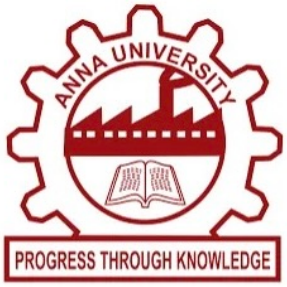
# University College of Engineering Villupuram




## Department of Computer Science and Engineering

**Experience Based Project Learning - IBM (E2324)**
**conducted by IBM**


**Personalized content recommendation system**
**phase - 4 Document**

**Submitted by**

1. S. Akalya
2. K. Harinitha
3. G. Muthazhagi
4. E.S. Mangala yazhini
5. PC .Vaishanavi

# Personalized content recommendation system

## Phase 4 : Model Building and Evaluation of Movie Recommendation system

### Introduction :

In this phase of our project, the emphasis is on constructing recommendation models using prepared data and selecting evaluation metrics to ensure precise and relevant recommendations. Recommendation systems are focused on enhancing user experience, especially with rich textual data, and leveraging similarity algorithms provides a strong foundation for building content-based movie recommendation systems.

### Objectives :

1. **Develop Recommendation system using similarity algorithm :**
   TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity are both components of similarity algorithms, particularly in the context of information retrieval and text processing to create robust content-based recommendation models that can accurately predict user preferences.

2. **Optimize Model for Enhanced Performance** :
   Fine-tune the parameters and configurations of the recommendation models to achieve optimal performance, ensuring they deliver the most accurate and relevant movie suggestions.

3. **Evaluate Model Performance** :
   Assess the effectiveness of the recommendation models using multiple evaluation metrics, including Precision@K, Recall@K, F1 Score, and Mean Average Precision (MAP), to ensure comprehensive performance analysis.

4. **Select the Most Suitable Model for Deployment Based on Evaluation Results** :
   Based on the evaluation results, identify and choose the best-performing model for deployment, ensuring it meets the desired criteria for accuracy, efficiency, and user satisfaction.

### Model Development :

Content-based recommender systems are a subset of recommender systems that tailor recommendations to users by analyzing items' intrinsic characteristics and attributes. These systems focus on understanding the content of items and mapping it to users' preferences. By examining features such as genre, keywords, metadata, and other descriptive elements, content-based recommender systems create profiles for both users and items.

**Working of Content Based Recommender Systems :**
A content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

**Similarity Algorithm for content based recommendation :**
In the context of natural language processing, we compute the cosine similarity value of two words or documents within a corpus, based on their vector representation using techniques, such as TF-IDF.

**Cosine similarity :**
Cosine similarity is often considered the best measure for content-based recommendation systems as it is well suited for sparse data, which is common in recommendation systems. However, the best measure may vary depending on the data, the requirements of the system, and the goals of the recommendation.

**Term Frequency (TF) :**
TF is simply the frequency of a word in a document. It means how often a term occurs in a document. In the context of natural language, terms correspond to words or phrases.

**formula for Term frequency :**

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

**Inverse Document Frequency (IDF) :**
IDF is the inverse of the document frequency among the whole corpus of documents. Less frequent are more informative than frequent terms like the. Therefore, some answer scorers multiply term match scores by the inverse document frequency (idf), which is computed from a large corpus using the equation

**formula for Inverse document frequency :**

$$idf(t) = \log_2 \frac{N}{c(t)+1}$$

TF-IDF weighting negates the effect of high frequency words in determining the importance of an item (document).But while calculating TF-IDF, log is used to dampen the effect of high frequency words. In other words the relevance of a word in a document cannot be measured as a simple raw count and hence the equation below :

**Equation :**

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

**Vector Space Model :**

After calculating TF-IDF scores, to do determine which items are closer to each other is accomplished using the Vector Space Model which computes the proximity based on the angle between the vectors.

In this model, each item is stored as a vector of its attributes (which are also vectors) in an n-dimensional space and the angles between the vectors are calculated to determine the similarity between the vectors. Next, the user profile vectors are also created based on his actions on previous attributes of items and the similarity between an item and a user is also determined in a similar way.

