
**Module Code & Module Title**

**CU6051NI - Artificial Intelligence**

**Assessment Weightage & Type**

**80% Individual Coursework**

**Year and Semester**

**2019-20 Autumn**

**Student Name: Kushal Bhattarai**

**London Met ID: 17031137**

**College ID: NP01CP4A170221**

**Due Date: 13th January 2020**

**Submission Date: 13th January 2020**

**Word Count: 4346**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and marks of zero*

# Table of Contents

# Table of Figures

# Chapter 1. Introduction

Recommender systems are systems/techniques that recommend certain products, services, or entities. The recommendation system is based on the problem of prediction or ranking of the products or entities. (Banik, 2018) .The types of recommendation systems consist of Collaborative filtering, Content-based systems, and Hybrid systems.

Collaborative filters are one of the most popular recommender models in the industry. Amazon uses this collaborative filter recommendation to display the customer who also bought this. Item-based and user-based is two types of filtering in collaborative recommender. In user-based filtering, similar users are calculated who have a similar history of buying products or viewing products. Amazon's Customers who bought this item are an example of user-based filtering. If a group of people has rated two items similarly, then the two items must be similar is the idea in the collaborative system of item-based filtering. Amazon recommends the products to a user based on your browsing and purchase history. The whole idea is based on history and user ratings and user similarity having the problem of cold start. For a relatively new website, there is no enough data and the similarity and recommendation cannot be decided because of fewer amounts of data. (Banik, 2018)

In Content-based systems, the recommendation is based on the metadata of the products or items. Netflix is an example of the Content-based system. The first time you sign in to Netflix, it Netflix asks for likes and dislikes finding the similar content based on your like and dislike from the metadata like genres, title, overview. Content-based systems easily often come up with results that are usually clear and the user can know about the recommendation if they know about the movie types but for a simple user who wants and enjoys a certain type of movie/product the content-based recommender can be the best recommender. (Banik, 2018)

Hybrid recommenders are systems that are a combination of many recommenders Hybrid systems are used to invalidate the disadvantage of one model against an advantage of another using all the recommenders like knowledge-based, content, collaborative. (Banik, 2018)

Content-based recommendation systems suggest those objects that are alike in features to certain selected objects or viewed objects. A CB recommender has a content analyzer, profile learner and filtering component to give a proper recommendation to the users. Content-based filtering uses descriptions of the content of the items to learn the relationship between a single user and the description of the items. LIBRA, ACR News, Amalthaea, ifWeb, InfoFinder, INFOrmer, Let's Browse, Letizia, News Dude, NewT, PSUN, Re: Agent , SiteIF, Syskill & Webert, Webmate, and WebSail are some of the systems which /websites which use content-based filtering recommendations. Netflix uses a content-based recommender as movies/ series similar to the recently watched movies/ series are displayed. (Khusro et al., 2016)

