Content-based Recommender System on MIND-small Dataset

1st Anna Patricia B. Desembrana
College of Computer Studies
De La Salle University
Manila, Philippines
anna_desembrana@dlsu.edu.ph

2nd Ina Patricia Evangelista
College of Computer Studies
De La Salle University
Manila, Philippines
ina_evangelista@dlsu.edu.ph

3rd John Matthew O. Gan
College of Computer Studies
De La Salle University
Manila, Philippines
john_matthew_gan@dlsu.edu.ph

4th Clarissa Mae S. Mandadero
College of Computer Studies
De La Salle University
Manila, Philippines
clarissa_mandadero@dlsu.edu.ph

***Abstract*—Content-based filtering is one of the ways that recommender systems can offer various firms a means to provide or recommend services and products in accordance with user requirements. It uses the features of items to build a user profile and to provide recommendations based on that user profile. Experiments were conducted using the MIND-small dataset, a dataset which contains a random sampling of 50,000 users with their corresponding user interaction behavior logs on the Microsoft News service. Two similarity measures were tested using two methods for statistical representation. There are four test cases included in this paper to determine the best method and similarity measure for providing recommendations. Results show that cosine similarity with TF-IDF based model is a suitable combination for a content-based recommender system.**

***Index Terms*—Content-based filtering, Recommender system, TF-IDF, Bag-of-Words, Cosine similarity, Euclidean distance**

I. Introduction

Content-based filtering is a technique used in creating recommender systems that use features and users' past behavior to create a user profile as a basis for recommending items. This user profile is then used by comparing the features of new items to generate recommendations of similar items that the involved user has liked in the past [1]. The main difference of content-based filtering from collaborative filtering is that content-based filtering does not require other users' data to create recommendations, while collaborative filtering requires the features of items to be given by other users to create a recommender system [2].

The main objective of this study is to use a content-based filtering approach for the creation of a recommender system. Moreover, this study aims to compare the effects of using different similarity measures in improving the recommender system.

The dataset that will be used is the Microsoft News Dataset (MIND). MIND contains news recommendations from anonymized behavior logs of random users in the Microsoft News service who had at least 5 click interactions from October 12, 2022 to November 19, 2022. MIND consists of 1 million users and 161,013 English news articles [3]. Considering that MIND is a large dataset, only the small version of MIND will be used in this study which is called the MIND-small dataset. The MIND-small dataset only contains a random sampling of 50,000 users together with their behavior logs, all of which comes from MIND.[1]

The MIND dataset contains four different files: (1) behaviors.tsv, (2) news.tsv, (3) entity_embedding.vec, and (4) relation_embedding.vec. The file behaviors.tsv contains the click histories and impressions logs of users in the dataset. The news.tsv file contains information of various news articles. The entity_embedding.vec file contains embeddings of the different entities in news extracted from knowledge graph. Lastly, the relation_embedding.vec file contains the embeddings of relations between entities extracted from knowledge graph[1].

Content-based recommender systems are prone to issues, such as scalability and overspecialization. In scalability, the number of items and users grows quickly which results in a large-scale data.

Scalability becomes an issue for content-based recommender systems because every time a user or item is added, its attributes must be defined and tagged by the system to be used for recommendations [4]. Thus, the continuous need to assign attributes becomes a challenge for content-based recommender systems [5].

Overspecialization, on the other hand, is a problem wherein recommended items are too similar to each other. This is because users are more likely to lose interest in the service or applications when they get tired of seeing the same recommended items every time. Overspecialization calls for the need to provide diverse recommendations where the items being

[1]https://github.com/msnews/msnews.github.io/blob/master/assets/doc/introduction.md.

recommended are not similar to one another, but are still of interest to the user [6].

This paper is outlined as follows: the next section, Section II, will discuss related literature and studies that used content-based filtering in the creation of the recommender system. Section III will then discuss the proposed approach and how it was used to create a content-based recommender system for the MIND-small dataset. Section IV will discuss the various experiments conducted, and the results extracted from the experiments. Lastly, Section V will present the conclusion of this study.

II. Review of Related Literature

In this section, various related literature and studies will be discussed which might be useful for the objective of this study, which is to create a content-based recommender system for the MIND-small dataset. The approaches used in the papers will also be studied and discussed. Moreover, the similarity measures used by the recommender systems created by other researchers will be discussed, as this is also another objective of this study.

