

FINAL REPORT

Articles recommendation system using Graph Theory - A tool for busy guys (like us)

Date: 09th December 2021

Team: Toiyeudiscretemat

Nguyễn Dương Tùng - 20tung.nd2@vinuni.edu.vn
Lê Hoàng Vinh - 20vinh.lh@vinuni.edu.vn
Nguyễn Tiết Nguyên Khôi - 20khoi.ntn@vinuni.edu.vn

Repository:

<https://github.com/tungnd237/DiscreteMath-Project-TextMining>

Keywords:

Graph theory, articles recommendation, text clustering, data crawling, Vietnamese newspaper

Abstract:

Discrete Mathematics is an important major for computer science and it is cover a large number of topics, one of which is graph theory. Our mini-project focused on applying graph theory to deploy a model that recommends newly worth-reading news and articles every day. This report goes through our introduction, mathematical background, implementation, and results of this project.

1. Introduction

The key concept of our project is using graph theory to detect articles (from different sources) that have similar content and give recommendations on articles that are worth reading.

1.1. Problem statement

For us, one of our regular habits in the morning is looking for the daily news by skimming through online newspapers and websites such as Vnexpress, Laodong, Dantri, etc. Each source usually focuses on different aspects, and thus, by observing information from multiple sources, we are able to understand and adapt to daily life. However, for many times, we realized that **many websites were actually reporting the same story**, maybe with different titles. It was totally a waste of energy and time to read these similar contents over and over again. And not every piece of information was worth our time to read.

1.2. Our solution

We came up with building a tool for news checking. Now, every morning, instead of going through multiple sources and manually selecting news that is worth reading, we just need to go to <https://articlerec-vinuni.herokuapp.com/>, then click on a button, and just in 30 seconds, the tool will **scan through your favorite sources and select significant articles** for us.

1.3. Approach methodology

At first, we referred to machine learning. In fact, there exist many unsupervised (and also supervised) machine learning models that are well-fitted for handling this text-analysis problem, such as K-Nearest Neighbors (KNNs). However, since machine learning is for the pros, we newbie guys prefer trying something that we had the chance to learn in school, which is the concept of graph theory. Besides, we are also curious about its effect compared to machine learning, so we think that it would be nice to try applying the graph theory.

We do not try to build a tool that will analyze the full content in every article. It would be a waste of time and thus we cannot give back the result for users in 30 seconds. In fact, we have an assumption that **articles that contain similar content usually have close titles**, and different sources usually just paraphrase the titles so that they would be different from other sources. Therefore, we approach this problem by analyzing just only the titles of articles, and then building a graph based on the titles, and finally picking up some titles (nodes) that contain heavy-weighted edges.

1.4. Result

The result is quite good and is applicable to use in daily life, even though we shall need some advanced upgrades to run it smoothly. However, with the scope of this project, we think that this should be enough. Also, with a small dataset (about 150-300 titles each time), our tool takes less time to give back the result when compared to machine learning models such as KNN. Since new articles (even from multiple sources) per day will hardly exceed this range, we think that it is totally acceptable.

2. Mathematical background

The titles of articles are being crawled and put into a dataframe. In the concatenated dataframe (Pandas), we set the vertex name to be the index of the news titles, since the true title is very long to display as a vertex name. From here, the **edge will be created if two vertices have the same term in the title, and the weight of that edge is the number of frequent terms between the two** (exclude stopwords such as: “biết bao”, “bởi vì”, etc.).

We are interested in the vertices of the graph ADT (abstract data type). The method is based on the fact that if one of the stories is significant enough, it must have been reported by almost every media outlet. There must be some terms in common across all the titles pertaining to this narrative. Each vertex with a story that is covered in many news is most likely to have a high degree of 2 or 3. Therefore, the vertex that is connected with most of the other vertices in its subset graph has a high probability of containing worth-reading articles.