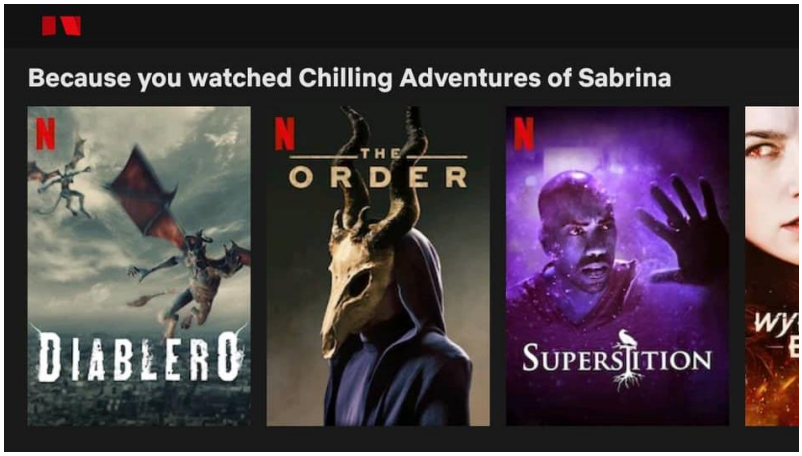


*Figure 1: Netflix recommender example*

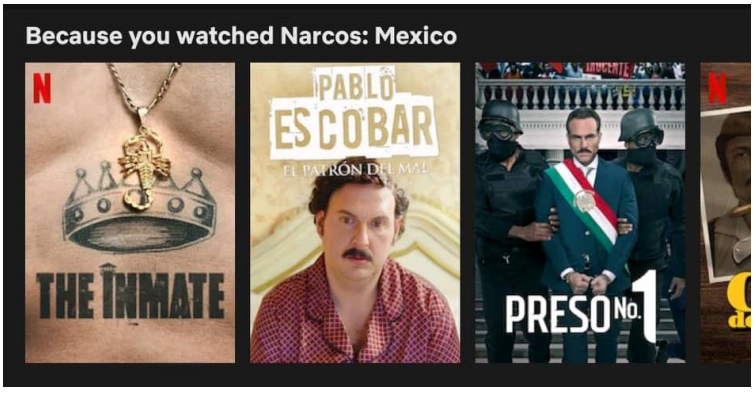


*Figure 2: Netflix recommender example*

## Chapter 2. Background

In the CB approach, different profile-item matching techniques are used to match features of new items with user profiles and to decide whether a particular item is interesting to a user or not. A user profile is either implicitly or explicitly updated, whereas utilities are assigned to items on the basis of utilities previously assigned to observed items. Item profile is represented by the features and descriptions of these items using either Boolean values or integer values representing term frequency (TF), or term frequency-inverse document frequency (TF-IDF). Items are automatically classified into relevant and non-relevant by comparing item representations with representations of user interest and preferences using keyword matching and cosine similarity.

One example of content-based filtering is Pandora Radio. When a user goes to the Pandora website, he/she is prompted to "Enter artist, genre or composer to create a station." Pandora then uses content-based filtering to find music with similar qualities to the song, artist, or genre that the user-provided. (Blanda, 2015)

Content-based recommenders depend on details about objects and consumers to be analyzed using information retrieval techniques. The limited availability of content leads to problems including overspecialization. Here, items are represented by their subjective attributes, where selecting an item is based mostly on their subjective attributes. Features that represent user preferences in a better way are not taken into account. For many domains, content is either scarce such as books or it is challenging to obtain and represent the content such as movies. In such cases, relevant items cannot be recommended unless the analyzed content contains enough information to be used in distinguishing items liked/disliked by the user. This also leads to a representation of two different items with the same set of features, where, e.g., well-written research articles can be difficult to distinguish from bad ones if both are represented with the same set of keywords. The limited content analysis leads to overspecialization in which CB recommenders recommend items that are closely related to the user profile and do not suggest novel items. In order to recommend novel and serendipitous items along with familiar items, we need to introduce additional hacks and note of randomness, which can be achieved by using genetic algorithms that bring diversity to recommendations being made. (Wang et al., 2018)

