CS 4624
Multimedia, Hypertext, and Information Access

Final Report
17 May 2023

ETD Recommendation System
Virginia Tech, Blacksburg, VA 24061

Instructor: Prof. Edward A. Fox, Client: Satvik Chekuri
Talia Blakemore, Long Phan
Abstract

Our project involves expanding upon a previous recommendation system built by CS 5604 students. Previous CS 5604 teams have created a chapter summarization model to generate summaries for over 5000 Electronic Theses and Dissertations (ETDs). We used these summaries to fuel our recommendation system. Using chapter summaries improved our ability to predict resources that a user may be interested in because we narrowed our focus to individual chapters rather than the abstract of the whole paper. Authors will benefit from this recommendation system because their work will be more accessible. We provide a web page for users to explore how different clustering algorithms impact the search results, giving the user the ability to modify parameters such as the number of clusters and minimum cluster size. This web page will appeal to niche users interested in experimenting with recommendation systems, allowing them to fine-tune the recommendation results. We recommend for future work to continue exploring different clustering algorithms, as well as using our chapter recommendations to fuel a recommendation list based on each chapter. During this project, we learned about clustering algorithms, working as a team, and starting a project from the ground up. A previous CS5604 team built a stand-alone website that supports search, a recommendation system, and the ability to experiment with different search methods. During this semester, we expanded upon the existing website, using clustering algorithms to experiment with the recommendation system. Users may specify different parameters to understand how different clustering algorithms may change the recommendations.
Table of Contents

List of Tables........................................................................................................................................6

List of Figures.........................................................................................................................................7

1 Introduction.........................................................................................................................................9
   1.1 Client...........................................................................................................................................9
   1.2 Problem.......................................................................................................................................9
   1.3 General Approach.......................................................................................................................9

2 Literature Review..............................................................................................................................11
   2.1 Faiss...........................................................................................................................................11
   2.2 ANNoy.......................................................................................................................................12
   2.3 Extracted Summary Based Recommendation System for Indian Legal Documents........12
   2.4 WebInEssence: A Personalized Web-Based Multi-Document Summarization and
       Recommendation System.............................................................................................................12
   2.5 Review of Existing System.........................................................................................................13

3 Requirements.....................................................................................................................................14
   3.1 Efficiency....................................................................................................................................14
   3.2 Accuracy.....................................................................................................................................14
   3.3 List of Requirements...................................................................................................................14

4 Design...............................................................................................................................................15
   4.1 Backend Design..........................................................................................................................15
   4.2 Frontend Design..........................................................................................................................18

5 Implementation................................................................................................................................19
5.1 Overview

5.2 Connecting Recommendation Systems to Website

6 Testing & Evaluation

6.1 KMeans Evaluation

6.2 DBScan Evaluation

6.3 HDBScan Evaluation

7 User’s Manual

7.1 Recommendation Experiments Page

8 Developer’s Manual

8.1 Methodology

8.2 Retrieving Chapter Summaries

8.3 Integrating Recommendation Systems

8.4 Clustering Algorithm

8.5 Sorting the Clustering Output

8.6 Preparing Output for Integration with Front-End

8.7 Recommendation Experiment Front-End

8.8 Gitlab Repositories

9 Lessons Learned

9.1 Timeline

9.2 Problems and Solutions
List of Tables

1  Silhouette Coefficient Table.................................................................24
2  General Timeline of Tasks.................................................................40
3  Methodology......................................................................................48
List of Figures

1. Pipeline of the proposed model.................................................................10
2. Common words in a cluster derived from KMeans......................................16
3. Common words in a cluster derived from UMAP + KMeans.......................16
4. KMeans Clustering...................................................................................17
5. DBScan Clustering ..................................................................................17
6. HDBScan Clustering ................................................................................17
7. Elbow Plot for KMeans............................................................................20
8. Number of Clusters versus Silhouette Coefficient.......................................21
9. KMeans Using Seven Clusters.................................................................21
10. KMeans Using Nine Clusters.................................................................22
11. KMeans Using Thirteen Clusters.............................................................22
12. DBScan with Epsilon of 0.4 and Minimum Cluster Size: 13 .......................25
13. DBScan with Epsilon of 0.3 and Minimum Cluster Size: 6 .......................26
14. HDBScan with 500 data points, min_cluster_size of 3, and min_samples of 10........27
15. HDBScan with 1000 data points, min_cluster_size of 3, and min_samples of 10........28
16. Profile Creation Page..................................................................................29
17. Home Page...............................................................................................30
18. Recommendation Experiments Page .......................................................31
19. Recommendation Experiments Page after Search....................................31
20. Recommendation Experiments Page after KMeans Clustering.....................32
21 Recommendation Experiments Page after DBScan Clustering
22 Methodology
23 Chapter Summarization Request Output
24 Chapter Summaries Retrieval Python Script
25 Chapter Summaries to Embeddings Python Script
26 Clustering Function
27 Sorting Function
28 Recommendation Experiment Controller
29 Goal 1 Workflow- Methodology
30 Goal 2 Workflow- Methodology
1 Introduction

The work completed by Electronic Theses and Dissertations (ETDs) publishers is extremely valuable to researchers, students, and universities. However, this information is difficult to find and parse through. Our project will build upon a previous system to make chapters of these documents more accessible.

1.1 Client

Satvik Chekuri, a Ph.D. student advised by Dr. Edward Fox, guides our team for this semester. Mr. Chekuri previously worked with a CS 5604 team to create a website that allows visitors to query the published ETDs, while still being served a list of recommended documents [7]. This website will serve as a more accessible manner of finding ETDs similar to ones the user has already viewed, thereby generating more traffic to these documents.