A content-based image retrieval system was created by [7]. To create the system, first, images are split into training and validation sets by using a Support Vector Machine (SVM). Afterward, Speeded Up Robust Features (SURF) are extracted from the images which are then represented as a bag of visual words through k-means clustering and image indexing. Lastly, images are retrieved using cosine similarity by estimating distance between the query image feature vectors and image index feature vector. The created system resulted in an image retrieval accuracy that ranges between 80% to 90%. The study of [7] can be useful for this study as it shows how Bag of Words (BoW) was utilized to improve the recommender system since BoW is one of the approaches that will be explored in this study.

Four different movie recommender systems were proposed by [8]: a Jaccard recommender model, a Count Vectorizer utilizing cosine similarity, and a Term Frequency-Inverse Document Frequency (TF-IDF) model utilizing cosine similarity for keywords and plot overview. For the combination of TF-IDF and cosine similarity model, the frequency of a word's appearance in a document as well as in other documents within a corpus was utilized to determine the word's value. Due to the independence of cosine similarity from scale and relative simplicity and efficiency of calculation, cosine similarity was utilized to determine a percentage that quantifies the similarity between movies. Using the similarity score for a given movie, TF-IDF for keywords ranked 2nd (24.26%) for resulting the best score and TF-IDF for plot overview (15.93%) as the worst. The implementation of TF-IDF by [8] might be useful for this study as it can give insights into using this method for a recommender system, as well as its performance for a given similarity measure. This can give way for the study to explore similarity measures that can provide better performance for the system.

A group recommender system was proposed which is based on the affective social network that was extracted from social big data [9]. In this recommender system, the focus is giving recommendations to a group of users rather than just one individual. Various similarity measures were used to compare its effects in improving the recommender system. The similarity measures used were cosine similarity, Euclidean distance, Mahalanobis distance, and Euclidean norm. It was discovered that cosine similarity and Mahalanobis distance performed the best out of the four similarity measures since they had the lowest Mean Standard Evaluation (MSE) with 0.007 and 0.008, respectively. Upon further experiments, it was discovered that Mahalanobis distance performs the best when balanced and unbalanced relationships were also considered for the group recommendation. A balanced relationship means that one person greatly relies on another person. While unbalanced relationship means that two users have the same similar value. At the end of the study, the created group recommender system exhibited superior performance since its performance was stable even if there was a decline in the growth of group sizes [9]. The study of [9] compared the effects of using various similarity measures in a recommender system. Thus, it [9] can be used as the basis when using different similarity measures for the experiments of this study.

III. Proposed Approaches

A. Dataset

The MIND-small dataset contains a random sampling of 50,000 users together with their user interaction behavior logs. It is a variant of the MIND dataset created by [3] with a much smaller sample size. MIND consists of 1 million users and 161,013 English news articles which were recorded from October 12, 2022 to November 19, 2022.

The recommender system uses the news.tsv file of the dataset as its data source, which contains entries of news articles and their metadata. More specifically, it contains the following columns: (1) News ID, (2) Category, (3) SubCategory, (4) Title, (5) Abstract, (6) Uniform Resource Locators (URL), (7) Title Entities, and (8) Abstract Entities. For the sake of this study, only four columns will be used, namely (1) Category, (2) SubCategory, (3) Title, and (4) Abstract.

Various data visualizations were used to present different findings on the MIND-Small dataset. Fig. 1 shows the highest number of categories used in the dataset are news, sports, and weather, while the highest number of sub-categories are newsus, football_nfl, and newspolitics. Additionally, most titles of news articles in the database are discovered to fall within the length of 50 to 100 characters, as seen in Fig. 2.

B. Data Pre-Processing

The data was pre-processed to check for any inconsistencies that would be removed through data cleaning techniques. Duplicate entries in the dataset are checked and removed by comparing the titles of the news articles. As seen in Fig. 3, it was discovered that there are 593 news articles with the same