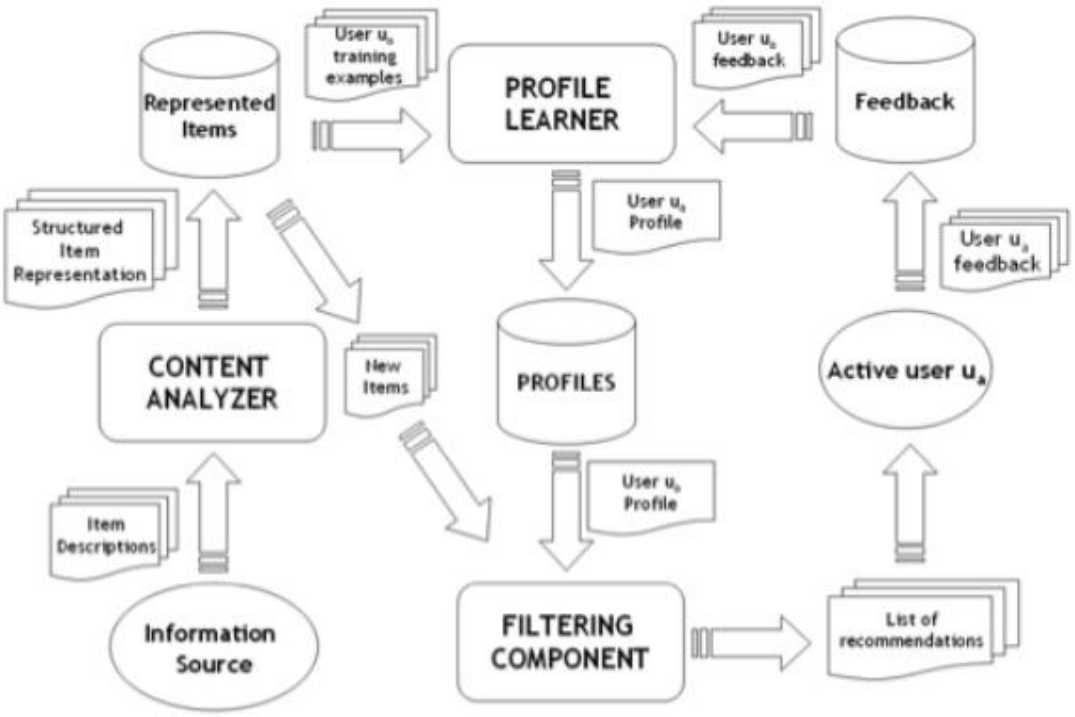


*Figure 3: High-level architecture of Content-based Recommender*

Content-Based Recommendation is a user-independent system that analyses certain items and a single user's profile for the recommendation making the process in finding similar items fast. Content-based recommender generates more dependable results with fewer users in the system. Content-based recommender items are recommended on a feature-level basis. New Items can be suggested even if the user is new and more data is not needed in the content-based recommendation. (Zha et al., 2015)

CBF has a number of advantages compared to stereotypes. CBF allows a user-based personalization so the recommender system can determine the best recommendations for each user individually, rather than being limited by stereotypes. CBF also requires less up-front classification work, since user models can be created automatically. Each item must be analyzed for its features, user models must be built, and similarity calculations must be performed. If there are many users and many items, these calculations require significant resources. (Beel et al., 2015)

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. Here it is used to analyze similarity in movies. (Martins et al., 2020)

TF-IDF representation algorithm is commonly used in a Content-based recommender also called Vector space representation. The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. (Wang et al., 2018)

Term frequency-inverse document frequency (TF-IDF) is a common weighting technique for information retrieval and text mining that reflects how important a word is for a document. The importance of a word increases proportionally to the times the word appears in the document. TF-IDF consists of TF and IDF, which are Term Frequency and Inverse Document Frequency respectively. TF represents the frequency a word appears in the document. If a term appears more in other documents, the term will be less important in the TF-IDF (Wang et al., 2018)

In the feature extraction section of the project, TfidfVectorizer is used to convert a collection of raw documents into a matrix of TF-IDF features. Parameters like analyzer, input, and stop_words can be used to extract proper data. (scikit-learn developers, 2019)

# Chapter 3. Solution

## 3.1 Solution

In recommending systems the content-based recommendation is an important approach. The basic idea is to recommend items similar to the products being viewed or the products that are liked. Calculating the similarities between things is the primary objective of the content-based recommender. Vector Space Model is used to model extracts keywords of the item and calculates the weight by TF-IDF. After calculating the TF-IDF matrix using the linear kernel to find the cosine similarities and the similarities are sorted and the sorted result is saved. Now a function can be called to display the recommendations of items in terms of their similarity score. While calculating similar items or users, we will apply cosine similarity on their rating vector and put them in descending order based on cosine similarity, which will sort all the other items based on similarity score, close to the vector we are comparing.

From the imported dataset using panda, for each movie a TF/IDF vector is generated in the terms in the movies keyword, genres , overview title or all combined. TF-IDF stands for term frequency times inverse document frequency. Term frequencies are the counts of each word in a document, Inverse document frequency means that you'll divide each of those word counts by the number of documents in which the word occurs. Then the cosine similarity of each movie's TF/IDF vector with every other paper's TF/IDF vector is calculated and the recommendation is a display based on similarity.

### 3.2 AI algorithms

Cosine similarity computes the L2-normalized dot product of vectors. That is, if and are row vectors, their cosine similarity is defined as:

$$k(x, y) = \frac{xyT}{(||x||\ ||y||)}$$

This is called cosine similarity because Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors. (scikit-learn developers, 2019)

Cosine similarity is the similarity between two non- zero vectors of an inner product space that measures the cosine of the angle between them. The orientation is the key factor in measuring similarity. In short, the similarity measure of two cosine vectors aligned in the same orientation is 1, when the perpendicularly aligned two vectors are of 0. If 2 vectors are diametrically opposed, which means that they are directed in exactly the opposite directions? Cosine similarity is often used in positive spaces between boundary 0 and 1. Cosine similarity is not affected and does not measure differences of magnitude (length) but represent similarities of orientation only. (Martins et al., 2020)

Similarities are determined by the data collection, mining, text knowledge retrieval and text matching The words in the data set can be evaluated and interpreted with a frequency analysis of the vector space model. This facilitates a calculation of cosine similarity to identify and compare texts on the basis of their differences and their sharing of topics. (DeepAI, 2020)

$$\boldsymbol{similarity} = \cos\boldsymbol{\theta} = \frac{\boldsymbol{A}\,.\,\boldsymbol{B}}{||\boldsymbol{A}||\,||\boldsymbol{B}||} = \frac{\sum_{i=1}^{n} \boldsymbol{AiBi}}{\sqrt{\sum_{i=1}^{n} \boldsymbol{B_i^2}}\sqrt{\sum_{i=1}^{n} \boldsymbol{A_i^2}}}$$

$$\boldsymbol{A}\,.\,\boldsymbol{B} = ||\boldsymbol{A}||\,||\boldsymbol{B}||\,\cos\boldsymbol{\theta}$$