1.2 Problem

Currently, the system built by the previous team supports a fusion of two scoring algorithms: BM25 [24] and K-Nearest Neighbors (KNN) [25], to sort the documents recommended to the user. BM25 is a common search algorithm that has been a standard for the information retrieval space for a long time. This algorithm uses text as input. Alternatively, K-Nearest Neighbors is an algorithm commonly used with vectors, so embeddings are used to run this algorithm. A search experiments page supports the ability to experiment with different search systems or change the weights to BM25 and KNN. This page allows the user to understand the logic fueling the search systems.

The original search system uses document abstracts provided by the authors to determine the similarity between documents [7]. Instead, our model uses chapter summaries generated by a previous group’s chapter summarization model to evaluate similarity on a narrower scope [8]. Our clustering algorithms allow users to apply a clustering algorithm after a completed search to find similar documents that may not be in the search results. The user has the power to modify parameters such as the number of clusters and minimum cluster size to understand how the clustering algorithm works.

1.3 General Approach

As seen in Figure 1, we utilize chapter summarizations collected by a previous team [8] using a summarization model in our recommendation system. The BERT [9] model created embeddings of the summary text. Then, we applied UMAP [10] to those embeddings to achieve dimensionality reduction. We wanted to explore embeddings with dimensionality reduction to see if it impacts the accuracy of results from running the clustering algorithms. After doing the necessary steps with the embeddings using BERT and UMAP, we performed a clustering algorithm, such as KMeans, on the embeddings. The results showed that chapters from the same ETDs can be found in different clusters due to differing topics. This is further explored in the Implementation section.

To integrate this finding with our system, we combined our recommendation system with the search algorithm. Once a user completes a search, they can apply our clustering algorithm to
the output. The output reveals similar chapters that the searching mechanism might not retrieve based on the keyword.

Figure 1: Pipeline of the proposed model
2 Literature Review

To understand this project, we completed literature reviews of two recommendation systems: Faiss and ANNoY, two recommendation systems in practice, and finally, the previous work completed by other teams for this effort. These resources answered our initial questions about recommendation systems and helped us determine which direction we wanted to take in our project.

2.1 Faiss

Faiss AI or Facebook AI Similarity Search is a library for efficient similarity search and clustering of dense vectors [11]. It supports code for parameter tuning and uses Euclidean distance to determine how similar two objects are. Using k-nearest neighbors (KNN) or a range search of a given radius, elements are returned based on their proximity to the given object.

Faiss can move from a keyword-based search to using the power of embeddings to build efficient search, even if little RAM is present. When combined with ElasticSearch, powerful, intelligent search systems can be made.

Since Faiss uses non-exhaustive searches in large datasets, searching is fast. It also can use a product quantization method for high-dimensional vectors, which allows relatively accurate reconstructions and distance computations in compressed domains. Additionally, to improve speed, an inverted multi-index can be used.

When implementing Faiss, the embeddings of the data need to be extracted and indexed into Faiss because it assumes that dense vectors with a unique integer ID represent the data. Since the data is reindexed to a singular ID encompassing many identifying factors, these vectors allow several indices to be used in the similarity search. Our system can use this to represent the ETD ID, chapter ID, and summary text.

There are several indices: flat index, quantization technique index, and inverted index with product quantization (IndexIVFPQ). The system directly indexes the vector without compression when using a flat index. The quantization technique index allows us to compress the vector, and KMeans divides vectors into clusters. Finally, the user can perform an approximate search based on the concept of nearest-neighbor search due to IndexIVFFPQ.

The most commonly used metric to retrieve results is cosine similarity. An n-probe parameter can be specified to reduce the time to search, but it must be less than the number of clusters specified to create the index.

Faiss can link with Principal Component Analysis (PCA). PCA in Faiss generally uses a flat index or IndexIVF linked with an IndexPreTransform Function. To succeed with PCA, the dimension of a vector needs to be a multiple of 4. However, if need be, the RemapDimensionTransform method can transform a vector of a random size to a size that is a multiple of 4 [5].

Compared to other recommendation systems, Faiss can make incremental updates more easily than ANNoY [2].
2.2 ANNoy

Approximate Nearest Neighbors for Recommender Systems, or ANNoy, is a recommendation system created by Spotify that provides an abstraction over core approximate nearest neighbors implementations [12]. The algorithm indexes training data points in a form so that they are efficiently searchable. Then it makes random projections to create hyperplanes that help determine a region where one or more points exist, so a search is only necessary for this region.

If need be, there are three phases to preprocess the textual data after it has been converted to embeddings: vector transformation for dimensionality reduction and vector rotation, vector encoding for constructing the actual index for search, and a non-exhaustive search component to avoid exhaustive search [20].

Vector transformation is applied to vectors before they are indexed using dimensionality reduction and vector transformation.

Vector encoding is applied to vectors to construct an actual index for search. It encodes vectors in a much more compact form to be more efficiently searchable. There are three ways that encoding can be performed: using trees, LSH, and Quantization [6].

2.3 Extracted Summary Based Recommendation System for Indian Legal Documents

In this paper, researchers look into the problem with Indian legal documents. These legal documents are known for being very verbose and unstructured. This paper gave us more insight into how summaries could improve a recommendation system. The paper goes in-depth about how they used summaries for entire documents. Since legal documents are quite lengthy, using summaries provides better performance in time and space complexity when compared to using the whole document in a recommendation system [18].

Instead of creating/generating a summary, the researchers looked into the documents more to find individual parts equivalent to a summary and extract those parts for the recommendation system [18]. Sampling parts of a paper gave us more information about why the previous system used abstracts in the recommendation system. While abstracts are more cohesive, we decided to explore summaries on individual chapters of ETDs to see if there would be more variance between chapter summaries than entire document summaries.