The ultimate reason behind using cosine is that the value of cosine will increase with decreasing value of the angle between which signifies more similarity. The vectors are length normalized after which they become vectors of length 1 and then the cosine calculation is simply the sum-product of vectors.

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.

**Model Evaluation :**

Model evaluation is the process of assessing the performance and effectiveness of a machine learning model. It involves quantitatively measuring how well the model performs its intended task using various metrics and techniques. Model evaluation is crucial for understanding how well a model generalizes to new, unseen data and for comparing different models to choose the best one for a particular problem.

Precision and Recall at K are common metrics that help evaluate the performance of ranking algorithms.Precision, Recall, and F-score can take values from 0 to 1. Higher values mean better performance.

**1. Precision @ K :**

Precision measures the proportion of relevant items among the recommended items.Precision@K calculates the precision considering only the top-K recommended items.

$$\text{Precision at K} = \frac{\text{Number of relevant items in K}}{\text{Total number of items in K}}$$

In the code, it's calculated by the function precision_at_k.

**2. Recall @ K :**

Recall measures the proportion of relevant items that were successfully recommended.Recall@K calculates the recall considering only the top-K recommended items.In the code, it's calculated by the function recall_at_k.

$$\text{Recall at K} = \frac{\text{Number of relevant items in K}}{\text{Total number of relevant items}}$$

**3. F1 score :**

F1 Score is the harmonic mean of precision and recall, providing a balance between the two metrics.It's used when there's a need to consider both precision and recall simultaneously.In the code, it's calculated by the function f1_score.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

These metrics are commonly used to evaluate recommendation systems to assess their effectiveness in providing relevant recommendations to users.

**Code for Recommendation system :**

The dataset loading ,data preprocessing , data formatting and Exploratory data analysis are performed in previous phases of this project .

**Similarity Algorithm for developing content based recommendation :**

**Import the libraries needed for find similar movies :**

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

**Overview of the dataset :**

```
df.head()
```

**Create an instance of TdidfVectorizer :**

```
tf_idf = TfidfVectorizer(stop_words='english')
```

**Transforming Text Data into a TF-IDF Matrix :**

```
tfidf_matrix = tf_idf.fit_transform(df['tags'])
```

**Calculating Cosine Similarity :**

```
cos_similarity = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

**Creating a Mapping of Movie Titles to Indices :**

```
data_set = pd.Series(df.index ,index = df["title"]).drop_duplicates()
```

**Function to calculate the similarity between the movies :**

```
def get_recommendation(title ,cosine = cos_similarity) :
        if title not in list(df["title"]) :
                print(f"{title} not present in the dataset")
                return
        movie_index = data_set[title]
        sim_scores = cosine[movie_index]
        sim_scores = list(enumerate(sim_scores))
        sim_scores = sorted(sim_scores , key = lambda x : x[1] , reverse = True)
        top_recommendation = sim_scores[1:15]

        l = [ ]
        for i in top_recommendation :
                l.append(i[0])

        movies_list = df["title"].iloc[l]
        recommended_list = []
        for movie in movies_list :
                recommended_list.append(movie)
        return recommended_list

def print_movie(get_recommendation):
        recommended_list = get_recommendation
        print("\nRecommended movies : \n")
        for recommended_movie in recommended_list :
                print(recommended_movie)
```

**Testing of the Recommendation system :**

```
print_movie(get_recommendation("The Dark Knight Rises"))
print_movie(get_recommendation("Spider-Man"))
print_movie(get_recommendation("Iron Man 3"))
```

**Model Evaluation :**

**# Define functions for evaluation metrics**