*Figure 4: cosine similarity finding equation (DeepAI, 2020)*

The output from this equation falls in the range of -1 and 1. The system can implement how many similarities should be used to display using the range from -1 to 1. If the system wants opposite or dissimilar data the range can be from -1 up to 0 excluding and if similar data are needed the value from 0 to 1 can be ranged. (DeepAI, 2020)

TF–IDF aims to improve upon the bag of words approach by providing an indication of how important each word is, taking into account how often that word appears across the entire data. Term frequency is about how many times a term appears in the document. Assuming t is the number of times a term has appeared in the dataset denoted by d which can be represented by TF(t,d). Similarly, DF(t, D) can be used to denote the dataset occurrence D, that contain the term t. The inverse document frequency is denoted by $\mathbf{IDF(t, D)} = \mathbf{log}\frac{|\boldsymbol{D}|+\mathbf{1}}{\mathbf{DF(t,D)}+\mathbf{1}}$ . The IDF is used as a measure of how important a term is, taking into account how often that term appears across the entire dataset. Terms that are less common across the corpus have a higher IDF metric. The TF-IDF is calculated by using $\boldsymbol{TF}\,\mathbf{IDF(t, d, D)} = \mathbf{TF(td).IDF(t, d)}$

(Quddus, 2018)

### 3.3 Pseudo code

Import packages

Import and StoreSampledata.csv

Preparing and correct data

Finding similarities of items

Finding recommendation

Displaying recommendations

### 3.4 Diagrammatical Representation

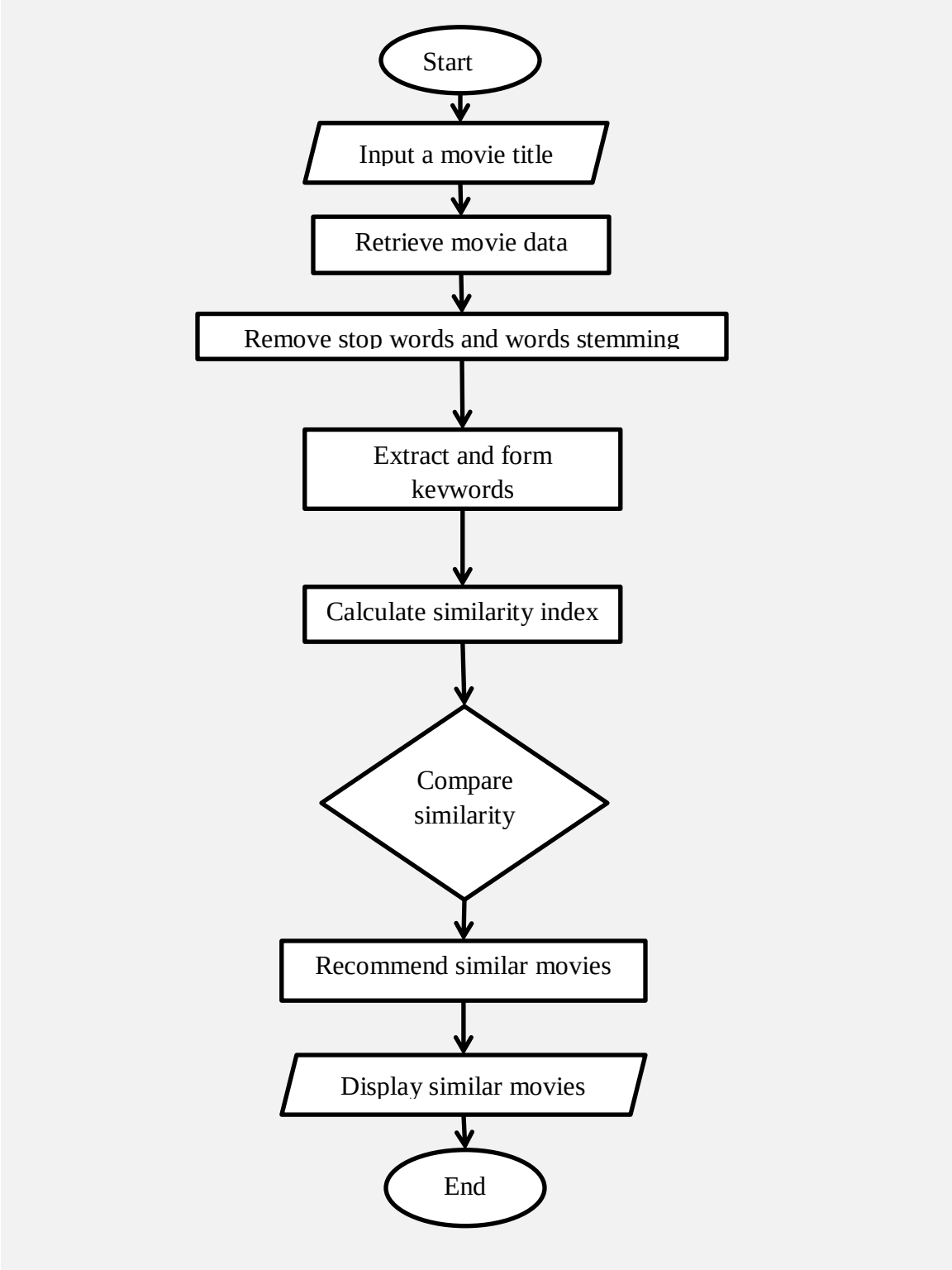


*Figure 5: flow chart*

## 3.5 Explanation of the development process

### 3.5.1 Libraries

- **numpy**

  NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. In this development process numpy is used to call a random function. (NumPy developers, 2020)

- **Pandas**

  Pandas are a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. (Panda, 2020)

- **ast**

  The ast module helps Python applications to process trees of the Python abstract syntax grammar. (Python Software Foundation., 2020)

- **Scikitlearn(sklearn)**

  Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities. (scikit-learn developers, 2019)

### 3.5.2 Class, Methods or Helpers

- **Helper for ast - literal_eval**