2.4 WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System

These researchers created a web interface for their recommendation system that uses summarizations and clustering algorithms for their documents [19]. Their work inspired our project for a search + clustering + recommendation + summarization model, as seen in our project. We want to give users a web interface to interact with and search for a chapter from a document and give the user recommendations based on the search result. We also use clustering algorithms to recommend based on the contents in the cluster and their distance from each other. The paper gave us a strong foundation on how a similar existing system works and how we could tweak their methods to our project.
2.5 Review of Existing System

The project built by the CS5604 Information Retrieval team involves building a system that supports searching, ranking, browsing, and recommendations for ETDs [7]. Due to its speed and ability to integrate easily with other aspects of Elastic Stack, this team leveraged the tool ElasticSearch to fuel their search system [14].

The recommendation system proposed pulls from Amazon’s idea of the component of time in conjunction with using the cosine similarity measure to effectively and accurately recommend the most similar item in their inventory [7]. Additionally, this team draws from Netflix’s recommendation system: a bag of items approach [7]. However, instead of having an unordered group of recommendations, Netflix assigns recently watched content a higher weight in the ranking algorithm. To expand on ordering recommendation output, Netflix looked at features such as the time of day or the user's device.

This team generated end-user logs and leveraged that data to provide recommendations for the user. The logged data is tracked by collecting a user’s search queries and the results that they click on. For example, when a user searches for something, their user ID and the search query will be saved. If the user clicks on a result, their user ID and the ETD ID of the result they clicked on will also be saved. The logs allowed this group to use them in their recommendation system and make user-specific recommendations based on logs tied to their user ID. Through various testing, they found this model to have the most accurate recommendations with the most engagement.
3 Requirements

3.1 Efficiency

The team that previously worked on a recommendation system for the ETDs used document abstracts to fuel their recommendations. Our project differs by using model-generated chapter summaries to fuel our recommendations. By narrowing the range of topics through using chapter summaries rather than document abstracts, we can more efficiently recommend individual chapters.

3.2 Accuracy

Our solution to this project enhances the current accuracy of the system by accounting for gaps in the search mechanism. Our system allows flexibility by providing options for clustering algorithms to achieve greater accuracy. By using chapter summaries rather than document summaries, our recommendation system has a greater ability to predict recommendations because the rankings are by a narrower source.

3.3 List of Requirements

The following is a list of requirements for our project to be considered successful:

- Retrieve chapter summaries from the chapter summarization model [8].
- Create embeddings to be used by clustering models.
- Compare clustering models before and after dimensionality reduction (UMAP) is applied [10].
- Provide a front-end for a user to be able to use different clustering models.
- Provide a front-end for a user to be able to specify features of clustering.
  - Number of documents
  - Number of clusters
  - Minimum cluster size
  - Whether or not Principal Component Analysis (PCA) is applied [13]
4 Design

4.1 Backend Design

The design for the backend uses clustering algorithms in conjunction with a distance matrix to sort the clustering output. The pre-existing recommendation system for the website uses this output to aid the recommendations.

The first step in our process was to extract the chapter summaries from a previous team’s project and upload them to our server. These summaries fuel our ranking and are the primary content for experimenting with recommendation systems. Upon starting this project, we believed that two chapters from a single document had the potential to not belong to the same cluster. For example, we can find a result in which chapter 1 from ETD-1 may be in cluster 0, whereas chapter 2 from ETD-1 may be in cluster 1. Thereby, using summaries will narrow the scope of our project to achieve more accurate recommendations not based on the whole document. More accurate recommendations will lead to better engagement.

After cleaning the data and getting chapter summaries from the 5000+ ETD documents, we transformed the text-based inputs into vector-based indices using the BERT model. We created these embeddings to evaluate the text for similarity. The next step in our process was to determine whether or not using dimensionality reduction aided our recommendation system.

We used the Uniform Manifold Approximation and Projection (UMAP) tool to implement dimensionality reduction [10]. UMAP helps keep the clustering algorithm from overlooking minor aspects of the vectors, and boosts performance. We can improve the accuracy and speed of the system by running UMAP prior to clustering.

Principal Component Analysis (PCA) is another form of dimensionality reduction [13]. The purpose of PCA is to reduce the number of variables in a dataset while preserving as much information as possible. The general algorithm finds the first principal component, then finds more components by finding the next uncorrelated component (perpendicular to) with the highest variance. Highly significant data is retained by finding the proper x number of uncorrelated components.

After using the KMeans algorithm with five clusters on the chapter summaries text, Figure 2 shows some of the most common words found in one of the derived clusters. Alternatively, using the same data Figure 3 shows some of the most common words found in clusters derived from the same KMeans algorithm preceded by dimensionality reduction using UMAP. In Figure 3, there are many words easily identified to help describe the cluster. However, in Figure 2, the cluster appears to sort on non-words like al, b, 126, and et. The improvement seen with UMAP convinced us to continue using this tool.
Throughout this project, we worked with three clustering algorithms: KMeans, Density-Based Spatial Clustering of Applications with Noise (DBScan), and HDBScan. The K-Means clustering algorithm from the scikit-learn package clusters all objects and is often used with sparse, high-dimensional data. The DBScan algorithm can identify any noise, aiding it in finding clusters of unusual shapes. It can also efficiently handle clusters of multiple sizes and structures [14]. Finally, HDBScan is built upon the idea of DBScan, but has a fine-tuned ability to assign noise to clusters and can identify clusters with non-uniform density [15].
Figures 4, 5, and 6 show how these three clustering algorithms classify a sample of random data. As seen in Figure 4, KMeans attempts to cluster all the data using the parameter for number of clusters specified during the initial call. However, in Figures 5 and 6, noise is identified around the clusters, leading to clusters of unusual shapes being identified. In the bottom left corner of Figures 5 and 6, those two unusual shapes are identified as individual clusters, while in Figure 4, they are split into three different clusters. The same is observed for the top right corner of the three diagrams. Another important distinction between these three clustering algorithms is the amount of time each one takes. As seen in the top left corner of each picture, KMeans took 0.08 seconds to complete, DBScan took 0.02 seconds to complete, and HDBScan took 0.06 seconds to complete. While these times are all very similar and would not make a difference to a user, these are important factors to take into consideration when experimenting with the code.
4.2 Frontend Design

The design for the front end, models the existing search experiment page [7], by adding a recommendation experiments page. First, the user can perform a search using the existing search algorithm. Then, to differentiate their results, the user has the option between using one of the three clustering algorithms – KMeans, DBScan, and HDBScan – on the results. The user must specify the number of documents to use in the algorithm, as well as the number of clusters if KMeans is selected, and the minimum cluster size if DBScan or HDBScan is selected. Finally, there is an option for PCA to be applied in addition to the already applied dimensionality reduction.