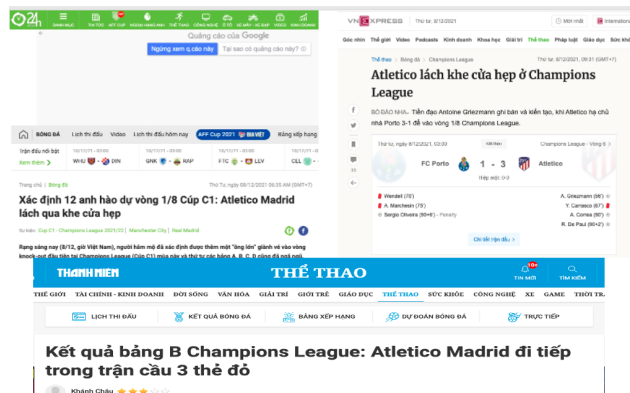


Fig 1. Similar headlines that contain the same news

Now we would want to **choose the vertices with high degrees**. The vital vertex should connect to all the other vertices inside a subset, this subset is having a unique story that is covered in news. Each subset indicates one unique story. All the vertices adjacent to the vital vertex should be redundant while they are all talking about the same story.

We expect the model to achieve the objective in linear time. The general flow (algorithm) to construct a graph is run as follow:

1. Create an ordered dictionary with key is vertex and degree is value.
2. In each iteration, remove the vertex V with the highest degree from the current ordered dictionary.
3. Add vertex V into the result set.
4. Delete the adjacent vertex of vertex V from an ordered dictionary.
5. Repeat step until the ordered dictionary is empty.

The result set will contain the vertex with the story that is vital and important to us.

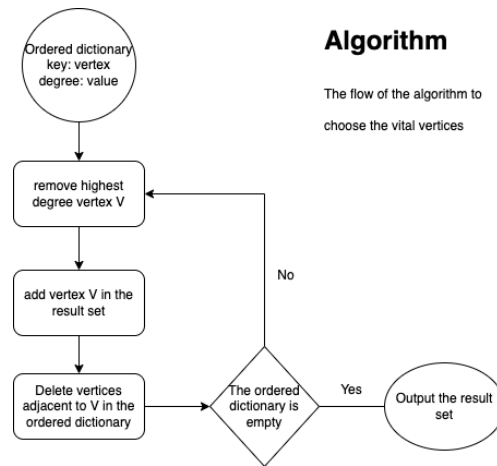


Fig 2. Algorithm to find the vital vertex

3. Implementation

3.1. Prepare data by crawling from websites

Since we focused on Vietnamese news, we built crawlers using Scrapy to crawl the content of press websites. It is impossible to crawl all websites, since each has a different HTML & CSS structure, and a crawler must be able to recognize and handle each structure. Therefore, we only built 3 crawlers to **crawl 3 websites which are vnexpress.net, laodong.vn, and dantri.com.vn**. For every website, we picked **4 genres** to crawl, which also our favorite genres, which were “news” (thời sự), “sports” (thể thao), “worlds” (thế giới, chính trị), “business & economics” (kinh doanh). Also, we are only interested in newly-published articles within a day, therefore, instead of crawling all pages of a website, we only crawled **15 newest articles per genre**. This means, for 3 websites with 4 genres each, we got about **180 new articles** (per run).

The information being crawled in an article are the **title, the article’s URL, and the summary part**. After being crawled, such information was put into a CSV file to further construct the dataframe. Since we only needed the titles and the URLs for building the graph and making recommendations, the dataframe only contained those 2 columns. The summary part was used for us to manually check the accuracy of the graph and recommendations.

