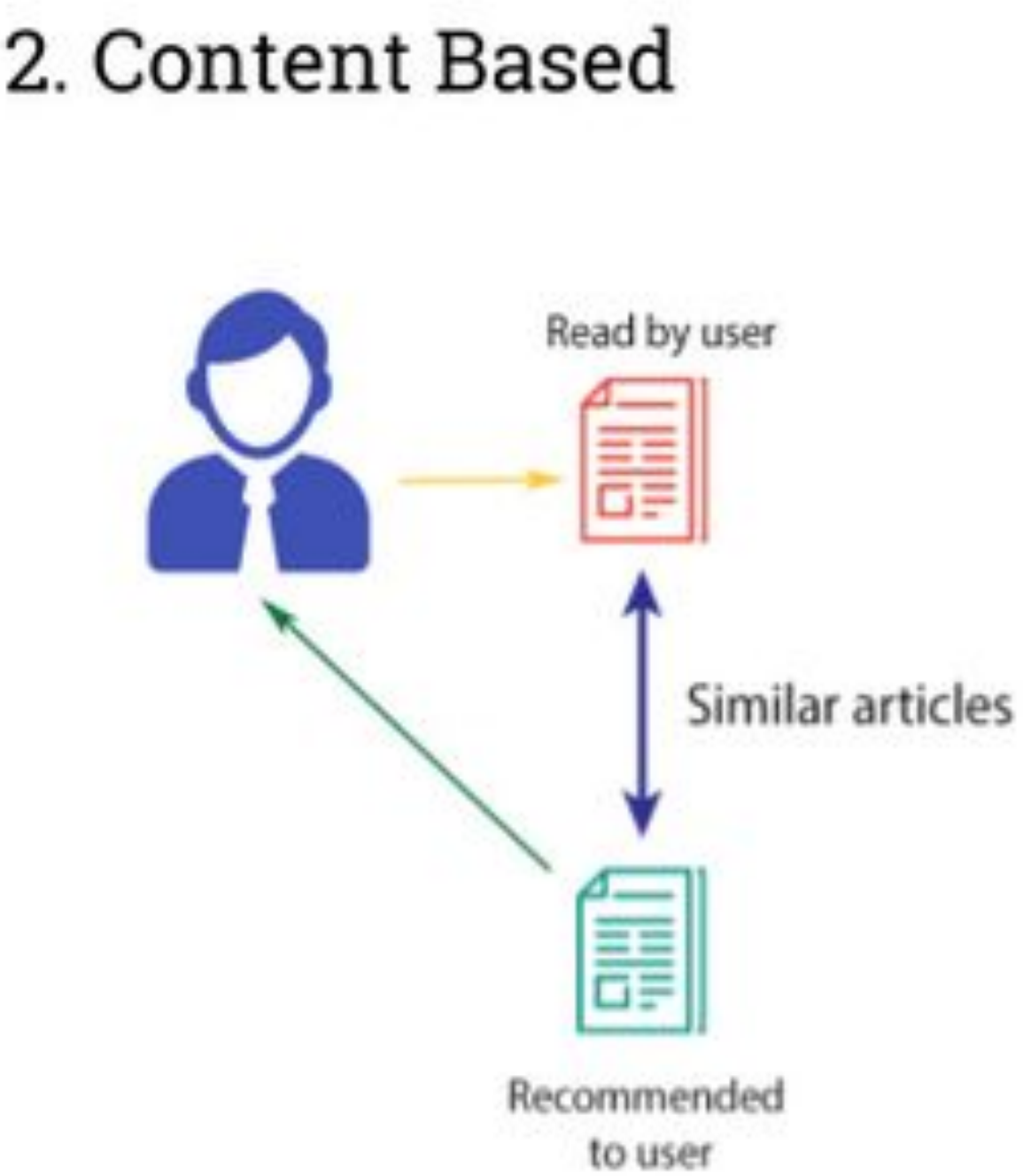
It turns out that there are (mostly) three ways to build a recommendation engine:

1. Popularity based recommendation engine
2. Content based recommendation engine
3. Collaborative filtering based recommendation engin

1. **Based on popularity , (trending) => those items/products which are mostly accessed and highest rated by the users**

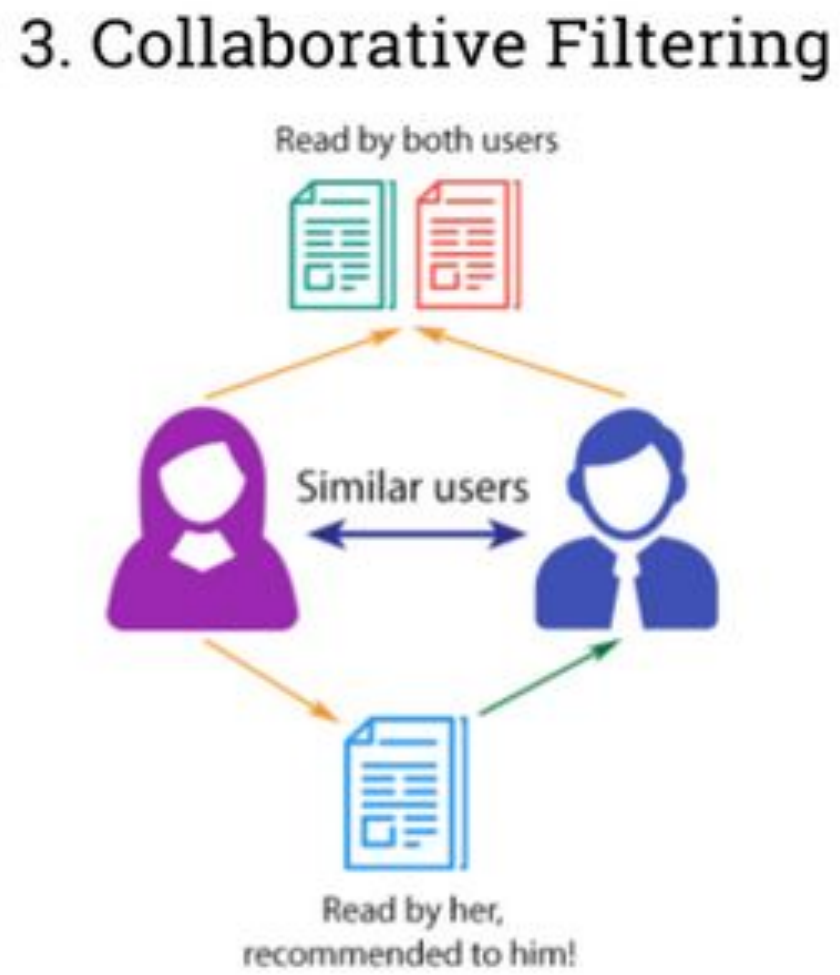
This is the simplest kind of recommendation engine that you will come across. The trending list you see in YouTube or Netflix is based on this algorithm. It keeps a track of view counts for each movie/video and then lists movies based on views in descending order(highest view count to lowest view count).

Content based , recommend similar items to those that the users are accessing



Collaborative Filtering ,
based on the users' historical preference on a set of items :

For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information collected from many users.



We have focused on content based recommender system

- A question that naturally arises : How to find the similarity between content ?