After the user fills out the form, for each of the first $n$ documents that are specified to be used, the two nearest chapters are appended to the list of chapters. For example, if a user specifies twenty documents to be used, for each of these twenty documents, the two nearest chapters will be added to the output, providing similar chapters that the search query might not have selected. This page allows users who are interested in the outputs between these various clustering algorithms to evaluate the differences.
5 Implementation

5.1 Overview

We are implementing a system where users can search through a database of over 5000 ETDs and receive results based on the user’s query. Using these search results, we expanded upon the existing recommendation system by running a clustering algorithm on the output. The user has the flexibility to determine which clustering method to use. We used work completed by previous teams to guide us through connecting the back-end development with the front-end website. Our approach to achieving our goals was to build and connect services for each of the following tasks:

- A front-end service to let the user enter a query to search and show the relevant ETD chapters
- A search to index through relevant chapters
- A recommendation system that lets the user choose a clustering algorithm and show relevant chapters based on the chosen algorithm and their query

5.2 Connecting Recommendation Systems to Website

When working on our front-end development, we used a previous team’s search service and created our own front-end and recommendation services to expand their work.

The front-end service uses React and MaterialUI to create a form on the website to allow users to query the database. With our system, users can apply a clustering algorithm to their search results to identify gaps in the search mechanism.

The search service uses ElasticSearch [14] to index through the collection of ETD chapters and match relevant keywords in the query to keywords in the database. The search service gets relevant chapters by using the Python requests library to query the curator team’s GET ETD API [7]. The search is implemented through an Axios POST request to call a REST API created by a previous team.

Our recommendation service preprocesses data and creates chapter embeddings using SentenceTransformer and the `distilbert-base-nli-stsb-mean-tokens` model [9]. The service also performs a clustering algorithm (KMeans, DBScan, or HDBScan) based on the user’s choice and sorts the cluster based on their distances from a particular chapter in an ETD.

These services call on one another to get data from and to the user. The user uses the front-end service to fill in the input fields. When the user enters a query, the search mechanism returns a list of matching chapters. When the user fills in the clustering form, the recommendation service indexes through the search results and appends relevant chapters to the output based on the clustering algorithm and PCA options chosen.
6  Testing & Evaluation

As discussed previously, we evaluated the accuracy of different clustering algorithms and dimensionality reduction before implementing our final system. By analyzing the data by hand, we saw patterns that wouldn’t have been reported otherwise. This data analysis helped us determine in which manner to proceed.

6.1  KMeans Evaluation

The elbow method is primarily used to evaluate the KMeans clustering algorithm to determine the best number of clusters to use. We can plot the sum of the squared errors after the centroids converge. This is defined as the sum of the squared Euclidean distances of each point to its closest centroid. The lower the SSE, the better the clustering algorithm performed [21]. The lowest SSE value is found by calculating the inertia of the clustering fit. Figure 7 shows the elbow plot. We can find the optimal number of clusters when the plot begins to plateau.

![Figure 7: Elbow Plot for KMeans](image)

Sometimes, it is difficult to determine exactly where the elbow is, so we can use the Python package kneed to determine the elbow [22]. Using this package, we found that the optimal number of clusters is 7.

Another way we can evaluate the KMeans clustering algorithm is by using the silhouette coefficient. This is a measure of cluster cohesion and separation. This coefficient is based on two factors: how close the data point is to other points in the cluster and how far away the data point is from other points in other clusters. The higher the silhouette coefficient, the better the clustering algorithm performed [21]. Figure 8 shows the silhouette coefficient for cluster sizes 2 - 20.
This graph isn’t as clear as the elbow plot and indicates that 9 or 13 clusters may be a better representation of our data. Since we know that there are a large number of summaries that must be clustered, we have to use these metrics in conjunction with our own knowledge to determine the best cluster arrangement. Figure 9 shows the KMeans clustering with 7 clusters. Figure 10 shows the KMeans clustering with 9 clusters, and Figure 11 shows the KMeans clustering with 13 clusters.
After visually analyzing the different clusters derived from KMeans, it appears that nine clusters allows for the best arrangement of chapter summaries without taking away from the integrity of each cluster. Figure 10 demonstrates the ability of the clustering algorithm to correctly sort noise into differing clusters, which was not previously shown in Figure 9. Finally, in Figure 10, some of the clusters on the left side of the graph are divided into more clusters than necessary. Therefore, the nine clusters appear to best represent the data.
6.2 DBScan Evaluation