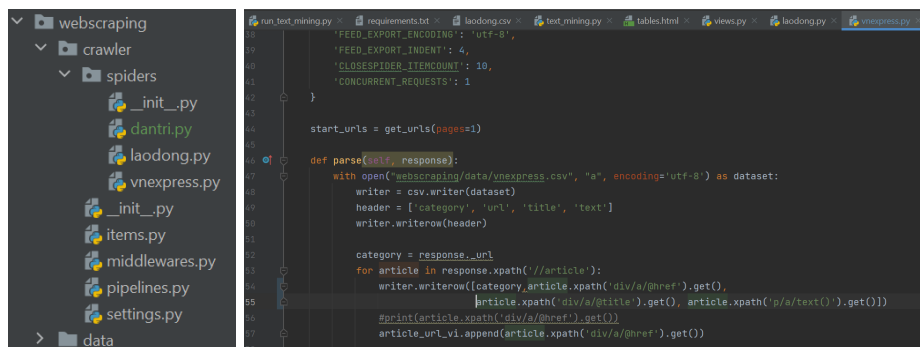


Fig 4 & 5. Structure of the crawlers (left), which includes 3 spiders to crawl 3 websites and other files to set up the crawlers. The parse function (right) is the most important part of a crawler since it gets all necessary data and pastes them into the CSV file. This function was being repeatedly call-back when we run the program, and each call-back execute one article each time.

```

category,url,title,text
https://vnexpress.net/the-thao,https://vnexpress.net/cuc-dien-champions-league-truoc-luot-tran-cuoi-4399891.html,Cực diện Champions League trước lượt trận cuối-4399891.html,Cực diện Champions League trước lượt trận cuối-4399891.html
https://vnexpress.net/the-thao,https://vnexpress.net/chien-luoc-nhap-tich-that-bai-cua-indonesia-4399745.html,Chiến lược nhập tịch thất bại của Indonesia-4399745.html,Chiến lược nhập tịch thất bại của Indonesia-4399745.html
https://vnexpress.net/the-thao,https://vnexpress.net/lewandowski-mong-messi-khong-noi-suong-4399657.html,Lewandowski: 'Mong Messi không nói suông'-4399657.html,Lewandowski: 'Mong Messi không nói suông'-4399657.html
https://vnexpress.net/the-thao,https://vnexpress.net/nham-lan-ky-la-o-hero-world-challenge-4399726.html,'Nhâm lần kỷ lạ' ở Hero World Challenge-4399726.html,'Nhâm lần kỷ lạ' ở Hero World Challenge-4399726.html
https://vnexpress.net/the-thao,https://vnexpress.net/klopp-toi-se-bi-an-don-neu-khong-xoay-tua-4399598.html,Klopp: 'Tôi sẽ bị an đôn nếu không xoay tua'-4399598.html,Klopp: 'Tôi sẽ bị an đôn nếu không xoay tua'-4399598.html
https://vnexpress.net/the-thao,https://vnexpress.net/ba-anh-em-singapore-tham-vong-doat-aff-cup-4399509.html,Bà anh em Singapore tham vòng đoạt AFF Cup-4399509.html,Bà anh em Singapore tham vòng đoạt AFF Cup-4399509.html
https://vnexpress.net/the-thao,,
https://vnexpress.net/the-thao,https://vnexpress.net/ronaldo-dan-dau-binh-chon-globe-soccer-awards-2021-4399531.html,Ronaldo dẫn đầu bình chọn Globe Soccer Awards 2021-4399531.html,Ronaldo dẫn đầu bình chọn Globe Soccer Awards 2021-4399531.html

```

Fig 6. Data after being crawled (from vnexpress crawler).

There is also another dataset that is crucial for this project - the Vietnamese stopwords dataset. In a sentence or text, there usually contains words that do not affect the meaning of that sentence/text, for example (with Vietnamese) are “thì”, “là”, “cái”, “ai”, etc. We don't want our system to analyze those words since they are redundant words and can be excluded.

In the NLTK module (for natural language processing) there is the English stopwords dataset, which contains all redundant words. We found the same dataset for Vietnamese stopwords on Github ([link](#)) and based on this dataset to exclude Vietnamese redundant words from our articles' titles.

3.2. Pre-processing

In natural language processing, lemmatization refers to the technique to transform a word, usually a verb, into its original form. For example, in English, “walked” and “walking” are different forms of “walk”. Since different forms of words do not (or hardly) refer to different meanings, people usually transform them into their original words to reduce the vocabulary load. Due to the characteristics of the Vietnamese language, we excluded the lemmatization process, however, this technique would be necessary if we analyze foreign English articles.

Another technique that usually goes with lemmatization is steaming, which is the process of word spelling correction. The NLTK module support steaming for the English language but not Vietnamese. Therefore, we excluded this process. However, since lemmatization and steaming are the basic and must-have pre-processing steps to process natural language, we still kept them here.

In general, since we did not use lemmatization and steaming, the purpose of our pre-processing function *preprocessing* (below) was only to **split the article's title into a list of words for graph building, and exclude stopwords**. In fact, we can also keep the code for lemmatization and steaming, and the result would still be the same.