  Safely evaluate an expression node or a Unicode or Latin-1 encoded string containing a Python literal or container display. This can be used for safely evaluating strings containing Python values from untrusted sources without the need to parse the values oneself. (Python Software Foundation., 2020)

- **read_csv()**

  It is used to read csv file from the system. (Panda development team, 2014)

- **os.getcwd()**

  This method is used to get the current directory of the process. (Python Software Foundation, 2020)

- **fillna()**

  It is used to Fill NA/NaN values. (Pandas development team, 2014)

- **apply()**

  Applies a defines function along an axis (column or row) of the DataFrame. Row data was used in the program. (Pandas development team, 2014)

- **Join()**

  The join() method takes all items in an iterable and joins them into one string. (Python Software Foundation., 2020)

- **Feature extraction.text**

  The sklearn.feature_extraction module can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as text. (scikit-learn developers, 2020)

- **TfidfVectorizer**

  TfidfVectorizer was used to convert a collection of raw documents to a matrix of TF-IDF features. fit_transform() method of TfidfVectorizer was used to learn vocabulary and idf and return term-document matrix. (scikit-learn developers, 2019)

- **Sklearn.metrics.pairwise** was used to import to find the cosine similarity.

  Cosine similarity is the similarity between two non- zero vectors of an inner product space that measures the cosine of the angle between them. The orientation is the key factor in measuring similarity. (Martins et al., 2020)

- **randon.randomint()**

  This function returns a random integer from the given parameters. (The SciPy community, 2018)

- **iloc[]**

  iloc is used as integer-location based indexing for selection by position. (Pandas development team, 2014)

- **Series()**

  Series() is used to create One-dimensional ndarray with axis labels. Here index and movie title are kept in Series. (Panda developement team, 2014)

- **sorted()**

  sorted() is a built-in function that builds a new sorted list from an iterable. Here the similarity scores are sorted . (Panda development team, 2014)

- **list()**

  this method converts to other datatype to list.

- **enumerate()**

  Enumerate is a built-in function of Python. It allows us to loop over something and have an automatic counter. (Khalid, 2019)

- **lamda function**

  Python lambdas are anonymous functions used .