  More precisely , how to quantify their similarity

Cosine Similarity ~ Cauchy-Schwarz

$$\text{similarity} = \cos(\theta) = \frac{\vec{\mathbf{A}} \cdot \vec{\mathbf{B}}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Let's start with a simple example:

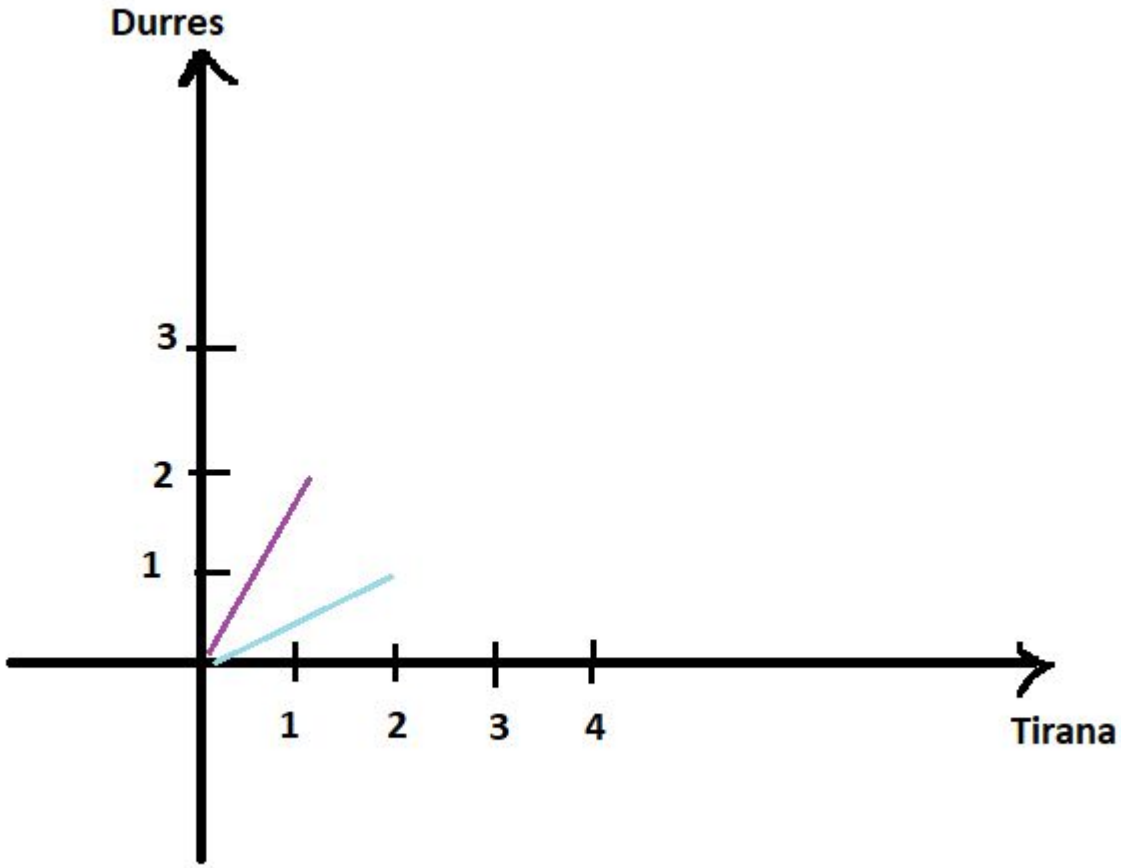
We have two texts:

Text 1: "Tirana Durres Tirana"

(Tirana 2, Durres 1)

Text 2: "Durres Durres Tirana"

(Tirana 1 , Durres 2)



```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

text = ['Tirana Durres Tirana','Durres Durres Tirana']
cv = CountVectorizer()


count_matrix = cv.fit_transform(text)  # should help in providing this output [[2,1],[1,2]]
print('Count matrix for the two variables')  #display the count for each name in a more visible way
print(count_matrix.toarray())
#in 1st sentence , Durres repeated once , Tirana twice
#in 2nd sentence , Durres repeated twice, Tirana once

similarity_scores = cosine_similarity(count_matrix)
print("Similarity")
print(similarity_scores) #first sentence is similar with second with 0.8
print("similarity between the two sentences is",similarity_scores[0][1])