After running the function *preprocessing*, each title from the dataset had its own list of words used to construct the graph as mentioned in part 3.3.

```

### function for preprocessing step.
# Since we use Vietnamese articles, the lemmatization and stemming step can be skipped.
def preprocessing(text, stopword, lower=True, is_lemma=False, is_stem=False):

    text_clean=text if lower==False else text.lower()

    #tokenize, remove stop words
    tokens=[]

    for i in nltk.tokenize.RegexpTokenizer(r'\w+').tokenize(text_clean):
        if i not in stopword:
            token.append(i)

    #lemmatization
    if is_lemma:
        text_processed=[]
        for i in token:
            text_processed.append(nltk.stem.wordnet.WordNetLemmatizer().lemmatize(i))
    else:
        text_processed=token

    #stemming
    if is_stem:
        output=[]
        for i in text_processed:
            output.append(nltk.stem.PorterStemmer().stem(i))
        else:
            output=text_processed

    #remove numbers since they are also stopword
    remove_list=[]
    for i in output:
        remove_list.append(i)

    for i in remove_list:
        try:
            float(i)
            output.remove(i)
        except:
            pass

    return_list=[]
    for i in output:
        if i not in stopword:
            return_list.append(i)

    return return_list

```

Fig 7. Pre-processing with lemmatization and steaming. We tested both keeping and excluding 2 techniques, and in fact, it still gave you the same result.

3.3. Graph building

The edge will be created if two vertices have the same term in the title, and the **weight of that edge is the number of frequent terms between the two**. We defined the `find_common_words` function with 3 inputs - title 1 & title 2 are 2 different titles for comparison, and the stopwords list to exclude stopwords from the common word list.

The `add_word_column` function simply adds a title's list of words into the dataframe. This means our dataframe after going through this function, would return a dataframe with 2 old columns and an additional one which are titles, URLs, and lists of words.

```

#find common words between two titles
#return the number of common words as the weight of the edge
def find_common_words(title_1,title_2,stopword):

    common=set(title_1).intersection(set(title_2)).difference(set(stopword))

    return len(common)

#add word list as an additional column.
def add_word_column(df,stopword,**kwargs):

    word_column=[]

    for i in df['title']:
        word_column.append(preprocessing(i,stopword,lower=True,**kwargs))
    df['word']=word_column

    return df

```

Fig 8. Find_common_words and add_word_column functions.

Next, we defined a function to construct an undirected weighted graph using the `networkx` module and used the dataframe indexes as the node name. **We only connect two nodes if they share common words (exclude stopword)** and set the number of common words as the weight of the edge.

```

labeldict = {}

def build_graph(df, stopwords):
    graph = nx.Graph()

    for i in range(len(df)):
        for j in range(i + 1, len(df)):
            w = find_common(df['word'][i], df['word'][j], stopwords)
            if w != 0:
                labeldict[i] = df['title'][i]
                labeldict[j] = df['title'][j]
                graph.add_edge(i, j, weight=w)

    # print title and position
    return graph

```

Fig 9. The build_graph function. The “labeldict” is an additional dictionary for plotting the graph.

Some titles may not share any common words with others. In other words, they are not included in the graph structure. We added them back to the output list in order not to leave the minority behind. Even though they may not be key information published by every website, they could still be some exclusive or niche information.

```

def add_non_connected(df, output, graph):
    for i in range(len(df)):
        if i not in list(graph.nodes):
            output.append(i)

    return output

```

Fig 10. Function to add non-connected vertices.

Finally, we defined **graph traversal to get the most-weighted vertices**. We first constructed the dictionary that contained all vertices and weights in the graph and ordered the dictionary by the weights of the vertices. Then, we found the most-weighted vertices by using iteration and queue structure to remove each vertex’s neighbors until the queue is empty.

```

def graph_traversal(graph):
    #dictionary of all vertices and weights in graph
    dict_vertex = dict(graph.degree)

    #order dict by each node's weight
    dict_vertex = dict(sorted(dict_vertex.items(), key=lambda x: x[1], reverse=False))

    queue = list(dict_vertex.keys())
    recommended_vertices = []

    #define a queue find the node with the highest degree.
    #the iteration will run until the queue is empty.
    while queue:
        V = queue.pop()
        recommended_vertices.append(V)

        redundant = set(queue).intersection(set(graph.neighbors(V)))

        for i in redundant:
            queue.remove(i)

    return recommended_vertices

def recommendation(df, stopwords, plot_original=False,
                  plot_result=False, **kwargs):
    # tokenization
    df = add_word_column(df, stopwords, **kwargs)

    # graph building
    graph = build_graph(df, stopwords, **kwargs)

    # fix node position for visual comparison
    pos = nx.spring_layout(graph, k=0.3)

    # plot original
    if plot_original:
        graph_visualization(graph, position=pos,
                            title='Original', **kwargs)

    # traversal to get the recommended vertices/ articles
    recommended_articles = graph_traversal(graph)