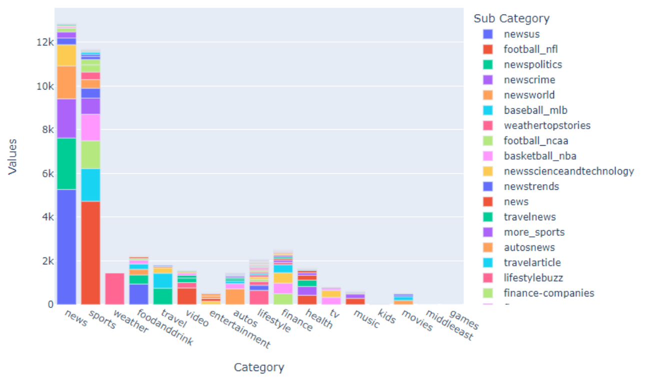
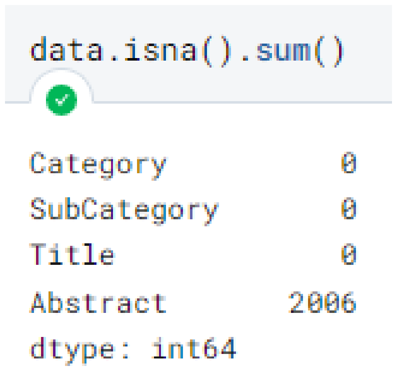


Fig. 1. Number of categories and sub-categories



```
data.isna().sum()
```

```
Category         0
SubCategory      0
Title            0
Abstract      2006
dtype: int64
```

Fig. 4. Checking for null values

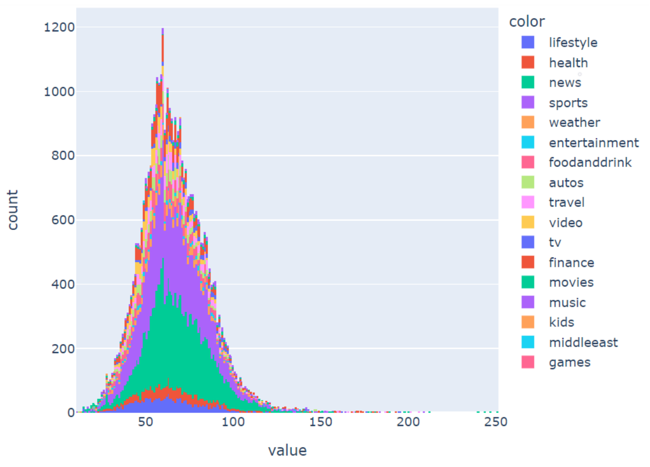
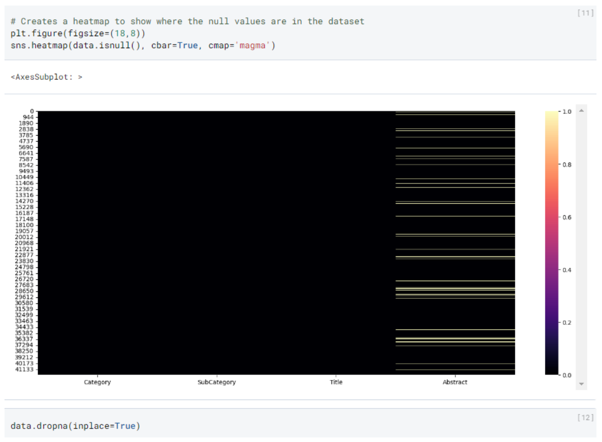


Fig. 2. Length of titles



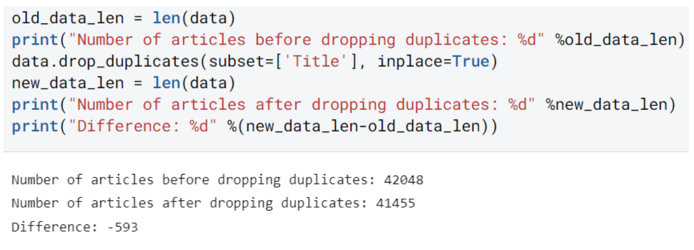
```
# Creates a heatmap to show where the null values are in the dataset
plt.figure(figsize=(18,8))
sns.heatmap(data.isnull(), cbar=True, cmap='magma')
```

```
<AxesSubplot: >
```

```
data.dropna(inplace=True)
```

Fig. 5. Dropping of entries with null values

titles. All entries with duplicate titles are then dropped from the dataset by calling the drop_duplicates() method in Pandas.



```
old_data_len = len(data)
print("Number of articles before dropping duplicates: %d" %old_data_len)
data.drop_duplicates(subset=['Title'], inplace=True)
new_data_len = len(data)
print("Number of articles after dropping duplicates: %d" %new_data_len)
print("Difference: %d" %(new_data_len-old_data_len))
```

```
Number of articles before dropping duplicates: 42048
Number of articles after dropping duplicates: 41455
Difference: -593
```

Fig. 3. Dropping of duplicate entries

All columns of the dataset are then checked for empty, null, or NaN values by calling the isna() method provided by Pandas. As seen in Fig. 4, it was discovered that the dataset's abstract column contains 2,006 rows with null or empty values. These entries were dropped from the dataset by calling the Pandas dropna() method, as seen in Fig. 5.