### 3.5.3 Development Process

The libraries were imported along with their functions. The dataset was read and the some features were extracted to find if the data that are read are python literal structure. The getlist function is created to change the dictionary data into list of genres and keywords. After the data are properly represented and can be evaluated easily. A key is created storing them in each row. The TF-IDF Vectorizer is used to analyse the data and any null values that are present are rewritten and the key, title, tagline and over view are added in the new key that will be used in tfidf vectorizer to create a matrix. The matrix will be compared using cosine similarity. Each row cosine similarity is calculated according and the data are sorted and save in a dictionary. Only 50 items are stored in the dictionary in sorted form. Then item function is created to get the title from the id using loc. The recommendation is defined using parameter id and only 5 recommendations can be displayed as the condition is given and called only 5 data that are similar from the id is called again a loop is used to check if the condition matches to be displayed. A counter is established to check if any items have been recommended. If the condition did not match and the similarity score was not high enough nor movies are recommended as no similar movies are present according to the system. The recommendation method calling is done by sending a random id that is inside the range of the id present in the dataset. Another method was created to display the similar movies in accordance to the title of the movie given. The dataset titles are saved in titles variable. Then the panda series is created taking the movie name, then a method is defined taking title as parameter , the cosine similarity is calculated , the list is sorted in accordance to scores usingn lamda function. Upto 5 data are taken and a loop is called to find the it od similar movies and the movies are displayed using titles.illoc[movie_indices]. Then the method is called taking string data as a parameter.

The coding portion was done in jupyter lab. JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows (Project Jupyter, 2020)

# Chapter 4. Achieved Results

```
import pandas as pd
import numpy as np
import os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from ast import literal_eval
ds = pd.read_csv(os.getcwd()+'\\sample data\\sampledata.csv')
features = ['keywords','genres']
for feature in features:
    ds[feature] = ds[feature].apply(literal_eval)
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        if len(names) > 1:
            names = names[:50]
        setname=set(names)
        listname=list(setname)
        return listname
    return []
for feature in features:
    ds[feature] = ds[feature].apply(get_list)
def create_key(x):
    return ' '.join(x['genres']) +' '+ ' '.join(x['keywords'])
ds['key'] = ds.apply(create_key, axis=1)
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
ds["overview"].fillna(" ", inplace = True)
ds["title"].fillna(" ", inplace = True)
ds["genres"].fillna(" ", inplace = True)
ds["keywords"].fillna(" ", inplace = True)
ds["tagline"].fillna(" ", inplace = True)
ds['key']=ds['key']+' '+ds['title']+' '+ds['tagline']+' '+ds['overview']
tfidf_matrix = tf.fit_transform(ds['key'])
cos_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)
result_dict = {}
for key, row in ds.iterrows():
    similar_ind = cos_similarities[key].argsort()[:-50:-1]
    similar_items = [(cos_similarities[key][i], ds['id'][i]) for i in similar_ind]
    result_dict[row['id']] = similar_items[1:]
def item(id):
    return ds.loc[ds['id'] == id]['title'].tolist()[0]
def recommend(id):
    num=5
    print("Movie "+ item(id) )
    print(' ')
    recs = result_dict[id][:num]
    for rec in recs:
        if rec[0] > 0:
            print(item(rec[1]))
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
```

*Figure 6: first portion of the coding*

The above code is how the output is gained and the movies are recommended.

```
titles = ds['title']
indices = pd.Series(ds.index, index=ds['title'])
def get_recommendations(title):
    print('Movie ' +title )
    idx = indices[title]
    sim_scores = list(enumerate(cos_similarities[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:6]
    movie_indices = [i[0] for i in sim_scores]

    return titles.iloc[movie_indices]
```

```
get_recommendations('X-Men')
```

*Figure 7: additional code*

```
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
Movie The Land Girls

The Incredibly True Adventure of Two Girls In Love
Tea with Mussolini
Unbroken
The Big Parade
Black Book
```

*Figure 8: recommendation of a movie found using random selection-1*

The 5 movies that are similar to the movies are recommended and displayed as output.

```
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
Movie Point Break

The Punisher
Big Momma's House 2
Extreme Ops
The Last Shot
Big Mommas: Like Father, Like Son
```

*Figure 9: recommendation of a movie found using random selection-2*

The same thing happens here as 5 movies that meet the similarity score and are displayed.

```
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
Movie Little Fockers

Meet the Parents
Meet the Fockers
My Big Fat Greek Wedding
Trippin'
Highway
```

*Figure 10: recommendation of a movie found using random selection-3*

The same thing happens here as 5 movies that meet the similarity score and are displayed.

```
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
Movie Flipper

Forget Me Not
Dolphin Tale
True Lies
Licence to Kill
Diary of a Wimpy Kid: Dog Days
```

*Figure 11: recommendation of a movie found using random selection-4*

The same thing happens here as 5 movies that meet the similarity score and are displayed.

```
get_recommendations('X-Men')
Movie X-Men
64              X-Men: Apocalypse
203                            X2
33       X-Men: The Last Stand
232                The Wolverine
101          X-Men: First Class
Name: title, dtype: object
```

*Figure 12: recommendation of a movie found using title-1*

```
get_recommendations('The Avengers')
Movie The Avengers
7          Avengers: Age of Ultron
26     Captain America: Civil War
511                         X-Men
68                       Iron Man
79                     Iron Man 2
Name: title, dtype: object
```

*Figure 13: recommendation of a movie found using title-2*

The last two pictures use a different function to get movie name and prints the recommended movies.