```

Fig 11. Graph traversal (left). Fig 12. Part of the compact function for preprocessing, graph building, and recommendation (right)

Figure 12 is the combined function **recommendation** which will run all functions in parts 3.2 (pre-processing) and 3.3. This function also plots the graph after being built.

3.4. Combine all pieces together

We created a file `run_text_mining.py` to run all of the commands from crawling data to getting recommendations and graph plotting.

```
webscraping.crawl_data()

df_vnexpress = pd.read_csv(r'webscraping/data/vnexpress.csv', encoding_='utf8')
df_laodong = pd.read_csv(r'webscraping/data/laodong.csv', encoding_='utf8')

list_df = []

for i in df_vnexpress['category'].unique():
    if i != 'category':
        list_df.append(df_vnexpress[['title', 'url']][df_vnexpress.category == i].iloc[:15].copy())

list_df.append(df_laodong[df_laodong.title != 'title'][['title', 'url']].sample(50))

df_title = pd.concat(list_df, axis=_0)
df_title = df_title.dropna()
df_title = df_title.reset_index(drop=True)
df_title = df_title[df_title.title != "title"]

### stopwords vietnam
f = open(stopwords, 'r', encoding='utf8')
stopword_vn = f.read().split('\n')

### Run graph
output_position=tm.recommendation(df_title, stopword_vn, plot_original=True, plot_result=True)
```

Fig 13. The `run_text_mining.py`

The general flow would be:

1. Calls the `crawl_data` function to crawl the first pages from every genre from 3 websites (vnexpress, dantri, laodong) and pastes the data into CSV files.
2. Read CSV files and combine multiple CSV files into one dataframe.
3. Drop null data, which is data that were wrongly crawled.
4. Process the Vietnamese stopwords dataset (file .txt).
5. Paste the crawled data and the stopwords list through the compact function `remove_similar` to construct and plot the graph, and select the most weighted vertices.

Finally, we create Django virtual environment and build a simple user interface (from template) to use it and all we need to do is to run this `run_text_mining.py` in Django. We also used a free hosting service to host our website online. Check <https://articlerec-vinuni.herokuapp.com/> and click TODAY'S ARTICLE RECOMMENDATION, then wait for 10-30 seconds to get the result.

4. Result and Analysis

4.1. Graph visualization

Here is the visualization for the constructed graph. You can get the graph on our hosted website (follow part 3.4 instruction). This graph is being constructed in real-time, which means it is based on newly-crawled data every day to create a whole new graph.

The purple nodes are our recommended articles to read, which means the most-weighted vertices selected from the graph.

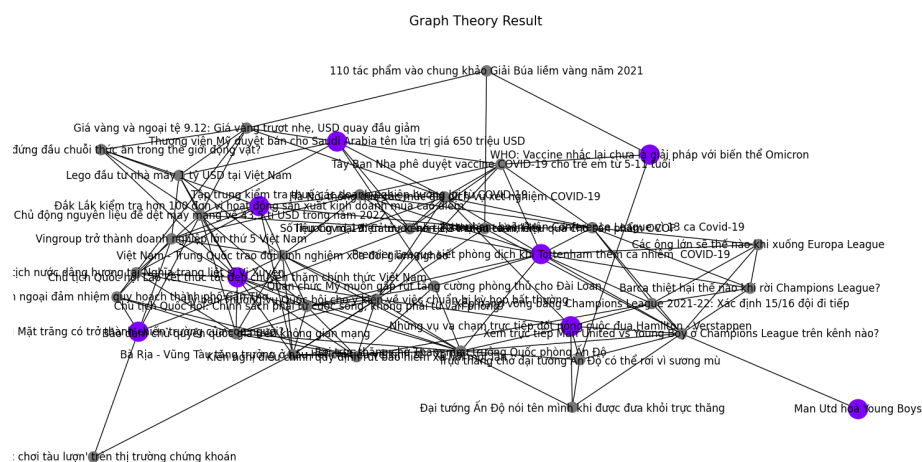


Fig 14. Graph with titles of selected nodes (50 titles).

