# **Movie Recommendation System Documentation**

## **Overview**

This document describes the Movie Recommendation System, focusing on the components and functionality implemented in the provided code snippet. The system utilizes TF-IDF for text feature extraction and cosine similarity for measuring movie similarity.

## **Libraries Used**

* **string**: Provides constants and utility functions for string operations.
* **numpy**: A library for numerical operations, including array and matrix manipulations.
* **pandas**: Used for data manipulation and analysis.
* **matplotlib.pyplot**: Provides functions for creating static, interactive, and animated visualizations.
* **plotly.graph\_objects**: Used for creating interactive plots and charts.
* **plotly.express**: Simplifies the process of creating interactive charts and visualizations.
* **wordcloud**: Generates word clouds from text data.
* **pickle**: Serializes and deserializes Python objects.
* **sklearn.feature\_extraction.text.TfidfVectorizer**: Converts text data into TF-IDF vectors.
* **sklearn.metrics.pairwise.cosine\_similarity**: Computes the cosine similarity between vectors.
* **scipy.sparse.save\_npz**: Saves sparse matrices in compressed NPZ format.
* **warnings**: Used to manage warnings and suppress them if necessary.

### **Load Data**

1. **Purpose**: Reads data from a CSV file named "netflix\_data.csv" into a Pandas DataFrame.
2. **Function**: pd.read\_csv() is used to load data from a CSV file into a DataFrame for further analysis and manipulation.
3. Then we’ll display first five rows of the DataFrame.

**EDA**

– data.describe(include='all').T

Generates and transposes descriptive statistics for all columns in the DataFrame data, including non-numeric columns. This provides a summary of counts, unique values, top values, and frequencies.

**Graph 1**

Counts the number of movies released each year from the release\_year column, sorts them by year, and creates a bar chart using Plotly. The chart shows the number of movies released each year, with blue bars and labeled axes.

**Graph 2**

Counts the occurrences of each content type from the type column and creates a pie chart using Plotly. The chart displays the distribution of different content types with yellow segments and a black font for labels.

**Graph 3**

Identifies the top 10 countries with the highest number of movies from the country column and visualizes this data as a treemap using Plotly. The chart displays the distribution of movies by country with a title indicating the top countries.

**Graph 4**

Calculates the count of each rating type from the rating column and creates a bar chart using Plotly. The chart shows the distribution of ratings with blue bars, labeled axes, and a black font.

**Graph 5**

Creates a word cloud from the movie titles in the title column, visualizing the most common titles with a black background and a coolwarm color map. The word cloud is displayed using Matplotlib with a title indicating "Most Common Titles."

**Pre-Processing of Data**

The Cleantext class provides methods for cleaning categorical text data:

* **separate\_text(text)**: Splits text by commas, trims whitespace, converts to lowercase, and removes duplicates.
* **remove\_space(text)**: Removes spaces from the text and converts it to lowercase.
* **remove\_punc(text)**: Removes punctuation, converts text to lowercase, and trims extra whitespace.
* **clean\_text(text)**: Applies all the above cleaning methods in sequence to produce a cleaned text.

**Model Development**

**TF-IDF Vectorization**

1. **vectorizer = TfidfVectorizer(stop\_words='english')**:
   * **Purpose**: Initializes an instance of the TfidfVectorizer class from scikit-learn.
   * **Parameters**:
     + stop\_words='english': Specifies that common English stop words (e.g., "the", "and") should be excluded from the analysis. This helps to focus on more meaningful terms in the text data.
2. **tfidf\_matrix = vectorizer.fit\_transform(data\_filtered['combined\_text'])**:
   * **Purpose**: Transforms the text data into a matrix of TF-IDF features.
   * **Function**:
     + fit\_transform(): Fits the TfidfVectorizer to the text data and then transforms the text into a TF-IDF matrix.
   * **Input**:
     + data\_filtered['combined\_text']: The column from the DataFrame data\_filtered containing the text data to be vectorized.
   * **Output**:
     + tfidf\_matrix: A sparse matrix where each row represents a document and each column represents a term from the text data. The values in the matrix are the TF-IDF scores for each term in each document.

This process converts the text data into a numerical format suitable for machine learning algorithms by calculating the Term Frequency-Inverse Document Frequency (TF-IDF) scores, which reflect the importance of terms relative to the entire dataset.

1. **cosine\_sim = cosine\_similarity(tfidf\_matrix, tfidf\_matrix)**:
   * **Purpose**: Computes the cosine similarity between the TF-IDF vectors of all documents.
   * **Function**:
     + cosine\_similarity(): Calculates the cosine similarity between pairs of vectors in the TF-IDF matrix. Cosine similarity measures the cosine of the angle between two vectors, which provides a measure of their similarity.
   * **Input**:
     + tfidf\_matrix: The matrix of TF-IDF features where each row represents a document and each column represents a term.
   * **Output**:
     + cosine\_sim: A square matrix where each element (i, j) represents the cosine similarity between the i-th and j-th documents. Values range from 0 (no similarity) to 1 (identical).

This computation is used to assess the similarity between documents based on their content. A higher cosine similarity indicates that the documents are more similar in terms of their TF-IDF feature vectors.