```

Count Vectorizer helps us in counting the frequency of the words
To find the cosine similarity of the two vectors we used cosine_similarity

```
Count matrix for the two variables
[[1 2]
 [2 1]]
Similarity
[[1.  0.8]
 [0.8 1. ]]
similarity between the two sentences is 0.7999999999999999
[Finished in 1.4s]
```

DATASET

- The dataset is taken from IMDb and contains around 5000 movies

We will get a user input and recommend to him/her similar movies that he may also like

Firstly , we read the CSV file which contains our dataset of movies , using Pandas library

df stands for data frame

```
#Read CSV File

df = pd.read_csv("movie_dataset.csv")
```

Pandas DataFrames make manipulating your data easy, from selecting or replacing columns and indices to reshaping your data.

Let's have a closer look:

```
#Read CSV File

df = pd.read_csv("movie_dataset.csv")        #content=>df(data frame)

print(df.columns)                              #print the columns(features) of the content
```

Below are the columns of our dataset, we will select the columns based on which we will try to accurately show similar movies

```
Index(['index', 'budget', 'genres', 'homepage', 'id', 'keywords',
       'original_language', 'original_title', 'overview', 'popularity',
       'production_companies', 'production_countries', 'release_date',
       'revenue', 'runtime', 'spoken_languages', 'status', 'tagline', 'title',
       'vote_average', 'vote_count', 'cast', 'crew', 'director'],
      dtype='object')
```

```
features = ['keywords','cast','genres','director']
#selecting the features, based on them we will recommend to the user
```

```
def combine_features(row):
    try:
        #takes a row from our dataset and return the combined values of features
        return row['keywords'] + " "+row['cast'] + " "+ (row['genres'])+" "+row['director']

    except:
        #when we do this , we have NAN values for some of the feature columns, we need to fix this
        print("Error ",row)
```

```
    return (row['keywords']) + " "+(row['cast']) + " "+ (row['genres'])+" "+(row['director'])
TypeError: unsupported operand type(s) for +: 'float' and 'str'
[Finished in 2.4s]
```

```
for feature in features:
    df[feature] = df[feature].fillna('')
```

When we face the unsupported operand type for float and str for a certain feature in our dataset , we will assign a value “” to it,

```
def combine_features(row):
    return str(row['keywords']) + " "+str(row['cast']) + " "+ str(row['genres'])+" "+str(row['director'])
```

We could also just type cast the values into str, this also fixes the issue

Combine New Features

```
def combine_features(row):
    try:
        #takes a row from our dataset and return the combined values of features
        return row['keywords'] + " "+row['cast'] + " "+ (row['genres'])+" "+row['director']

    except:
        #when we do this , we have NAN values for some of the feature columns, we need to fix this
        print("Error ",row)
```

```
df['combined_features'] = df.apply(combine_features,axis=1)
#we apply the function for every row in our data frame object
```

Here , we are creating a new feature (named combined_features)

Its value will be a combined value of 4 features values namely : keywords, cast , genres , director

Finding similarities

```
text = ['Tirana Durres Tirana','Durres Durres Tirana']
cv = CountVectorizer()

count_matrix = cv.fit_transform(text)  # should help in providing this output [[2,1],[1,2]]
print('Count matrix for the two variables')  #display the count for each name in a more visible way
print(count_matrix.toarray())
#in 1st sentence , Durres repeated once , Tirana twice
#in 2nd sentence , Durres repeated twice, Tirana once

similarity_scores = cosine_similarity(count_matrix)
print("Similarity")
print(similarity_scores) #first sentence is similar with second with 0.8
```

We observe that we are using the same logic for comparing the similarity between movies as in the previously explained example. Here the difference is only the text that we are comparing. Now we are comparing the combined value of the features for each row (movie)