In our hosting website, we replaced the word titles with their index numbers for better visualization. We also defined a step to print the list of the full titles of recommended articles below the graph image so that users will be easier to follow.

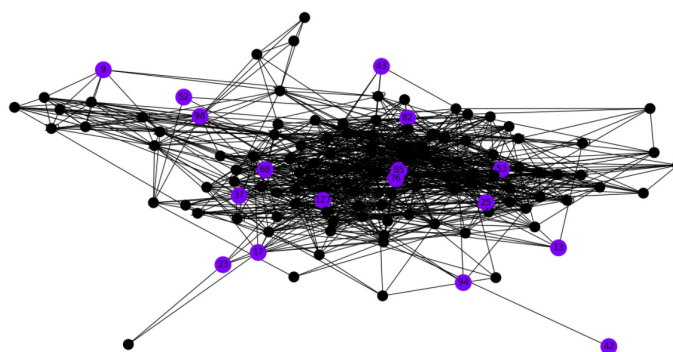


Fig 15. Graph visualization - website version

0	13	Juventus soạn ngôi đầu của Chelsea	https://vnexpress.net/juventus-soan-ngoi-dau-cua-chelsea-4400542.html
1	17	Evergrande bị hạ tin nhiệm xuống 'vỡ nợ giới hạn'	https://vnexpress.net/evergrande-bi-ha-tin-nhiem-xuong-vo-no-gioi-han-4400877.html
2	23	Xu hướng 'diễn cuồng mua sắm' cuối năm giúp doanh nghiệp bất tốc	https://vnexpress.net/xu-huong-dien-cuong-mua-sam-cuoi-nam-giup-doanh-nghiep-but-toc-4397583.html
3	25	Thanh toán số không cần tài khoản ngân hàng	https://vnexpress.net/thanh-toan-so-khong-can-tai-khoan-ngan-hang-4400597.html
4	37	Bộ trưởng Đức đập xe đi nhậm chức	https://vnexpress.net/bo-truong-duc-dap-xe-di-nham-chuc-4400705.html
5	42	Hơn 4,3 triệu người già cần hỗ trợ chăm sóc	https://vnexpress.net/hon-4-3-trieu-nguoi-gia-can-ho-tro-cham-soc-4400831.html
6	43	Đưa dữ liệu điều tra dân số lên mạng	https://vnexpress.net/dua-du-lieu-dieu-tra-dan-so-len-mang-4400596.html
7	52	Robot mặt người Sophia bước chân vào Metaverse với phiên bản NFT riêng	https://laodong.vn/the-gioi/robot-mat-nguoi-sophia-buoc-chan-vao-metaverse-voi-phien-ban-nft-rieng-982714.Ido
8	53	Tuyển Việt Nam đấu Malaysia: Tái hiện chung kết AFF Cup 2018	https://laodong.vn/bong-da/tuyen-viet-nam-dau-malaysia-tai-hien-chung-ket-aff-cup-2018-982697.Ido
9	68	Vasep kiến nghị về những quy định bất cập trong kiểm dịch thủy sản	https://laodong.vn/kinh-te/vasep-kien-nghy-ve-nhung-quy-dinh-bat-cap-trong-kiem-dich-thuy-san-982554.Ido
10	76	Thủ tướng khen 180 tập thể, cá nhân xuất sắc trong phòng, chống COVID-19	https://laodong.vn/thoi-su/thu-tuong-khen-180-tap-the-ca-nhan-xuat-sac-trong-phong-chong-covid-19-982330.Ido

