SVKM’s NMIMS

Mukesh Patel School of Technology, Management and Engineering

Department of Computer Engineering

B.Tech\MBA.Tech VI semester Project Submission

Subject Machine Learning

**Project Title:** Document retrieval from Wikipedia Articles

**Project Team Members:**

|  |  |
| --- | --- |
| **Roll No** | **Name** |
| N061 | Divyanshu Sharma |
| N064 | Bhavyaraj Singh |
| N069 | Anshuman Thapliyal |

1. **Project Description**

Aim of the project is display the top five related articles on a famous person to the user based on their present search article from the corpus. While also showing the current article about the famous person along with a visual depiction in the form of word cloud.

Document retrieval from Wikipedia articles using unsupervised classification is a technique that involves automatically categorizing Wikipedia articles into groups based on their content. The process involves several steps, including data pre-processing, feature extraction, clustering, and evaluation. First, the Wikipedia articles are cleaned and transformed into a suitable format for analysis using count vectorization. Next, relevant features such as the frequency of words or the presence of specific keywords are extracted from the text using TF-IDF. Then, unsupervised clustering algorithms such as k-means clustering are applied to group the articles based on the similarity of their features. Finally, the quality of the clusters is evaluated using cosine distance matrices.

Steps:

1. Pre-process the text data: Before applying any clustering algorithm, the text data needs to be pre-processed to remove any noise or irrelevant information. This can be done by converting the text data to lowercase, removing stop words, punctuations, and performing stemming or lemmatization.
2. Calculate the TF-IDF scores: After pre-processing the text data, the next step is to calculate the TF-IDF scores for each term in the documents. TF-IDF stands for term frequency-inverse document frequency, which is a statistical measure used to evaluate the importance of a term in a document.
3. Convert the TF-IDF scores into a vector representation: The TF-IDF scores for each document can be represented as a vector, where each dimension of the vector represents the TF-IDF score of a particular term in the document.
4. Calculate the cosine similarity matrix: After converting the TF-IDF scores into a vector representation, the next step is to calculate the cosine similarity between each pair of documents. The cosine similarity is a measure of similarity between two vectors in a high-dimensional space.
5. Apply the KNN algorithm: Finally, apply the KNN algorithm to cluster the documents based on their similarity in terms of cosine distance. The KNN algorithm works by selecting the K nearest neighbours to a particular document based on their cosine similarity, and then assigning the document to the cluster with the highest number of nearest neighbours.
6. **Type of problem: (supervised\Unsupervised), (Prediction\Classification). Why the problem falls into particular category**

Unsupervised Classification

It is unsupervised as there are no labels and we are trying to identify patterns and group similar objects in a dataset without prior knowledge of their labels or categories. This approach does not require any labelled data or prior knowledge of the categories, but rather relies on the similarity of the articles' textual features.

Here the inputs are articles from the corpus and give the output as cluster labels

1. **Python Libraries used:**
2. import pandas as pd
3. import numpy as np
4. from sklearn.feature\_extraction.text import CountVectorizer
5. from sklearn.feature\_extraction.text import TfidfTransformer
6. import matplotlib.pyplot as plt
7. from wordcloud import WordCloud
8. from sklearn.metrics.pairwise import cosine\_distances
9. from sklearn.neighbors import KNeighborsClassifier
10. **Data set details: Describe data set in detail. Description must be in terms of number of rows, features, meaning of features**

* Dataset used: people\_wiki.csv
* There are 3 columns and 59071 rows in the dataset
* Data contains:

1. URI: link to Wikipedia article
2. Name: name of famous person
3. Text: text of article(text).

* All columns are object type

1. **Data Cleaning steps (Describe in detail data cleaning steps under taken. If required along with code)**



Figure : code and comments for data cleaning

1. **Data Pre-processing steps (Describe in detail data pre-processing steps under taken. If required along with code)**

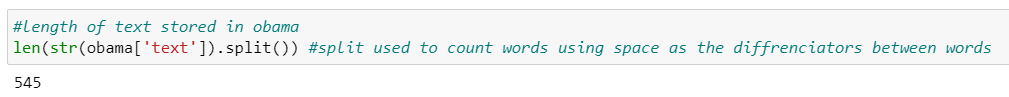


Figure : Converting text to string to use split function to count number of words in the article

For TF-IDF we need to pre-process the data i.e the text data by removing stop words, punctuation, and converting all the words to lowercase to reduce noise and make the text data consistent.