Through the data pre-processing, all data have been checked for inconsistencies. All found inconsistencies have been cleaned and fixed. Thus, the data from the MIND-small dataset can now be used for the creation of a recommender system.

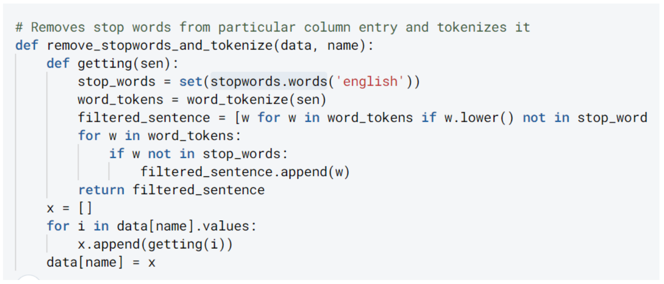
C. *Text Pre-Processing*

The aim of text pre-processing is to single out unique words and remove unnecessary words from the dataset. This improves the performance of the recommender system since it sorts out the words in the dataset and only keeps words that are vital in creating recommendations for the users. To perform text pre-processing, the Natural Language Toolkit (NLTK) module was used. NLTK is a module for Python that is used when working with human language data. It contains a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum[2].

The first task performed in text-preprocessing is removing stop words from a particular column entry which is then tokenized. Tokenizing means splitting sentences or phrases into words [10]. Stop words refer to words that are usually used in any language, examples of stop words are prepositions, such as of, about, by, as, and in. Stop words are commonly eliminated in Natural Language Processing to eliminate commonly used words that are not useful for data processing [11].

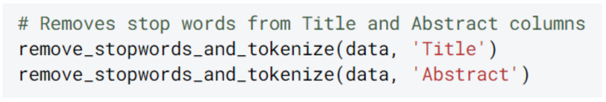
[2]https://www.nltk.org/

The function that handles the removal of stop words is remove_stopwords_and_tokenize(), as seen in Fig. 6. In the remove_stopwords_and_tokenize() function, the stop-words.words('english') from NLTK was called which contains all stop words in the English language[3]. Each word in the dataset is then checked to see if it is a stop word. If it is a stop word, then the word is filtered or deleted from the dataset. Only the columns which contain the title and abstract of the news articles had its stop words removed, as seen in Fig. 7. After the removal of stop words, each entry in the title and abstract columns are then tokenized to prepare it for lemmatization which is the next step in text pre-processing.



```
# Removes stop words from particular column entry and tokenizes it
def remove_stopwords_and_tokenize(data, name):
    def getting(sen):
        stop_words = set(stopwords.words('english'))
        word_tokens = word_tokenize(sen)
        filtered_sentence = [w for w in word_tokens if w.lower() not in stop_word
        for w in word_tokens:
            if w not in stop_words:
                filtered_sentence.append(w)
        return filtered_sentence
    x = []
    for i in data[name].values:
        x.append(getting(i))
    data[name] = x
```

Fig. 6. Function for removing stop words



```
# Removes stop words from Title and Abstract columns
remove_stopwords_and_tokenize(data, 'Title')
remove_stopwords_and_tokenize(data, 'Abstract')
```

Fig. 7. Removal of stop words in title and abstract columns

Lemmatization refers to the process of converting a word to its base form, for example, the lemmatized word of rocks is rock and the lemmatized word of dancing is dance [10]. Through lemmatization, it is easier to group together similar words that were originally in different forms which allows the recommender system to analyze the different forms of the word as a single item.

