Improving User Experiences for Wiki Systems

A Project Report

Presented To

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the

Degree of Master of Science

By
Parth Amrutbhai Patel

Dec 2022

Improving User Experiences for Wiki Systems

By
Parth Amrutbhai Patel

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

DEC 2022

Dr. Christopher Pollett	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Philip Heller	Department of Computer Science

ACKNOWLEDGEMENT

I want to express my gratefulness to my project advisor Dr. Chris Pollett for the continuous guidance and support during the creation of this master's project over the past one year. I am heavily indebted to him for the skills and knowledge gained while working with him. I would also like to appreciate other faculties in Department of Computer Science at San José State University for their efforts and guidance while teaching the advanced computer science courses. Lastly, I would like to thank my family and friends for their constant care and encouragement during my journey for completing Master of Science in Computer Science degree.

ABSTRACT

Wiki systems are web applications that allow users to collaboratively manage the content. Such systems enable users to read and write information in the form of web pages and share media items like videos, audios, books etc. Yioop is an open-source web portal with features of a search engine, a wiki system and discussion groups. In this project I have enhanced Yioop's features for improving the user experiences. The preliminary work introduced new features like emoji picker tool for direct messaging system, unit testing framework for automating the UI testing of Yioop and redeeming advertisement credits back into real money. The major work in the project was to improve and extend the recommendation system of Yioop to recommend media items (referred to resources in Yioop) of wiki pages having different formats like video and text. This was achieved by developing a media job that can gather the description for resources by searching through various configured web search sources using just the name of resources. Then the recommendation media job utilizes these descriptions by applying Hash2Vec term embedding algorithm to generate linear embedding vector for resources and leverage those vectors to generate linear vector for users representing their history of consuming resources. Scores are calculated using cosine similarity between the vector of a user and the embedding vector of resources not consumed by that user, ultimately recommending resources in descending order of this score. Experiments were conducted to compare the results of older and newer version of recommendation systems.

Keywords – Wiki systems, Media jobs, Hash2Vec embedding, Yioop, Recommendation

TABLE OF CONTENTS

I. INTRODUCTION

A wiki system provides an efficient way of creating, consuming, and managing of information through web pages. It often allows collaboration among users by providing facilities for creating groups and discussion boards for common communication among group members. Yioop, an open-source web portal, is one such wiki system developed using PHP which additionally has a search engine functionality. Users can create groups and configure rules of joining the group and the access level of the group members. Inside a group, users can create pages called wiki pages allowing them to create textual information and upload media items regarded as resources. Every group has a discussion board where users can start new threads and post comments to the existing threads.

For any wiki system it is important to establish positive user experiences. The primary elements of any wiki system that affect user experience are the User Interface (UI) and the features or functionalities provided to the user. The other important factor for wiki systems is to ensure that a user can easily find the information relevant to their taste and need from the huge amount of data generated in the system. This is where a recommendation engine is required that can filter out the irrelevant data based on users' history of consuming system resources and recommend information that satisfies the taste or need of users.

At present almost all web applications like Amazon, Netflix, Twitter, etc. with large number of users and resources have functionality of recommendation engine. Yioop uses a term frequency – inverse document frequency (TF-IDF) based collaborative recommendation system which can recommend unconsumed groups and discussion threads to the users. However, this

recommendation system lacks support for generating recommendations for wiki resources and the recommendations generated for groups and threads are inefficient because it recommends only popular items and requires longer time for processing the data.

This project attempts to improve the existing Yioop's recommendation system by redesigning the recommendation logic to use a novel Hash2Vec approach for generating the recommendations for groups and threads and extend it to calculate the recommendations for wiki resources using their descriptions. The newer version of the recommendation system will also reduce the processing time thus making it suitable for running Yioop server on devices with low processing capabilities.

The remainder of this report is organized as follows: The next section covers the background about the existing recommendation system in Yioop and the related work done using Hash2Vec algorithm. The preliminary work section covers the details of work done in first half of the project that contributed towards improving the user experiences while using Yioop. The following section discusses the details of Yioop's wiki system and the mechanism for gathering description for wiki resources through web crawling. The enhanced recommendation system section introduces the concept of Hash2Vec algorithm and describes every detail regarding the proposed recommendation system. The experiment section contains the details of experiment conducted to test the working of proposed method and the results of comparison between the recommendations generated by older and newer version. Finally, the last section provides conclusion from the project and scope of future work that can further improve the proposed recommendation system.