Finding the similarity using cosine function!

```
#Create count matrix from this new combined column
cv = CountVectorizer()

count_matrix = cv.fit_transform(df['combined_features'])
# Compute the Cosine Similarity based on the count_matrix

cosine_sim = cosine_similarity(count_matrix)
```

Cosine similarity between movies (*cosine_sim variable will have this matrix as a value*)

Movies	0	1	2	3
0	1	0.3	0.6	0.7
1	0.7	1	0.5	
2	0.6	0.5	1	
3	0.3			1

We observe from the table that movie 1 is most similar to movie 0 , since the cosine value between them is largest

```
similar_movies = list(enumerate(cosine_sim[movie_index]))
#enumerate the row of cosine similarity matrix and convert to a list
```

similar_movies = [(0,1),
(1,0.3),
(2,0.6),
(3,0.7)]

Sort the results based on similarity:

similar_movies = [(0,1),(1,0.3),(2,0.6),(3,0.7)]

After Sort:
similar_movies = [(0,1),(3,0.7),(2,0.6),(1,0.3)]

```
sorted_similar_movies = sorted(similar_movies,key=lambda x:x[1],reverse = True)
#we sort according to the second value( the similarity cosine value)
# reverse=True because we want it in descending order
```

Get the title of the move from given index

```
def get_title_from_index(index):
    return df[df.index == index]["title"].values[0]
#With this function we get the title of the movie , when given its index
```

Sort the movies based on similarities

```
sorted_similar_movies = sorted(similar_movies,key=lambda x:x[1],reverse = True)
#we sort according to the second value( the similarity cosine value)
# reverse=True because we want it in descending order
```

Print the final result

```
# Print titles of first 20 movies , which are most relevant to the selected movie

for i in range(20):
    print(get_title_from_index(sorted_similar_movies[i][0]))
```

```
movie_user_likes = "The Pink Panther"

# Get index of this movie from its title

movie_index = get_index_from_title(movie_user_likes)
```

```
The Pink Panther
Cheaper by the Dozen
Love the Coopers
Alex Cross
Micmacs
Barney's Great Adventure
Amidst the Devil's Wings
The 51st State
Paul Blart: Mall Cop
Bringing Down the House
The R.M.
Cheaper by the Dozen 2
Daddy Day Care
Beverly Hills Cop
Barnyard
Big Mommas: Like Father, Like Son
High Heels and Low Lifes
3000 Miles to Graceland
Shutter Island
Safe Men
[Finished in 3.2s]
```

When we pick this movie of genre: adventure ,comedy and crime

Cheaper by the Dozen 2 - Wikipedia

Cheaper by the Dozen 2 is a 2005 American family comedy film directed by Adam Shankman. Steve Martin, Bonnie Hunt, Hilary Duff, Piper Perabo, Alyson ...

Love the Coopers - Wikipedia

Love the Coopers is a 2015 American Christmas comedy-drama film directed by Jessie Nelson and written by Steven Rogers. The film stars an ensemble cast, ...

Plot · Cast · Production · Release

Micmacs (film) - Wikipedia

Micmacs is a 2009 French comedy film by French director Jean-Pierre Jeunet. Its original French title is **MicMacs** à tire-larigot (loosely "Non-stop shenanigans").

Barney's Great Adventure - Wikipedia

Barney's Great Adventure is a 1998 American musical comedy adventure film based on the children's television series Barney & Friends, featuring Barney the ...

GUI

```
# Import - Please check the requirements for every library that is used. Thank you
from tkinter import *
from tkinter import messagebox
from tkinter.ttk import Progressbar
```

Using Tkinter

Requirements: *pip install tkinter*

```
import pandas as pd
import numpy as np
# sklear used to train our "model"
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from PIL import ImageTk, Image
import time
```