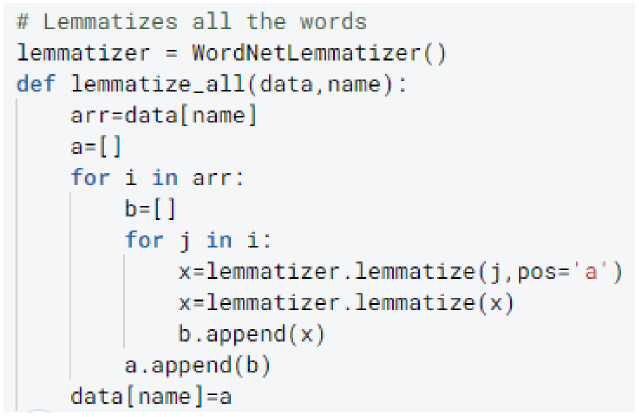
To perform lemmatization, the WordNetLemmatizer() from NLTK was used, as seen in Fig. 8. Afterward, lemmatization was performed on the title and abstract columns by calling the lemmatize_all() function, as seen in Fig. 9. Through calling the lemmatize_all() function, each tokenized word from all entries in the title and abstract columns are converted into its base form.

D. BoW Method

BoW is a method used for feature extraction using text data [12]. BoW turns arbitrary text into fixed-length vectors by counting the number of occurrences of a specific word [13]. In BoW, the position of words is ignored, and only the frequency count of the words in the document is taken into consideration [14].

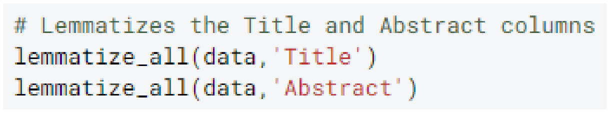
To implement BoW, the CountVectorizer() function from Scikit-learn (Sklearn) was used, as seen in Fig. 10. The

[3]https://gist.github.com/sebleier/554280



```
# Lemmatizes all the words
lemmatizer = WordNetLemmatizer()
def lemmatize_all(data,name):
    arr=data[name]
    a=[]
    for i in arr:
        b=[]
        for j in i:
            x=lemmatizer.lemmatize(j,pos='a')
            x=lemmatizer.lemmatize(x)
            b.append(x)
        a.append(b)
    data[name]=a
```

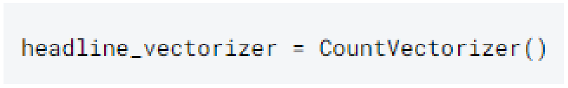
Fig. 8. Function for lemmatization of words



```
# Lemmatizes the Title and Abstract columns
lemmatize_all(data,'Title')
lemmatize_all(data,'Abstract')
```

Fig. 9. Lemmatization of words in title and abstract columns

CountVectorizer() function converts a collection of text documents to a matrix of token counts [15]. Through this, a sparse representation of the number of words and its frequency count is produced. After getting the frequency count of each word through a CountVectorizer(), a similarity measure is then incorporated to the BoW to create a recommender system that can recommend titles of news articles that are similar to the title that was given by the user.



```
headline_vectorizer = CountVectorizer()
```

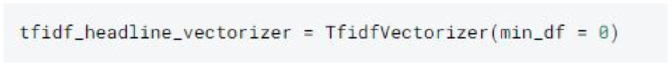
Fig. 10. Implementation of BoW using CountVectorizer()

E. TF-IDF Method

TF-IDF is used in information retrieval through the weighing of co-occurrence matrices, and is based on BoW. It has applications in many other areas of NLP since it is a great baseline method [14]. It is the product of the term frequency (TF) and the inverse document frequency (IDF).

The first factor is TF, which is the frequency of the word in the document. The second factor is IDF, which assigns lower weights to words that occur frequently and assigns higher weights to words that occur infrequently [16]. Terms that are specific to a few number of documents can be used as the distinguishing features in a whole collection. On the other hand, terms that are used often throughout the entire collection are hardly a representation of the collection [14].

To implement the TF-IDF method, the TfidfVectorizer() function from Sklearn was used, as seen in Fig. 11. This function converts a collection of raw documents to a matrix of TF-IDF features [17]. After the vectorizer object is created, it will be used to fit and transform the text data into vectors. Then, it will be used to extract the names of the titles.



```
tfidf_headline_vectorizer = TfidfVectorizer(min_df = 0)
```

Fig. 11. Implementation of TF-IDF using TfidfVectorizer()