To evaluate DBScan, we can compute a similar metric to those for KMeans. Using the Silhouette Coefficient method, we can set the correct epsilon level and the minimum number of points per cluster, by running a for loop through epsilon levels from 0.2 to 0.5 and minimum points between 2 to 13. The higher the silhouette coefficient, the better the clustering algorithm performed [21]. In Table 1, until an epsilon level of 0.3 is used, the number of clusters is unreasonable. However, an epsilon value of 0.3 begins to yield a higher silhouette score. Finally, with an epsilon value of 0.4, the number of clusters again becomes unreasonable until a minimum cluster size of 11 is required. While these silhouette scores are not very high, we do have to keep in mind the large number of chapters we are clustering. Therefore, the best combination of epsilon values and minimum points is 0.4 and 13.
<table>
<thead>
<tr>
<th>no_of_clusters</th>
<th>silhouette_score</th>
<th>epsilon_values</th>
<th>minimum_points</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>-0.070603</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>26</td>
<td>-0.085796</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>0.090311</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>0.161827</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>0.195047</td>
<td>0.2</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>0.063987</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>-0.175631</td>
<td>0.2</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>-0.286733</td>
<td>0.2</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>-0.339403</td>
<td>0.2</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0.019093</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>-0.003701</td>
<td>0.2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0.025023</td>
<td>0.2</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>-0.337994</td>
<td>0.3</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>-0.286422</td>
<td>0.3</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>-0.178819</td>
<td>0.3</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>-0.199951</td>
<td>0.3</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
<td>0.150273</td>
<td>0.3</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>0.148937</td>
<td>0.3</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>0.098961</td>
<td>0.3</td>
</tr>
<tr>
<td>19</td>
<td>6</td>
<td>0.124691</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>0.196560</td>
<td>0.3</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>0.084210</td>
<td>0.3</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>0.029186</td>
<td>0.3</td>
</tr>
<tr>
<td>23</td>
<td>12</td>
<td>0.061092</td>
<td>0.3</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>0.117353</td>
<td>0.4</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>-0.095729</td>
<td>0.4</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>-0.140061</td>
<td>0.4</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>-0.133362</td>
<td>0.4</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>0.225088</td>
<td>0.4</td>
</tr>
<tr>
<td>34</td>
<td>4</td>
<td>0.226745</td>
<td>0.4</td>
</tr>
<tr>
<td>35</td>
<td>4</td>
<td>0.249850</td>
<td>0.4</td>
</tr>
</tbody>
</table>
As noted previously, Figure 12 represents the best possible combination of epsilon values and minimum points: 0.4 and 13. Another combination we can test is an epsilon value of 0.3 and a minimum cluster size of 6. Figure 13 represents this output.

Figure 12: DBScan with Epsilon of 0.4 and Minimum Cluster Size: 13
As you can see, a lower epsilon value simply makes the algorithm identify more noise. Overall, the bulk of each cluster is retained, with small, outlier clusters being identified as the minimum cluster size goes down.

6.3 HDBScan Evaluation

To evaluate HDBScan, we followed a tutorial on tuning hdbscan [23]. In the article by Charles Henzel, he talks about using Density Based Clustering Validation, which is a tool for evaluating density-based clustering algorithms, such as hdbscan. DBCV is useful for clustering algorithms like HDBScan because it takes noise into account and cares more about the shape of clusters rather than the distances from each other [23].
Figure 14: HDBScan with 500 data points, min_cluster_size of 3, and min_samples of 10

We were also able to visualize our dataset as seen in Figure 14. The coverage from running HDBScan was around 86%, which meant that coverage was good but there was room for improvement. We found that changing some parameters like `min_cluster_size` and `min_samples` had an effect on the number of noise points in accordance with the number of data points.
Figure 15: HDBScan with 1000 data points, min_cluster_size of 3, and min_samples of 10

As we increased the number of data points when running HDBScan, we found that there were only so many data points we could use until we had to change the min_cluster_size and min_samples parameters. We found that the data would still be intact at 1000 data points and coverage even increased to around 94%.

Although 1000 data points was great in covering more data points, we found that increasing it to 1500 data points but keeping min_cluster_size and min_samples the same would make the coverage go all the way down to around 54%. So, changing these two parameters as data points increase would be a good idea to get more coverage from running HDBScan on larger datasets.
7 User’s Manual

After logging into the website and attempting to create a profile, the previous team implemented a profile creation page for the user (see Figure 16). Users enter their first name, last name, university, department, etc. The topics the user chooses as their interests help fuel the recommendation system. These terms come from the topic modeling toolkit of the previous team. They can choose a wide variety of topics, from STEM to liberal arts. Topics can be as similar or as different as the user wants them.

![Profile Creation Page](image)

Figure 16: Profile Creation Page [7]

After a user logs in, they are prompted to the home page, as seen in Figure 17. The previous team implemented the recommendation system to greet a user with recommendations based on the topics they chose during the profile creation step. They can also search the database with keywords for ETDs they may be interested in.
Figure 17: Home Page [7]

7.1 Recommendation Experiments Page

Figure 18 is the recommendation experiments page. This page was modeled after Figure 9 and, after performing a search, gives the user the ability to use the KMeans, DBScan, or HDBScan clustering algorithms on the search output, shown in Figure 19. The user must specify the number of documents to use, the number of clusters (KMeans), the minimum cluster size (DBScan and HDBScan), and an option to use PCA. After clicking on the search button, users can view the results set by the parameters. The results show the ETD ID, title, and chapter. Users can then click on "View Document" to get to a new page displaying the expected chapter. Figures 20 and 21 show the recommended chapters after choosing a clustering algorithm (KMeans in Fig. 20 and DBScan in Fig. 21).
### Recommendation Experiments

#### Choose Clustering Algorithm

- **Clustering Algorithm**
- **Number Of Clusters**
- **Number of Documents**
- **Minimum Cluster Size**
- **PCA**

*"UMAP is already applied"

#### Message:

<table>
<thead>
<tr>
<th>ID</th>
<th>Title &amp; Chapter</th>
<th>Document Viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>204115</td>
<td>On Building Generalized Learning Agents: Chapter 5</td>
<td>View Document</td>
</tr>
<tr>
<td>210880</td>
<td>The Child as an Active Learner: Chapter 4</td>
<td>View Document</td>
</tr>
<tr>
<td>204919</td>
<td>Applications of Many-Term Quantum Computers: Chapter 8</td>
<td>View Document</td>
</tr>
<tr>
<td>197623</td>
<td>Behavior of Machine Learning Algorithms in Adversarial Environments: Chapter 2</td>
<td>View Document</td>
</tr>
<tr>
<td>216424</td>
<td>Exploratory model analysis for machine learning: Chapter 2</td>
<td>View Document</td>
</tr>
<tr>
<td>202123</td>
<td>Testing Innovations in Machine Learning-Based Decision Models: Chapter 1</td>
<td>View Document</td>
</tr>
<tr>
<td>201995</td>
<td>Computational Tradeoffs in Statistical Learning: Chapter 1</td>
<td>View Document</td>
</tr>
<tr>
<td>195046</td>
<td>System Design for Large Scale Machine Learning: Chapter 2</td>
<td>View Document</td>
</tr>
<tr>
<td>195045</td>
<td>System Design for Large Scale Machine Learning: Chapter 1</td>
<td>View Document</td>
</tr>
<tr>
<td>204774</td>
<td>Statistical Learning Towards Generalization in Human-Centric Cyber-Physical Systems: Chapter 6</td>
<td>View Document</td>
</tr>
</tbody>
</table>