1. **Feature scaling\Normalization (if applied) – Which technique implemented why?**

* **Count vectorization**: all words have the same weight causing problem in finding similarity

Steps:

1. Tokenize the text: Tokenization is the process of breaking up a text into smaller units called tokens.
2. Create a vocabulary: create a vocabulary of all the unique words in the text.
3. Count the frequency of each word: For each document, count the number of times each word in the vocabulary appears in the document. This will create a frequency vector, where each element represents the frequency of a word in the document.
4. Convert the frequency vector to a Bag of Words: use a technique called "flattening". This involves concatenating all the elements of the frequency vector to create a single long vector.
5. Repeat for each document: Repeat steps 3-4 for each document in your corpus to create a matrix of Bag of Words, where each row represents a document and each column represents a word in the vocabulary.
6. Use the Bag of Words for downstream tasks: Once you have created the Bag of Words matrix, you can use it for downstream NLP tasks such as text clustering.

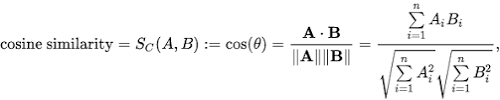
* **TF-IDF(Term frequency – inverse document frequency):**

Using log help in balancing out the weights assigned to the words. Words common in large number of articles will make the denominator large so the value will be close to log 1 i.e 0, while rare words will have a small denominator so making the log value a high number. This is how rare words get greater weights.

Steps:

1. Tokenization: splitting the text data into smaller chunks such as words
2. Term Frequency (TF): calculate the frequency of each term in the document. The term frequency (TF) is calculated as the number of times a term appears in a document divided by the total number of terms in the document.
3. Inverse Document Frequency (IDF): IDF is calculated as the logarithm of the total number of documents divided by the number of documents that contain the term. The IDF value decreases as the number of documents that contain the term increases. This step helps to give more weight to the terms that are less frequent and more informative.
4. TF-IDF Calculation: The final step is to calculate the TF-IDF score for each term in the document. The TF-IDF score is calculated by multiplying the TF value with the IDF value for each term in the document.
5. **Model building- Describe in detail what all models were build. If required put sample code for model building**

* **Cosine similarity and cosine distance**
* Calculate the cosine similarity between the two vectors using the following formula:



where A and B are the two vectors being compared, "." represents the dot product, and "||A||" and "||B||" represent the magnitudes of A and B, respectively.

* The cosine distance is simply the complement of the cosine similarity, i.e.,

cosine distance = 1 - cosine similarity

This gives a measure of the distance between the two vectors, with 0 meaning they are identical and 1 meaning they are completely dissimilar.

* **K-Nearest Neighbour Model**

Query article: Input by the user for a famous person

Corpus: dataset (people\_wiki.csv)

Specify: Cosine distance metric

Output: Set of most similar articles

Algorithm: -

1. Search over each article in corpus
2. Compute s = similarity(search article , corpus article)
3. If s > Best\_s, record[best fit search] = article in loop

and set Best\_s = s

1. Return record[best fit search]
2. **Model evaluation**

Comparison between using TF-IDF over Count Vectorization

Count Vectorization represents a document as a bag of words by counting the frequency of each word in the document. It creates a matrix where each row represents a document and each column represents a unique word in the corpus. The matrix is often high-dimensional and sparse, meaning that most of the values are zero. Count Vectorization is simple and efficient but does not take into account the importance of each word in the document.

TF-IDF, on the other hand, aims to reflect the importance of words in a document. It multiplies the term frequency of a word by the inverse document frequency, which measures how rare the word is in the corpus. The resulting value is a weight that reflects how important the word is in the document. Words that are common in many documents, such as "the" and "and", are down-weighted, while words that are rare in the corpus, but appear frequently in the document, are up-weighted. TF-IDF produces a matrix where each row represents a document, and each column represents a unique word in the corpus. The resulting matrix is also often sparse but less so than the Count Vectorization matrix.

1. Sparsity: Count Vectorization creates a high-dimensional sparse matrix, whereas TF-IDF typically creates a lower dimensional, but still sparse matrix.
2. Weighting: Count Vectorization only considers the frequency of the word in the document, while TF-IDF also considers the frequency of the word in the corpus.
3. Stop words: Count Vectorization typically removes stop words, while TF-IDF may keep them, as they may be informative.
4. Normalization: Count Vectorization does not normalize the values, while TF-IDF normalizes the values.
5. **Attach collab code file (ipynb file as object)**



1. **Conclusion**

From the about steps we have made an user interactive article retrieval system.

The user inputs in the search bar the famous person it wants to read about.

As the output the article text is displayed. A word cloud is created to show the important words of that article. In the end, top 5 articles most closely related to the search person which is already present in the corpus are displays with their URL and name of the person.