IV. Results and Discussion

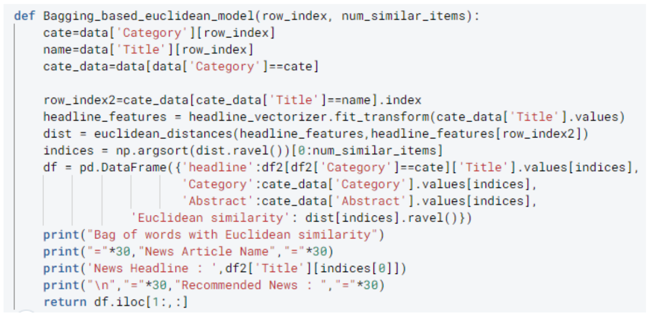
A. Set-up of the Experiments

Two methods were used for the creation of the recommender system which are BoW and TF-IDF. After the implementation of the two methods, two similarity measures were incorporated to both methods: Euclidean distance and cosine similarity. To summarize, a total of four test cases were used for the experiments. The four test cases are the following:

- Test Case #1: BoW based model using Euclidean distance
- Test Case #2: BoW based model using cosine similarity
- Test Case #3: TF-IDF based model using Euclidean distance
- Test Case #4: TF-IDF based model using cosine similarity

B. Implementation of the Experiments

To implement test cases #1 and #3, which both use Euclidean distance, the euclidean_distances() function from Sklearn was used. The euclidean_distances() function computes the distance matrix between each pair from a vector array X and Y. [18]. After computing for the Euclidean distance, it is then incorporated to the BoW based model to create recommendations. The code used to implement test case #1 can be seen in Fig. 12, while the code used to implement test case #3 can be seen in Fig. 13.

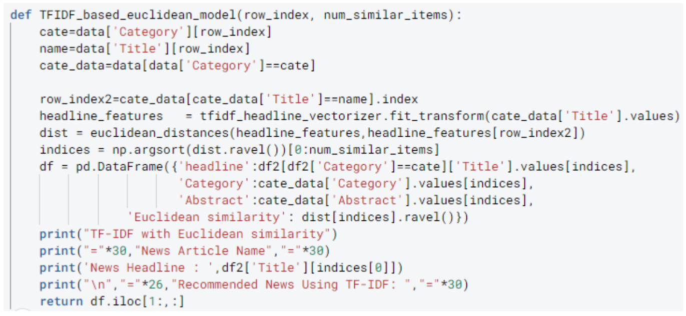


```
def Bagging_based_euclidean_model(row_index, num_similar_items):
    cate=data['Category'][row_index]
    name=data['Title'][row_index]
    cate_data=data[data['Category']==cate]

    row_index2=cate_data[cate_data['Title']==name].index
    headline_features = headline_vectorizer.fit_transform(cate_data['Title'].values)
    dist = euclidean_distances(headline_features,headline_features[row_index2])
    indices = np.argsort(dist.ravel())[0:num_similar_items]
    df = pd.DataFrame({'headline':df2[df2['Category']==cate]['Title'].values[indices],
                       'Category':cate_data['Category'].values[indices],
                       'Abstract':cate_data['Abstract'].values[indices],
                    'Euclidean similarity': dist[indices].ravel()})
    print("Bag of words with Euclidean similarity")
    print("="*30,"News Article Name","="*30)
    print('News Headline : ',df2['Title'][indices[0]])
    print("\n","="*30,"Recommended News : ","="*30)
    return df.iloc[1:,:]
```

Fig. 12. Code implementation of test case #1

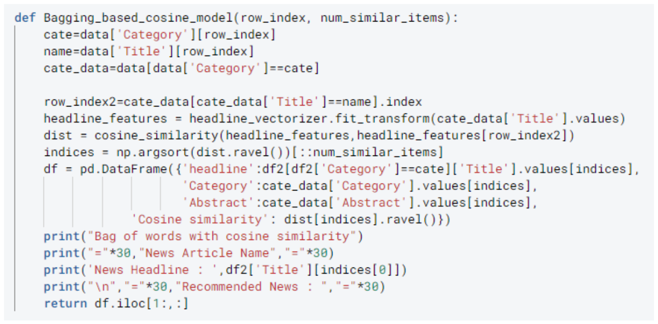
On the other hand, test cases #2 and #4, which both used cosine similarity, were implemented by using the cosine_similarity() function from Sklearn. The cosine_similarity() function computes for the cosine similarity of samples x and y [19]. After computing for the cosine similarity, it is then incorporated to the TF-IDF based model to create recommendations. The code used to implement test case #2 can be seen in Fig. 14, while the code used to implement test case #4 can be seen in Fig. 15.



```
def TFIDF_based_euclidean_model(row_index, num_similar_items):
    cate=data['Category'][row_index]
    name=data['Title'][row_index]
    cate_data=data[data['Category']==cate]

    row_index2=cate_data[cate_data['Title']==name].index
    headline_features   = tfidf_headline_vectorizer.fit_transform(cate_data['Title'].values)
    dist = euclidean_distances(headline_features,headline_features[row_index2])
    indices = np.argsort(dist.ravel())[0:num_similar_items]
    df = pd.DataFrame({'headline':df2[df2['Category']==cate]['Title'].values[indices],
                       'Category':cate_data['Category'].values[indices],
                       'Abstract':cate_data['Abstract'].values[indices],
                  'Euclidean similarity': dist[indices].ravel()})
    print("TF-IDF with Euclidean similarity")
    print("="*30,"News Article Name","="*30)
    print('News Headline : ',df2['Title'][indices[0]])
    print("\n","="*26,"Recommended News Using TF-IDF: ","="*30)
    return df.iloc[1:,:]
```