# Chapter 5. Conclusion

In this project first, the data set are downloaded from kaggle website and the data set was reviewed. Research work was done in terms of the recommendation system and two topics initially were selected. The two topics were collaborative and context-based recommender. After proper research work was done of the system needed to be developed the content-based recommender was selected. The topic was approved by the lecturer and tutor teacher and the work was started. The data set analysed first was reviewed and the contents of the data set were learned. Then the algorithms related to content-based recommendation were researched and cosine similarity and Pearson similarity algorithm were found to be suitable for the program. The cosine similarity was chosen to be done in the program. The coding portion was too much thought many research was done. The linux_kernal function or cosine similarity functions were two functions that could be used to find similarity. The cosine similarity function was used to find the similarity.

The content-based recommender can be used in E-commerce sites as well as movie streaming websites, learning management systems, health care, mobile and cloud-based applications. The future application of context-based recommender can be context recommendation methods with implicit context inference and modeling, Recommendation methods incorporating the context in recommendation process, Developing context-based recommendation systems for different domains like mobile, cloud and user applications.

Any further work that can be done is to create a web application where the user can view movie data, and the recommendations according to the movies. The dataset can be increased that will help in finding more accurate recommendation system. Additional data can be added to the dataset and content based recommender can be continued and developed into hybrid and more complex version of the recommendation system can created.

## Bibliography

Balabanovic, M. & Shoham, Y. (1997) FAB: Content-based collaborative recommendation. *COMMUNICATIONS OF THE ACM*, 40(3).

Emmanuel, M. & Deshpande, A.R. (2016) Context based Recommendation Methods: A Brief Review. *International Journal of Computer Applications (0975 – 8887) International Conference on Cognitive Knowledge Engineering 2016*.

Hapke, H., Howard, C. & Lane, H. (2019) *Natural Language Processing in Action*. Manning Publications.

Medium towards datascience. (2019) *introduction-to-two-approaches-of-content-based-recommendation-system-fc797460c18c* [Online]. Available from: https://towardsdatascience.com/introduction-to-two-approaches-of-content-based-recommendation-system-fc797460c18c [Accessed 11 January 2020].

TFIDF. (2020) *http://www.tfidf.com/* [Online]. Available from: http://www.tfidf.com/ [Accessed 12 January 2020].

Vaidya, N. & Khachane, A.R. (2017) Recommender systems-the need of the ecommerce ERA. *International Conference on Computing Methodologies and Communication (ICCMC)*, pp.100-04. Available at: https://ieeexplore.ieee.org/document/8282616/keywords.

## References

Banik, R. (2018) *Hands-On Recommendation Systems with Python*. Packt Publishing.

Beel, J., Gipp, B., Langer, S. & Breitinger, C. (2015) Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 14(4), pp.305-38.

Blanda, S., 2015. "Online Recommender Systems – How Does a Website Know What I Want?" *American Mathematical Society*, 25 May.

DeepAI. (2020) *cosine-similarity* [Online]. Available from: https://deepai.org/machine-learning-glossary-and-terms/cosine-similarity [Accessed 10 January 2020].

Khalid, M.Y.U. (2019) *enumerate.html* [Online]. Available from: http://book.pythontips.com/en/latest/enumerate.html [Accessed 6 February 2020].

Khusro, S., Ali, Z. & Ullah, I. (2016) *Recommender Systems: Issues, Challenges,and Research Opportunities*. illustrated ed. Singapore: Springer 2017.

Martins, L.F. et al. (2020) *Python: End-to-end Data Analysis*. Packt Publishing.

NumPy developers. (2020) *https://numpy.org/* [Online]. Available from: https://numpy.org/ [Accessed 6 February 2020].

Panda developement team. (2014) *pandas.Series.html* [Online]. Available from: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html [Accessed 7 February 2020].

Panda development team. (2014) *pandas.read_csv.html* [Online]. Available from: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html [Accessed 6 February 2020].

Panda development team. (2014) *sorting.html* [Online]. Available from: https://docs.python.org/3/howto/sorting.html [Accessed 12 February 2020].

Panda. (2020) *https://pandas.pydata.org/* [Online]. Available from: https://pandas.pydata.org/ [Accessed 6 February 2020].