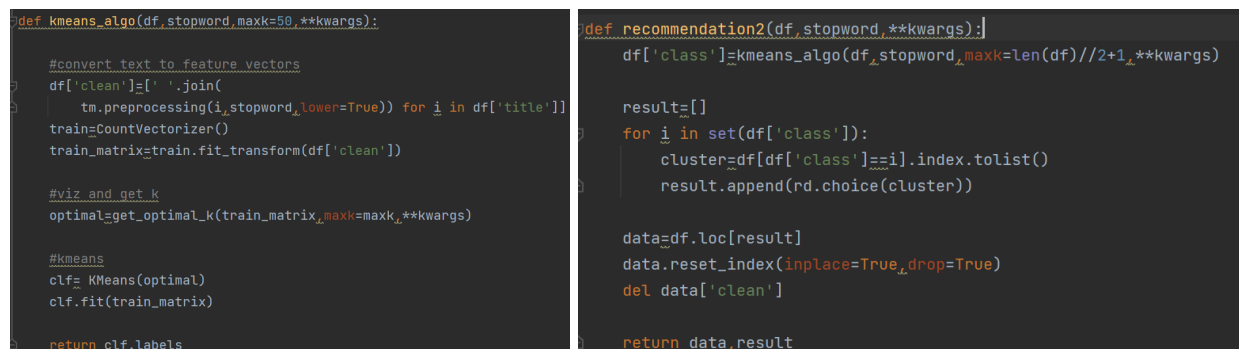
Fig 16. Chart of recommended articles provided together with the graph visualization

4.2. Result

The system works pretty well and it is **able to recommend to users significant articles**. However, sometimes it still generates awkward recommendations, especially when using datasets with less than 100 titles. We believe this issue is well-solved by increasing the number of titles since it will increase the number of words as well, and thus the weights of different vertices will be more significant. At first, we only took 50-100 titles for visualization purpose and testing so that we could better evaluate the graph. Our final product (which was also implemented in our hosting website) constantly took 15 articles per genre, 4 genres per website, and 3 websites in total, so it would be about 180 titles per run time.

Also, we believe that this problem can be fixed by using the “summary” column of the dataset since each summary of each article contains longer content compared to the title only. We believe that the more word (exclude stopwords) you feed in, the more accurate the graph will be.

We also compared the accuracy of our graph model with the machine learning model KNN, one of the most used models when it comes to text clustering. The accuracy of the graph is hardly measured if we consider the “correctness” of the recommendations because different people will have different points of view on what content is suitable for them. Therefore, we compared the effects of such 2 models by comparing their processing time.



```
def kmeans_algo(df, stopwords, maxk=50, **kwargs):
    #convert text to feature vectors
    df['clean'] = [' '.join(
        tm.preprocessing(i, stopwords, lower=True)) for i in df['title']]
    train = CountVectorizer()
    train_matrix = train.fit_transform(df['clean'])

    #viz and get k
    optimal = get_optimal_k(train_matrix, maxk=maxk, **kwargs)

    #kmeans
    clf = KMeans(optimal)
    clf.fit(train_matrix)

    return clf.labels_

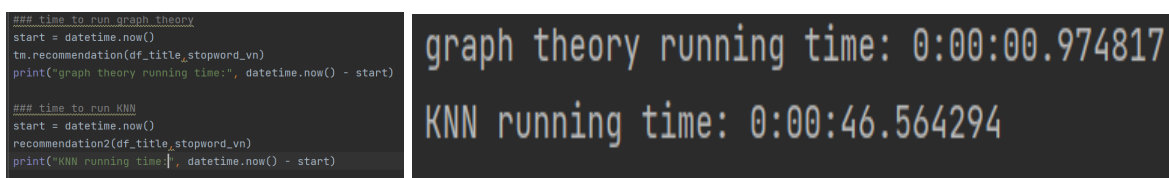
def recommendation2(df, stopwords, **kwargs):
    df['class'] = kmeans_algo(df, stopwords, maxk=len(df)//2+1, **kwargs)
    result = []
    for i in set(df['class']):
        cluster = df[df['class'] == i].index.tolist()
        result.append(rd.choice(cluster))

    data = df.loc[result]
    data.reset_index(inplace=True, drop=True)
    del data['clean']

    return data, result
```

Fig 17. Functions for KNN (using Jupyter Notebook)