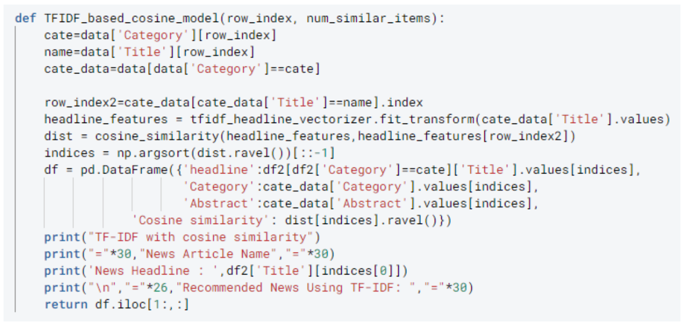
Fig. 13. Code implementation of test case #3



```
def Bagging_based_cosine_model(row_index, num_similar_items):
    cate=data['Category'][row_index]
    name=data['Title'][row_index]
    cate_data=data[data['Category']==cate]

    row_index2=cate_data[cate_data['Title']==name].index
    headline_features = headline_vectorizer.fit_transform(cate_data['Title'].values)
    dist = cosine_similarity(headline_features,headline_features[row_index2])
    indices = np.argsort(dist.ravel())[::num_similar_items]
    df = pd.DataFrame({'headline':df2[df2['Category']==cate]['Title'].values[indices],
                       'Category':cate_data['Category'].values[indices],
                       'Abstract':cate_data['Abstract'].values[indices],
                  'Cosine similarity': dist[indices].ravel()})
    print("Bag of words with cosine similarity")
    print("="*30,"News Article Name","="*30)
    print('News Headline : ',df2['Title'][indices[0]])
    print("\n","="*30,"Recommended News : ","="*30)
    return df.iloc[1:,:]
```

Fig. 14. Code implementation of test case #2



```
def TFIDF_based_cosine_model(row_index, num_similar_items):
    cate=data['Category'][row_index]
    name=data['Title'][row_index]
    cate_data=data[data['Category']==cate]

    row_index2=cate_data[cate_data['Title']==name].index
    headline_features = tfidf_headline_vectorizer.fit_transform(cate_data['Title'].values)
    dist = cosine_similarity(headline_features,headline_features[row_index2])
    indices = np.argsort(dist.ravel())[::-1]
    df = pd.DataFrame({'headline':df2[df2['Category']==cate]['Title'].values[indices],
                       'Category':cate_data['Category'].values[indices],
                       'Abstract':cate_data['Abstract'].values[indices],
                  'Cosine similarity': dist[indices].ravel()})
    print("TF-IDF with cosine similarity")
    print("="*30,"News Article Name","="*30)
    print('News Headline : ',df2['Title'][indices[0]])
    print("\n","="*26,"Recommended News Using TF-IDF: ","="*30)
    return df.iloc[1:,:]
```

Fig. 15. Code implementation of test case #4

C. Evaluation of Models

The metric that was used to evaluate the performance of each model in the test cases is the average distance of the recommended titles generated by the model. The average distance for each model can be found in Table I.

TABLE I
AVERAGE DISTANCE OF RECOMMENDED TITLES

Test Case #	Average Distance
1	3.916472
2	0.018728
3	1.380527
4	0.009037

It can be seen in Table I that the model that gave the most accurate recommendations is the model from test case #4 which is the TF-IDF based model using cosine similarity since it has the lowest average distance which is 0.009037. While the model that gave the least accurate recommendations is the model from test case #1 which is the BoW based model using Euclidean distance with an average distance of 3.916472.