Figure 18: Recommendation Experiments Page

Figure 19: Recommendation Experiments Page after Search
Figure 20: Recommendation Experiments Page after KMeans Clustering

Figure 21: Recommendation Experiments Page after DBScan Clustering
8 Developer’s Manual

8.1 Methodology

We follow Dr. Prashant Chandrasekar’s methodology to create a system and list out the following users, goals, tasks and services.

![Diagram of Methodology]

Figure 22: Methodology

8.2 Retrieving Chapter Summaries

The first technical task that needed to be completed in our project was collecting the chapter summaries that the chapter summarization team had previously created using an NLP model [8]. Using their supplied API, we sent a request to “https://team-1-flask.discovery.cs.vt.edu/v1/etds/<etd_index>/objects?type=cleaned_text” to retrieve the JSON object with the summary present. See Figure 23 for the output we received.
We created a Python script to not only retrieve this output but also parse and organize the summary text so we don’t have to manage the metadata that accompanies a GET request. The code in Figure 24 expedited this process for us and saved and uploaded the data in “etd_chap_summaries.csv” to our server.
8.3 Integrating Recommendation Systems

The next step to setting up the chapter summaries is generating embeddings for each chapter summary. We will use 5000 chapter summaries and translate the text representation to a vector representation.

Using the sentence-transformers package, we can create embeddings of a subset of the chapter summaries in Figure 25 [9].
After we collected the embeddings for the chapter summaries, we had two tracks in which we used this information. The first track involved implementing dimensionality reduction using UMAP and then running a clustering algorithm. Alternatively, we could run the clustering algorithm directly on the embeddings. We attempted these two methods to determine if dimensionality reduction significantly improved the recommendations. Ultimately, we determined that dimensionality reduction yielded an improvement as demonstrated previously in Figures 2 and 3.

8.4 Clustering Algorithm

The penultimate step for our recommendation system was a clustering algorithm. As just mentioned, we attempted this either after we ran a dimensionality reduction algorithm or immediately after we created our embeddings. We worked with the following clustering algorithms: KMeans, DBScan, and HDBScan.

In Python, we created a function (see Figure 26) that used user input to determine which clustering algorithm to run, as well as whether or not PCA is applied to the data and how many clusters to use. The output of this function is a list of the assigned cluster labels and a distance matrix for all of the ETDs.

```python
# clustering algorithm, accepts embeddings of data, the number of clusters and if UMAP should be used # returns list of cluster labels and distance matrix
def clustering(data, num_clusters, min_cluster_size, clustering_algo, use_PCA):
    distances = []
    if (use_PCA):
        # Reduce the data using PCA
        pca = PCA(.75).fit(data)
        data = pca.transform(data)

    clusterable_embeddings = umap.UMAP(
        n_neighbors=30,
        min_dist = 0.0,
        n_components = 2,
        random_state = 42,
    ).fit_transform(data)

    # Get distance matrix
    df = pd.DataFrame(clusterable_embeddings)
    distances = pd.DataFrame(distance_matrix(df.values, df.values))

    if(clustering_algo == "KMeans"):
        # Define kmeans model
        clustering_model = KMeans(n_clusters=num_clusters, n_init = 10)
        # Fit the embedding with kmeans clustering.
        clustering_model.fit(clusterable_embeddings)

    elif (clustering_algo == "DBScan"):
        clustering_model = DBSCAN(eps=0.3, min_samples=min_cluster_size).fit(clusterable_embeddings)

    elif (clustering_algo == "HDBScan"):
        clustering_model = hdbscan.HDBSCAN(min_cluster_size=3, min_samples=min_cluster_size,
                                            metric='euclidean', cluster_selection_method='eom').fit(clusterable_embeddings)

    return clustering_model.labels_, distances
```

Figure 26: Clustering Function
8.5 Sorting the Clustering Output

Since we use our clustering output to leverage the search system, we determined we needed to have the ability to sort the output of the clustering algorithm. This function (see Figure 27) accepts an ETD_ID, Chapter_ID, all of the ETDs, and the distance matrix. It returns a list of title and chapter numbers sorted by their distance to the ETD and chapter specified.

```python
def sort_cluster(etd_id, chapter_id, data, clusterables):
etd_to_sort = data.loc[(data['ETD_ID'] == int(etd_id)) & (data['Chapter_ID'] == int(chapter_id))]
if (etd_to_sort.empty == False):
    cluster_id = (etd_to_sort).iloc[0,4]
    index_of_etd = etd_to_sort.index.values.astype(int)[0]

    in_cluster = data.loc[data['cluster'] == cluster_id]
    out_cluster = clusterables.loc[in_cluster.index]
    out_list = out_cluster[index_of_etd]
    final_list = out_list.sort_values().index
    ret = []
    for val in final_list:
        ret2 = {"id": str(data.get_value(val, 'ETD_ID')),
                "etd_title": data.get_value(val, 'Title'),
                "title": str(data.get_value(val, 'Chapter_ID'))}
        ret.append(ret2)
    return ret
```