Pandas development team. (2014) *pandas.DataFrame.apply.html* [Online]. Available from: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html [Accessed 6 February 2020].

Pandas development team. (2014) *pandas.DataFrame.fillna.html* [Online]. Available from: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html [Accessed 6 February 2020].

Pandas development team. (2014) *pandas.DataFrame.iloc.html* [Online]. Available from: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html [Accessed 7 February 2020].

Python Software Foundation. (2020) *ast.html* [Online]. Available from: https://docs.python.org/2/library/ast.html [Accessed 6 February 2020].

Python Software Foundation. (2020) *stdtypes.html?highlight=join#str.join* [Online]. Available from: https://docs.python.org/3/library/stdtypes.html?highlight=join#str.join [Accessed 6 February 2020].

Project Jupyter. (2020) *https://jupyter.org/* [Online]. Available from: https://jupyter.org/ [Accessed 6 February 2020].

Python Software Foundation. (2020) *os.html* [Online]. Available from: https://docs.python.org/3/library/os.html [Accessed 6 February 2020].

Quddus, J. (2018) *Machine Learning with Apache Spark Quick Start Guide*. Packt Publishing.

scikit-learn developers. (2019) *https://scikit-learn.org/stable/getting_started.html* [Online]. Available from: https://scikit-learn.org/stable/getting_started.html [Accessed 6 February 2020].

scikit-learn developers. (2019) *https://scikit-learn.org/stable/modules/metrics.html#cosine-similarity* [Online]. Available from: https://scikit-learn.org/stable/modules/metrics.html#cosine-similarity [Accessed 11 January 2020].

scikit-learn developers. (2019) *sklearn.feature_extraction.text.TfidfVectorizer.html* [Online]. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html [Accessed 12 January 2020].

scikit-learn developers. (2020) *classes.html#module-sklearn.feature_extraction.text* [Online]. Available from: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text [Accessed 6 February 2020].

The SciPy community. (2018) *numpy.random.randint.html* [Online]. Available from: https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.randint.html [Accessed 6 February 2020].

Wang, D.H. et al. (2018) Knowledge-Based Systems. *A content-based recommender system for computer science publications*, 157, pp.1-9.

Zha, Z., Cheng, Z., Hong, L. & Chi, E.H. (2015) *Improving User Topic Interest Profiles by Behavior Factorization*. Research Article. WWW '15: Proceedings of the 24th International Conference on World Wide Web.

# Appendix

## CODE

```
import pandas as pd

import numpy as np

import os

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

from ast import literal_eval

ds = pd.read_csv(os.getcwd()+'\\sample data\\sampledata.csv')

features = ['keywords','genres']

for feature in features:

    ds[feature] = ds[feature].apply(literal_eval)

def get_list(x):

    if isinstance(x, list):

        names = [i['name'] for i in x]

        if len(names) > 1:

            names = names[:50]

        setname=set(names)

        listname=list(setname)

        return listname
```

```
    return []
for feature in features:
    ds[feature] = ds[feature].apply(get_list)
def create_key(x):
    return ' '.join(x['genres']) +' '+ ' '.join(x['keywords'])
ds['key'] = ds.apply(create_key, axis=1)
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
ds["overview"].fillna(" ", inplace = True)
ds["title"].fillna(" ", inplace = True)
ds["genres"].fillna(" ", inplace = True)
ds["keywords"].fillna(" ", inplace = True)
ds["tagline"].fillna(" ", inplace = True)
ds['key']=ds['key']+'      '+ds['title']+'      '+ds['tagline']+' '+ds['overview']
tfidf_matrix = tf.fit_transform(ds['key'])
cos_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)
result_dict = {}
for key, row in ds.iterrows():
    similar_ind = cos_similarities[key].argsort()[:-50:-1]
    similar_items = [(cos_similarities[key][i], ds['id'][i]) for i in similar_ind]
```

```
    result_dict[row['id']] = similar_items[1:]
def item(id):
    return ds.loc[ds['id'] == id]['title'].tolist()[0]
def recommend(id):
    num=5
    print("Movie "+ item(id) )
    print(' ')
    recs = result_dict[id][:num]
    for rec in recs:
        if rec[0] > 0:
            print(item(rec[1]))
recommend(id= np.random.randint(1,len(cos_similarities),1)[0])
titles = ds['title']
indices = pd.Series(ds.index, index=ds['title'])
def get_recommendations(title):
    print('Movie ' +title )
    idx = indices[title]
    sim_scores = list(enumerate(cos_similarities[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = sim_scores[1:6]
    movie_indices = [i[0] for i in sim_scores]
```

```
    return titles.iloc[movie_indices]
```