For a dataset of 180 titles, the result showed that **the graph model (0.97 sec) did much better than the KNN model (46 secs)**. However, the story was different when we tried with a larger dataset, which is about 7000 titles per run. The KNN needed approximately 5-6 minutes to return the result, meanwhile, our graph model was still processing after 15 minutes. However, as we discussed in the introduction part, since the number of newly-published articles (for 3-4 websites) is about 150-300 articles per day, we think our graph model would be okay.



```
### time to run graph theory
start = datetime.now()
tm.recommendation(df_title_stopword_vn)
print("graph theory running time:", datetime.now() - start)

### time to run KNN
start = datetime.now()
recommendation2(df_title_stopword_vn)
print("KNN running time:", datetime.now() - start)
```

graph theory running time: 0:00:00.974817
KNN running time: 0:00:46.564294

Fig 18. Processing time between graph model and KNN

4.3. Further work & improvement

Even though this work was only for the mini-project of Discrete Math class, we think this one is very potential to develop in the future if we have the chance to do so.

One thing to improve is to extend the number of websites that the system is able to crawl. We thought about creating a filter on our website so that users can choose multiple websites that they normally read news from. However, we were not able to do so since it was near the deadline.

Also, since the NLTK module was built to (mostly) handle English, we found it was difficult to apply in the Vietnamese case. We thought about researching some Vietnamese-built open-source modules (of VinAI or somewhere) to better handle the Vietnamese natural language processing.

The graph visualization is also a part that needs improvement. We tried to code a 3D interactive graph so that it would be better to look, however, we could not complete it on time so we just put it away.

5. Work reflection

P.I.C	Work division	Time committed	Problems encounter and solutions
Nguyễn Tiết Nguyễn Khôi	<ul style="list-style-type: none"> - Build web-scraping - Pre-processing part - Build graph 	35 hours	<ul style="list-style-type: none"> - Modified spiders to crawl multiple sources => Must change the crawling structure based on the websites' structure. - Found Vietnamese stopwords list and try to make the graph suitable for analyzing Vietnamese news. - Spent time on the graph traversal part.
Nguyễn Dương Tùng	<ul style="list-style-type: none"> - Build Django and host website - Implement raw code into the website - Build graph 	35 hours	<ul style="list-style-type: none"> - Try AWS, PythonAnywhere hosting services and failed because of the large package required => Deploying website using Heroku and dividing loading process into multiple parts, and checking if the requirements file is sufficient. - Failed to implement in a function in views.py of Django => importing os system. - Rendering CSV file and graph

			into HTML components => using encoding utf-16 and using plt.savefig.
Lê Hoàng Vinh	<ul style="list-style-type: none"> - Graph visualization - Research mathematical background - Build graph 	35 hours	<ul style="list-style-type: none"> - Create and modify the graph using networkx and panda library, comprehend the necessary functions to apply => carefully refer and go through the documentation - Visualize and arrange the graph in a satisfying and readable manner => Try to run the program many times and find the pattern to improve and create better visualizations.

6. References

Y. Zhang, M. Chen, and L. Liu, "A review on text mining," in 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2015, pp. 681–685.

A. Hagberg, P. Swart, and D. Chult, "Exploring network structure, dynamics, and function using networkx," Jan. 2008.

E. Loper, Nltk: Building a pedagogical toolkit in python, 2004.

Trung Nguyen, T. (2020, September 16). GitHub - trungngv/web_scraping: Web scraping. GitHub. https://github.com/trungngv/web_scraping

Duc Duy, N. (2020, June 19). scrapy-vietnamese-news-spiders/vina_news/vina_news/spiders at master · nguyenducduy/scrapy-vietnamese-news-spiders. GitHub. https://github.com/nguyenducduy/scrapy-vietnamese-news-spiders/tree/master/vina_news/vina_news/spiders

Graph Clustering Topic on GitHub. (n.d.). GitHub. <https://github.com/topics/graph-clustering>

Suis, J. (2021, August 14). Graph Theory Repository, Text Mining Project. GitHub. <https://github.com/je-suis-tm/graph-theory>

Sumit, P. (2009, November 24). What is the difference between lemmatization vs stemming? Stack Overflow. <https://stackoverflow.com/questions/1787110/what-is-the-difference-between-lemmatization-vs-stemming>

Real Python. (2021, April 10). The k-Nearest Neighbors (kNN) Algorithm in Python. <https://realpython.com/knn-python/>