Other libraries useful in the process of reading the dataset, manipulating data, learning, images

```
# This is a secondary window that will serve as a loading screen - Just for an appealing design
display = Tk()
# place the title
display.title("Loading Screen")
# Configure
display.configure()
# specify the path of the image
path = "images/myPic.png"
# open the image
img = ImageTk.PhotoImage(Image.open(path))
# Create a label for that image as content
my = Label(display, image=img)
# the image of tha label will be the desired image
my.image = img
# Place the image in the window
my.place(x=0, y=0)
# Display the window
Frame(display, height=516, width=5, bg='black').place(x=520, y=0)
# Put a label with an overiview information
lbl1 = Label(display, text="Recommendation System", font='Timesnewroman 20 ', fg='blue')
lbl1.config(anchor=CENTER)
# Put the label on the window
lbl1.pack(padx=100, pady=100)
```

The layout here is PACK. Just put at the window in form of a stack.

This is a loading screen just exploring tkinter library.

It does not have any special effect on the recommendation system.

How to get rid of loading screen

```
# A function to be called after some time (after loading screen have to go)
def call_display():
    display.destroy()


# Call the destroy function after 3 seconds
display.after(3000, call_display)
display.mainloop()

# Create Window object - this will be our main window
window = Tk(className=' Music Recommendation System')
```

The loading screen does not provide any specific information so we could remove it after 3 seconds by calling a function that destroys the window.

The next step is to create a new window that will be used to handle the recommendation system

Adding Labels

A label is a static text that could not be changed or modified later. It gives some information about the program and it stays in the window.

```
# define the labels - this will be the header part
l1 = Label(window, text='Music Recommendation System', bg='black', fg='white', pady=20, padx=20)
l1.grid(row=1, column=2)

# define a label that inform the user to enter the Title of the movie
l1 = Label(window, text='Title', pady=10)
l1.grid(row=2, column=1)

# Another label to get the director of the movie you are looking for
l1 = Label(window, text='Author', pady=10)
l1.grid(row=4, column=1)
```

Notice here that we use the grid LAYOUT: Grid is similar to a square notebook. We place the label in a specific square by determining the row and column number.

Adding text-fields

The text fields will be responsible to get the information from user as the title of the move or the director name.

```
# define Entries - textfields to enter information
title_text = StringVar()
e1 = Entry(window, textvariable=title_text, width=30)
e1.grid(row=2, column=2)

author_text = StringVar()
e2 = Entry(window, textvariable=author_text, width=30)
e2.grid(row=4, column=2)
```

The stringVar is used to handle text is just a string input. Here, we determine the length of the textfield so the length of the grid square holding this widget will be fixed to 30 pixels. Then we specify the positon of the textfields in our window.