```
def precision_at_k(actual, predicted, k):
        actual_set = set(actual)
        predicted_set = set(predicted[:k])
        common_items = actual_set.intersection(predicted_set)
        precision = len(common_items) / k if k > 0 else 0
        return precision

def recall_at_k(actual, predicted, k):
        actual_set = set(actual)
        predicted_set = set(predicted[:k])
        common_items = actual_set.intersection(predicted_set)
        recall = len(common_items) / len(actual) if len(actual) > 0 else 0
        return recall

def f1_score(precision, recall) :
        f1 = 2 * (precision * recall) / (precision + recall)  if (precision +recall) > 0 else 0
        return f1
```

**Function to select random elements :**

```
import random
def select_elements(lst):
        # Ensure the list is not empty
        if not lst:
                return None # or raise an exception
        # Randomly select at least one element
        num_elements_to_select = random.randint(1, len(lst))
        selected_elements = random.sample(lst, num_elements_to_select)
        return selected_elements

# Evaluate Precision@K, Recall@K, and F1 Score
# Define actual user preferences
```

**Function to Evaluate :**

```
def evaluate(title):
        actual = get_recommendation(title)
        actual = actual[:5]
        actual_preferences = select_elements(actual)

        # Generate recommendations using the recommendation function
        recommended_movies = get_recommendation(title)
        k = 3
        precision = precision_at_k(actual_preferences, recommended_movies, k)
        recall = recall_at_k(actual_preferences, recommended_movies, k)
```

```
    f1 = f1_score(precision, recall)
    print("Precision@{}: {:.4f}".format(k, precision))
    print("Recall@{}: {:.4f}".format(k, recall))
    print("F1 Score: {:.4f}".format(f1))
```

**Evaluation of  the system :**

```
evaluate("Spider-Man")
evaluate("Ice Age")
```

**Output :**

**Overview of the dataset :**

| | id | title | tags |
|---|---|---|---|
| 0 | 19995 | Avatar | action adventure fantasy sciencefiction action... |
| 1 | 285 | Pirates of the Caribbean: At World's End | adventure fantasy action adventure fantasy act... |
| 2 | 206647 | Spectre | action adventure crime action adventure crime ... |
| 3 | 49026 | The Dark Knight Rises | action crime drama thriller action crime drama... |
| 4 | 49529 | John Carter | action adventure sciencefiction action adventu... |

**Testing of the Recommendation system :**

**Recommendation for the movie "The Dark Knight Rises" :**

```
Recommended movies :

The Dark Knight
Batman Begins
Batman Forever
Batman Returns
Batman
Batman
Slow Burn
Inception
Batman: The Dark Knight Returns, Part 2
Batman & Robin
Batman v Superman: Dawn of Justice
JFK
Sin City: A Dame to Kill For
Premium Rush
```

**Recommendation for the movie "Spider Man" :**

```
Recommended movies :

Spider-Man 2
Spider-Man 3
The Amazing Spider-Man 2
The Amazing Spider-Man
Gremlins 2: The New Batch
Arachnophobia
Charlotte's Web
Hook
Small Soldiers
Spawn
Kick-Ass
Mystery Men
Man of Steel
Heavenly Creatures
```

**Recommendation for the movie "Iron Man 3" :**

```
Recommended movies for Iron Man 3 :

Iron Man
Iron Man 2
The Helix… Loaded
Avengers: Age of Ultron
Captain America: Civil War
Star Trek Into Darkness
Cradle 2 the Grave
X-Men
The Walk
The Avengers
The Abyss
Avatar
Transcendence
A Sound of Thunder
```

**Evaluation of  the system :**

**Evaluate the system with movie "Spider Man" :**

```
Precision@3: 0.3333
Recall@3: 0.5000
F1 Score: 0.4000
```

**Evaluate the system with movie "Ice Age" :**

```
Precision@3: 0.6667
Recall@3: 0.5000
F1 Score: 0.5714
```

**Conclusion :**

The development of our content-based recommendation system has shown promising results in providing personalized recommendations to users.This is the final stage of developing and evaluating our personalized content discovery engine. We use similarity recommendation algorithms and thorough evaluation metrics to create a strong system for suggesting personalized content to users. Overall, our content-based recommendation system holds great potential for enhancing user experience and facilitating content discovery in various domains.