Furthermore, it can also be observed that the models that used cosine similarity as its similarity measure gave better results compared to models that used Euclidean distance. Thus, it can be concluded that cosine similarity is the better similarity measure to be used for this recommender system and the TF-IDF based model that used cosine similarity is the most preferred model.

V. Conclusion

To develop a recommender system, different techniques have been proposed by several other works and researchers, one of which is the use of a content-based recommender system. This technique creates a user profile based on the features of the items to provide recommendations. With the use of the MIND-small dataset, the study was able to create a content-based recommender system while also comparing the effects of using different similarity measures. The recommender system was developed using the BoW and TF-IDF methods. Two similarity measures, Euclidean distance and cosine similarity, were used in both methods to compare which combination of similarity measure and method gave the best recommendations.

The study concludes that using cosine similarity with TF-IDF based model is a suitable combination for a content-based recommender system for recommending news articles to users, given that it achieved a shorter distance score between recommended articles and has a smaller average distance score as well.

However, the researchers recommend that other methods may be employed for a content-based recommender system to explore other ways of recommending content with not only better overall performance but also those that may be more suited to specific kinds of content. This is also given that the methods used in this study are language-independent and may not consider certain subtle and intrinsic factors within the language of the data used.

References

[1] A. Pujahari and D. S. Sisodia, "Item feature refinement using matrix factorization and boosted learning based user profile generation for content-based recommender systems," *Expert Systems with Applications*, vol. 206, p. 117849, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422011046

[2] A. Roy, "Introduction to recommender systems- 1: Content-based filtering and collaborative filtering," Jul 2020. [Online]. Available: https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421

[3] F. Wu, Y. Qiao, J.-H. Chen, C. Wu, T. Qi, J. Lian, D. Liu, X. Xie, J. Gao, W. Wu, and M. Zhou, "MIND: A large-scale dataset for news recommendation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 3597–3606. [Online]. Available: https://aclanthology.org/2020.acl-main.331

[4] Y. Xin, "Challenges in recommender systems : Scalability, privacy, and structured recommendations," Jun 2015. [Online]. Available: https://dspace.mit.edu/handle/1721.1/99785

[5] "What content-based filtering is amp; why you should use it," Apr 2021. [Online]. Available: https://www.upwork.com/resources/what-is-content-based-filtering

[6] N. Mishra, S. Chaturvedi, A. Vij, and S. Tripathi, "Research problems in recommender systems," *Journal of Physics: Conference Series*, vol. 1717, no. 1, p. 012002, jan 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1717/1/012002

[7] A. Rudrawar, "Content based remote-sensing image retrieval with bag of visual words representation," in *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*, 2018, pp. 162–167.

[8] A. N. S., H. Kumaar, S. N. D., S. S., and V. S., "Content-based movie recommender system using keywords and plot overview," *IEEE Xplore*, p. 49–53, Mar 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9767172

[9] M. Hong and J. J. Jung, "Towards social big data-based affective group recommendation," *Mobile Networks and Applications*, vol. 23, no. 4, pp. 1111–1122, 08 2018, copyright - Mobile Networks and Applications is a copyright of Springer, (2018). All Rights Reserved; Last updated - 2022-11-22. [Online]. Available: https://www-proquest-com.dlsu.idm.oclc.org/scholarly-journals/towards-social-big-data-based-affective-group/docview/2065177982/se-2

[10] S. Prabhakaran, "Lemmatization approaches with examples in python," Oct 2018. [Online]. Available: https://www.machinelearningplus.com/nlp/lemmatization-examples-python/

[11] K. Ganesan, "What are stop words?" Apr 2019. [Online]. Available: https://www.opinosis-analytics.com/knowledge-base/stop-words-explained/

[12] "An introduction to bag of words (bow) — What is bag of words?" Oct 2022. [Online]. Available: https://www.mygreatlearning.com/blog/bag-of-words/

[13] V. Zhou, "A simple explanation of the bag-of-words model," Dec 2019. [Online]. Available: https://towardsdatascience.com/a-simple-explanation-of-the-bag-of-words-model-b88fc4f4971

[14] D. Jurafsky and J. H. Martin, *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 3rd Edition.* Stanford University, 2022. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf

[15] "CountVectorizer." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[16] S. Qaiser and R. Ali, "Text mining: use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018.

[17] "TfidfVectorizer." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[18] "euclidean_distances." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html

[19] "cosine_similarity." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html