Figure 27: Sorting Function

8.6 Preparing Output for Integration with Front-End

To show the nearest two chapters to each chapter from the search output, we use a controller to retrieve the form output from the front end and then retrieve the two nearest using our sorting function (see Figure 28). The output is transformed to be JSON readable for the front end.
To allow the user to search on our recommendation experiment page, we used the previous team's ElasticSearch method for searching on the website. Then, using the Material UI package the previous team used in their front-end implementation, we created an HTML form that accepts $n$, the number of documents to use in the clustering algorithm, the number of clusters (KMeans), minimum cluster size (DBScan/HDBScan), whether to use PCA, and which clustering algorithm to use. If the HTML form is filled out, the clustering algorithm is applied to the search output. Then, for each of the $n$ documents, the two nearest chapters will be appended to the list.

```python
from flask import Flask, request, send_from_directory, jsonify
from flask_restful import Resource, Api
import sys
import subprocess
import json
import os
import cmd
from shutil import copyfile
import pandas as pd
from sentence_transformers import SentenceTransformer
import rec_exp_service as rec_exp_service
etd_chapters = pd.read_csv("../data/etd_chap_summaries-4.csv", nrows = 5000)
embedder = SentenceTransformer('distilbert-base-nli-stsb-mean-tokens')
summary_embeddings = embedder.encode(etd_chapters['Summary'])

sys.path.append("..")
app = Flask(__name__)
api = Api(app)
todos = {}

class Cluster(Resource):
    def post(self):
        data = request.json
        numDocuments = data["num_documents"]
        numClusters = data["num_clusters"]
        minClusterSize = data["min_cluster_size"]
        algo = data["algChosen"]
        pca = data["PCA"]
        searchOutput = data["searchOutput"]

        #use all chapters to cluster (prob should make only happen once)
        output = rec_exp_service.clustering(summary_embeddings, numClusters,minClusterSize, algo, pca)
        etd_chapters["cluster"] = output[0]

        #look at the first n documents and find the two closest
document_ids = []
        for i in range(numDocuments):
            etd_id = str(searchOutput[i]["etd_id"])
            closest = rec_exp_service.sort_cluster(searchOutput[i]["etd_id"],
                                                  searchOutput[i]["title"], etd_chapters, output[1])
            document_ids.append(searchOutput[i])
            if closest != None:
                document_ids.append(closest[1])
            document_ids.append(closest[2])
        return jsonify(document_ids)

api.add_resource(Cluster, '/experiment/recommendation/cluster')

@app.after_request
def after_request(response):
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Headers', 'Content-Type,Authorization')
    response.headers.add('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS')
    return response

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

Figure 28: Recommendation Experiment Controller

8.7 Recommendation Experiment Front-End

To allow the user to search on our recommendation experiment page, we used the previous team's ElasticSearch method for searching on the website. Then, using the Material UI package the previous team used in their front-end implementation, we created an HTML form that accepts $n$, the number of documents to use in the clustering algorithm, the number of clusters (KMeans), minimum cluster size (DBScan/HDBScan), whether to use PCA, and which clustering algorithm to use. If the HTML form is filled out, the clustering algorithm is applied to the search output. Then, for each of the $n$ documents, the two nearest chapters will be appended to the list.
As the previous team developed this website, this is their explanation for developing with the front end.

“The developer needs to have Node.js installed in their system. After installation, the developer needs to clone the project and run the following commands:

```bash
npm i
npm start
```

The first command installs all the dependencies for the project. Next, the second command creates the development web server and starts the application. The application can then be accessed on http://localhost:3000” [7].

### 8.8 Gitlab Repository

[https://code.vt.edu/taliamb/etd-recommendation-system](https://code.vt.edu/taliamb/etd-recommendation-system)
9 Lessons Learned

9.1 Timeline

Refer to Table 2 for a brief overview of the tasks we completed over the course of this semester.

<table>
<thead>
<tr>
<th>February 2023</th>
<th>March 2023</th>
<th>April 2023</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Research recommendation systems</td>
<td>- Continue working on recommendation systems</td>
<td>- Continue working on front-end endpoint for recommendation experiments</td>
</tr>
<tr>
<td>- Answer initial questions</td>
<td>- Attempt dimensionality reduction</td>
<td>- Test recommendation models</td>
</tr>
<tr>
<td>- Determine scope of project</td>
<td>- Work on clustering algorithms</td>
<td>- Start work on report/presentation</td>
</tr>
<tr>
<td>- Collect chapter summaries</td>
<td>- Test recommendation models throughout</td>
<td>- Conclude front-end work</td>
</tr>
<tr>
<td>- Create embeddings of chapter summaries</td>
<td>- Start integrating recommendation models &amp; existing websites</td>
<td>- Conclude back-end work</td>
</tr>
<tr>
<td></td>
<td>- Start a front-end endpoint for the recommendation experiments in website</td>
<td>- Work on report &amp; presentation</td>
</tr>
</tbody>
</table>

9.2 Problems and Solutions

Despite the progress made by our team during this semester, we encountered many problems throughout that impeded our ability to complete the project as we had hoped.

When we began this project in January, we were under the impression that we would be using user behavior to dictate recommendations to the user, specifically dwell time and clicks to external links. However, after meeting with our client and fleshing out the details of our project, we determined that with our limited team size, it would not be feasible for us to have such a lofty goal for this semester. After meeting with Dr. Fox, our client reevaluated the project. Since the previous team used document abstracts to fuel their recommendation system, he proposed using chapter summaries. Previously, a chapter summarization team created a model to generate chapter summaries for over 5000 ETD chapters [8]. He believed this would foster a narrower, more accurate recommendation to the user. By rehashing our project, we found a way to use the lessons learned by the previous team to efficiently create a recommendation system that used chapter summaries rather than document abstracts.

Additionally, as we became more well-versed in our project, we needed to consider whether our system would aid the current recommendation system with clustering algorithms or act as its own separate entity. After meeting with Dr. Fox, Mr. Chekuri decided that serving as a separate entity would be better suited for this project. Therefore, we created a recommendation
experiments page that applies a clustering algorithm to the search output. This is a separate web page that allows users to visually see the impact of a clustering algorithm and identify gaps in the search algorithm.

9.3 Future Suggestions

Similar to the last group that worked on the recommendation system portion of this project, we suggest that future group members leverage user behavior to create a recommendation system that learns from the user. We believe that in conjunction with using our work with chapter summaries, this will generate more traffic to the website due to the accuracy of the system.

Another development we suggest future teams work on is dynamically responding to new ETDs. As mentioned before, we use chapter summaries that a past team generated. If our system could respond to users uploading new documents, the whole system would be more flexible. If the system can handle more ETDs being uploaded, then we can generate more traffic for those documents.

Finally, we believe our recommendation system is more accurate and efficient than the current system because it uses chapter summaries rather than abstracts. As discussed previously, we decided against aiding the current recommendation system so that our deliverables would be separate. However, if given more time, we would have liked to leverage our clustering algorithm in the current system rather than as a separate entity.
10 Acknowledgements

Satvik Chekuri & Professor Fox

We would like to thank Satvik Chekuri and Professor Fox for their roles in our project. We extend our gratitude to Satvik for explaining the project and getting the proper resources to help us to succeed in the project. We would also like to thank Professor Fox for proposing a project that we were interested in and helping us get to a good start by implementing helpful resources like OKRs, team discussions during class time, and interesting guest lectures.

Previous teams

We would also like to thank the previous teams whose work contributed extensively to our project: Team 2 and Team 4 from CS 5604 Fall 2022. Team 2 [7] was the End User team and Team 4 [8] was the Language Models and Summarization/Classification/Segmentation team. Their efforts from last semester have helped us with important tasks in our project and have made some parts of our project easier.
11 References


no. 11, Taylor & Francis, 1901, pp. 559–572,


*Medium*, Towards Data Science, 8 Sept. 2021,


[22] Arvai, Kevin. *Kneed Package*, kneed. 0.7.0, 8 2020,


12 Appendix

Methodology

The user-centered design methodology -- covering goals, tasks, subtasks, services, and workflows -- underlying this project is summarized in the description below, along with Figures 29 and 30, and Table 3.

Goals of each of the types of users the system needs to support:

- Interface that provides a recommendation system, giving users an assortment of theses that relate to the ones that the user has searched for based on
  - KMeans Clustering Algorithm
  - DBScan Clustering Algorithm
  - HDBScan Clustering Algorithm

- Interface that allows users to experiment with different recommendation models to understand the results by manipulating the:
  - Clustering Algorithm
  - Number of clusters/minimum cluster size
  - Whether or not PCA is applied

Break-down of goals into tasks and subtasks that achieve the goals:

- To support the goal of using various clustering algorithms to fuel a recommendation system the system needs to support the tasks of:
  1) Collecting all of the chapter summaries and encoding them into embeddings to be clustered
  2) Run applicable clustering algorithm and determine which cluster a certain chapter is located

Figure 29: Goal 1 Workflow- Methodology
To support the goal of allowing users to experiment with different recommendation models to understand the results by manipulating the parameters of the clustering algorithm the system needs to support the tasks of:

1) Retrieve user input for the clustering algorithm to use, number of clusters/minimum cluster size and whether to use PCA
2) Run clustering algorithm with user input

Figure 30: Goal 2 Workflow - Methodology

Table 3: Methodology

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Name</th>
<th>Input file name(s)</th>
<th>Input file IDs (comma-sep)</th>
<th>Output file name</th>
<th>Output file ID</th>
<th>Libraries; Functions; Environments</th>
<th>API endpoint (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>Collecting chapter summaries &amp; create embeddings</td>
<td>5k_Segmented_ETD_list.xlsx</td>
<td>1A_I</td>
<td>etd_chap_summaries.csv</td>
<td>1A_O</td>
<td><a href="https://team-1-flask.discovery.cs.vt.edu/v1/etds/etd_index/objects?type=cleaned_text">https://team-1-flask.discovery.cs.vt.edu/v1/etds/etd_index/objects?type=cleaned_text</a></td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Task Description</td>
<td>Data Source</td>
<td>Parameters</td>
<td>Library</td>
<td>Algorithm</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------</td>
<td>----------------------</td>
<td>------------</td>
<td>---------</td>
<td>-----------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>Run clustering algorithm</td>
<td>etd_chapter_summaries.csv</td>
<td>1A_O</td>
<td>n/a</td>
<td>Scikit-learn Dbscan Hdbscan</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>Determine where a chapter is located and sort by distance</td>
<td>n/a</td>
<td>distance_s.txt</td>
<td>1C_O</td>
<td>Distance matrix</td>
<td>Recommendation experiments page</td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>Retrieve user input</td>
<td>n/a</td>
<td>user_input.json</td>
<td>2A_O</td>
<td>JSON library</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>Run clustering algorithm using user input</td>
<td>user_input.json</td>
<td>n/a</td>
<td>n/a</td>
<td>Scikit-learn Dbscan Hdbscan</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>Determine where a chapter is located and sort by distance</td>
<td>n/a</td>
<td>distance_s.txt</td>
<td>2C_O</td>
<td>Distance matrix</td>
<td>Recommendation experiments page</td>
<td></td>
</tr>
</tbody>
</table>

**Goal 1 (Interface for recommendation system based on clustering algorithm):**
Recommendation system using clustering algorithm = Collecting chapter summaries & create embeddings(1A) + Run clustering algorithm(1B) + Determine where a chapter is located and sort by distance (1C)

**Goal 2 (Interface for recommendation system based on a user input to a clustering algorithm):**
Recommendation system using user input = Retrieve user input (2A) + Run clustering algorithm using user input (2B) + Determine where a chapter is located and sort by distance (2C)