1. **class Recommender**:
   * **Purpose**: Defines a recommendation system for suggesting similar movies and TV shows based on a given title.
2. **\_\_init\_\_(self, data\_rec, cosine\_sim)**:
   * **Purpose**: Initializes the Recommender class with the dataset and cosine similarity matrix.
   * **Parameters**:
     + data\_rec: DataFrame containing movie and TV show data with titles and types.
     + cosine\_sim: Matrix of cosine similarity scores between items.
3. **recommendation(self, title, total\_result=5, threshold=0.5)**:
   * **Purpose**: Provides recommendations based on a given title.
   * **Parameters**:
     + title: The title of the item for which recommendations are to be generated.
     + total\_result: Number of top results to return (default is 5).
     + threshold: Minimum similarity score required to consider an item (not used in current implementation).
   * **Function**:
     + Finds the index of the given title using find\_id().
     + Computes similarity scores and sorts items by similarity.
     + Separates recommendations into movies and TV shows.
     + Formats and returns lists of similar movies and TV shows.
4. **find\_id(self, name)**:
   * **Purpose**: Finds the index of the item whose title matches the given name.
   * **Parameters**:
     + name: The title to search for.
   * **Function**:
     + Searches for a title that contains the given name using regular expressions.
     + Returns the index of the first matching title, or -1 if not found.

This class provides a basic recommendation system based on similarity scores, allowing users to find and list similar movies and TV shows.

**Model Optimization**

1. **model = NearestNeighbors()**:
   * **Purpose**: Initializes the NearestNeighbors model from scikit-learn, which is used for finding the nearest neighbors in the feature space.
2. **param\_grid**:
   * **Purpose**: Defines a grid of parameters to search over during model tuning.
   * **Parameters**:
     + 'n\_neighbors': List of numbers of neighbors to use for finding the nearest neighbors (e.g., 5, 10, 15).
     + 'algorithm': List of algorithms to use for computing nearest neighbors ('auto', 'ball\_tree', 'kd\_tree', 'brute').
     + 'p': List of values for the Minkowski distance parameter, where p=1 corresponds to Manhattan distance and p=2 corresponds to Euclidean distance.
3. **grid\_search = GridSearchCV(model, param\_grid, cv=3, scoring='accuracy')**:
   * **Purpose**: Sets up a GridSearchCV object to perform an exhaustive search over the specified parameter grid using cross-validation.
   * **Parameters**:
     + model: The NearestNeighbors model to be tuned.
     + param\_grid: The parameter grid to search.
     + cv=3: Number of cross-validation folds (3-fold cross-validation).
     + scoring='accuracy': Metric used to evaluate the model's performance. (Note: 'accuracy' is not typically used for nearest neighbors; consider using metrics like 'precision', 'recall', or 'f1' for classification problems.)
4. **grid\_search.fit(tfidf\_matrix)**:
   * **Purpose**: Fits the GridSearchCV object to the TF-IDF matrix, performing the search and evaluating different parameter combinations.
5. **best\_model = grid\_search.best\_estimator\_**:
   * **Purpose**: Retrieves the best model found during the grid search, based on cross-validated performance.

This code sets up and runs a grid search to optimize the parameters of a NearestNeighbors model using cross-validation, ultimately selecting the best-performing model based on the specified metrics.

1. **class Recommender\_opt**:
   * **Purpose**: Provides an optimized recommendation system using nearest neighbors.
2. **\_\_init\_\_(self, data\_rec, model, tfidf\_matrix)**:
   * **Purpose**: Initializes the Recommender\_opt class with necessary data and model.
   * **Parameters**:
     + data\_rec: DataFrame containing movie and TV show data with titles and types.
     + model: A trained NearestNeighbors model used for finding similar items.
     + tfidf\_matrix: TF-IDF matrix of item descriptions for similarity computation.
3. **recommendation\_opt(self, title, total\_result=5)**:
   * **Purpose**: Provides optimized recommendations based on a given title.
   * **Parameters**:
     + title: The title of the item to find similar items for.
     + total\_result: Number of top recommendations to return (default is 5).
   * **Function**:
     + Finds the index of the given title using find\_id().
     + Uses the nearest neighbors model to find similar items based on the TF-IDF matrix.
     + Converts distances to similarity scores (1 - distance).
     + Creates a DataFrame of recommended items, sorted by similarity.
     + Separates recommendations into movies and TV shows.
     + Formats and returns lists of similar movies and TV shows.
4. **find\_id(self, name)**:
   * **Purpose**: Finds the index of the item whose title matches the given name.
   * **Parameters**:
     + name: The title to search for.
   * **Function**:
     + Searches for a title containing the given name using regular expressions with case-insensitivity.
     + Returns the index of the first matching title, or -1 if not found.

This class enhances the recommendation system by efficiently finding and returning similar items based on nearest neighbor search and TF-IDF features, with separate lists for movies and TV shows.

**Evaluation of Model**

1. **calculate\_intra\_list\_diversity(tfidf\_matrix, recommended\_indices)**:
   * **Purpose**: Computes the intra-list diversity of a list of recommended items based on their TF-IDF vectors.
   * **Parameters**:
     + tfidf\_matrix: The matrix of TF-IDF feature vectors for all items.
     + recommended\_indices: A list of indices for the recommended items.
   * **Function**:
     + **Extract TF-IDF Vectors**: Retrieves the TF-IDF vectors for the items at the specified indices.
     + **Compute Similarity Matrix**: Calculates the pairwise cosine similarity between the recommended items using cosine\_similarity().
     + **Calculate Diversity**: Computes the intra-list diversity by averaging the cosine similarity scores and subtracting from 1. This metric measures how dissimilar the items in the recommendation list are from each other.
     + **Return**: The diversity score, where a higher score indicates greater diversity.
2. **Example Usage**:
   * **recommended\_indices**: A sample list of indices for which the diversity score is calculated.
   * **diversity\_score**: The result of the diversity calculation, printed to show the intra-list diversity of the recommended items.

This function helps in evaluating the diversity of a recommendation list by quantifying how varied the recommended items are with respect to each other, which can be crucial for enhancing user satisfaction with the recommendations.