Add the list box

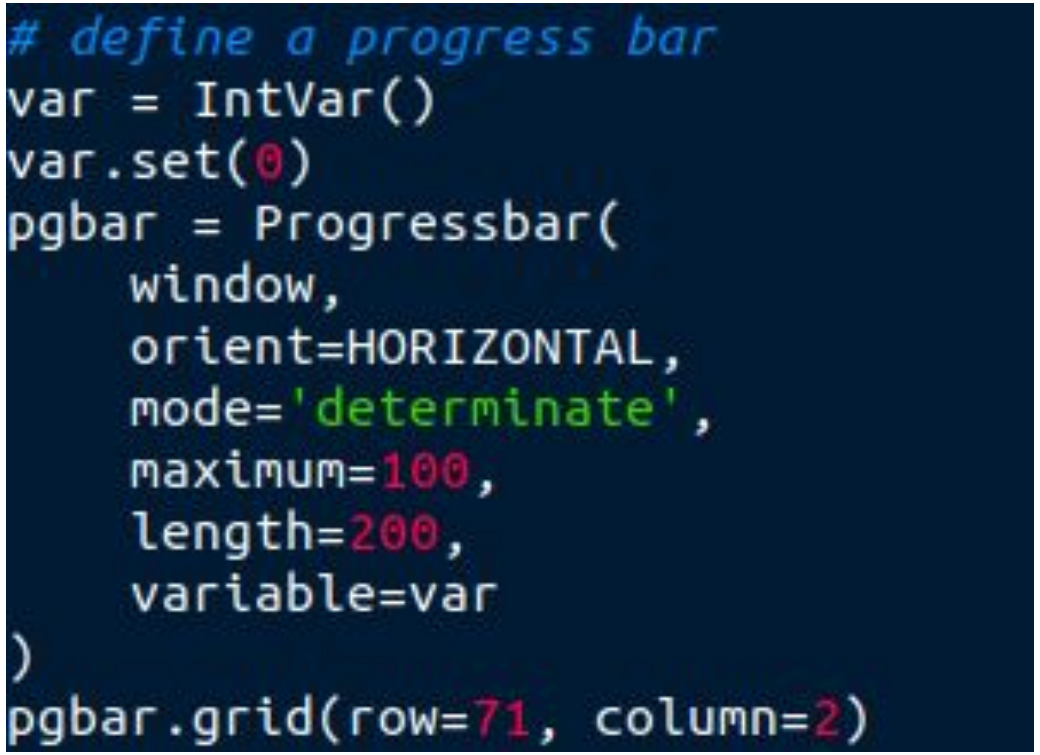
The list box will be used to store the result coming from the search.

```
# define ListBox - a placeholder to store the result of our search
list1 = Listbox(window, height=12, width=55)
list1.grid(row=20, column=1, rowspan=50, columnspan=60)

# Attach scrollbar to the list - easily explore the list
sb1 = Scrollbar(window)
sb1.grid(row=20, column=3, rowspan=60)

# Configure that scroll bar to our list box
list1.configure(yscrollcommand=sb1.set)
sb1.configure(command=list1.yview)
```

We specify the size of the list box, the column span and the row span, and than we add a scroll bar that will simplify the navigation of the list box(similar text area in JavaFx) We have to configure the scrollbar to respond only to this specific listbox.

A progress bar

This is a loading/progress bar that measures how much data is read during the process. Just a visual effect for the GUI. (Statistics)



```
# define a progress bar
var = IntVar()
var.set(0)
pgbar = Progressbar(
    window,
    orient=HORIZONTAL,
    mode='determinate',
    maximum=100,
    length=200,
    variable=var
)
pgbar.grid(row=71, column=2)
```

Define the buttons.

Note: Each button has a handler that is given by the command. Each handler is a function that does a specific task

```
# Define buttons
# Creating a photoimage object to use image
photo = PhotoImage(file="images/green.png")
b1 = Button(window, text='Google', width=15, command=recommend)
b1.grid(row=10, column=1)

# Define the search button
photo = PhotoImage(file="images/search_icon.png")
b1 = Button(window, text='Search Entry', width=15, command=onclick)
b1.grid(row=10, column=2)

# Define buttons
photo = PhotoImage(file="images/clear.png")
b1 = Button(window, text='Clear', width=85, command=clear, image=photo, compound=LEFT)
b1.grid(row=10, column=3)

# Set an icon to our main background
path = "images/search.png"
img = ImageTk.PhotoImage(Image.open(path))
my = Label(window, image=img)
my.image = img
my.grid(row=1, column=1)
```

Just a simple example:

This is the handler/function to clear the display after one search so we can have different result if we search again

```
# A function to clear the previous information on the screen
def clear():
    load_text.set('Clear the results!')
    # Clear the list box and setting the progress bar to 0
    global val
    val = 0
    var.set(val)
    list1.delete(0, END)
```

GOALS

Our recommendation system used only content based filtering to recommend movies not considering rating or popularity. Our idea is to add a recommendation based on rating of the movies where each user would *like/dislike* a movie and based on the new information we could build a more efficient recommendation system.(*HYBRID*)

References

https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

https://en.wikipedia.org/wiki/Collaborative_filtering

https://www.youtube.com/watch?v=XoTwndOgXBM