II. BACKGROUND

This section discusses about the background concepts necessary to familiarize with the mechanism for the existing Yioop's recommendation system. An overview of the group and threads functionalities in Yioop is given. A brief discussion of the TF-IDF measure and its usage in the existing Yioop's recommendation system is explained. At the end, related work on using Hash2Vec for improving the recommendation system of Yioop will be covered.

A. Groups and Threads in Yioop

Yioop allows users to create a group and configure the access rules for it governing how other users can join that group. A group has a discussion board that collects threads and provides feature to create new threads. A thread contains a title and a description which describes the summary and purpose of that thread. Within a thread users can post comments as shown in Figure 1. Users are allowed to edit or delete a comment or thread based on the access rules of the group.

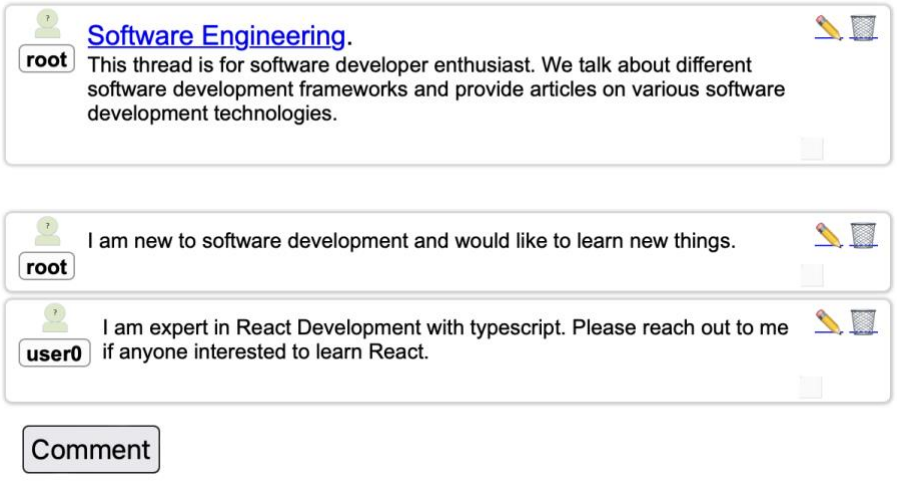

Fig 1. *A Thread in Yioop*

B. Overview of TF-IDF

TF-IDF stands for term frequency – inverse document frequency. It is used in information retrieval settings like search engine and recommendation systems [1]. It is a numeric statistic that marks the importance of a word in a particular document [2]. A document refers to any piece of textual information like a book, a text file or a web page. Term Frequency (TF) is a measure of how frequently a term appears in a document, more frequently occurring terms in a document will have high TF value. There are various ways of calculating TF like raw count of term, logarithm of raw count, etc. Inverse Document Frequency (IDF) is a measure of how common or rare a term is across a collection of documents, which helps to mark common words like “a”, “the”, “is”, etc. unimportant as these terms will have lower IDF value. Finally, the product of TF and IDF is the measure of importance of a term in a particular document.

C. Existing Yioop’s Recommendation System

The existing recommendation system in Yioop is based on a TF-IDF vectorization method. Vectorization refers to the process of converting text data into a vector of numerical data. In TF-IDF vectorization, a document is represented as a linear vector by using the TF-IDF scores of the terms in it. The length of such vectors is equal to number of unique terms in the entire collection of documents.

In order to generate recommendations for threads, each thread title and description is combined to represent one document. Then TF-IDF scores are calculated for all the unique terms in these documents and using TF-IDF vectorization process each thread is represented as a linear vector. Yioop has a mechanism to capture the information regarding the threads viewed by a user, which will be discussed in the later sections. Based on this data, TF-IDF scores for the terms

occurring in the threads viewed by a user are calculated, then using a vectorization process, linear vectors for users are calculated based on TF-IDF scores of terms found in the threads viewed by a user. To measure how closely a thread matches with a user's taste, a cosine similarity score between thread's vector and user's vector was calculated and ultimately threads with top 3 scores were recommended to the user. To calculate a similarity score for a group, the similarity scores of all the threads in that group are summed up and groups with top 3 scores are recommended.

D. Related Work

In [3], Mallya proposes enhancing the existing Yioop's recommendation system by leveraging the Hash2Vec word embedding method. In their work, the TF-IDF scores for terms were supplemented with similarity scores between terms calculated using Hash2Vec method. The reasoning behind this was to capture contextual information for terms which was not captured by TF-IDF method. The experiment conducted showed an increase of 60.28% in F-1 score as compared to the existing recommendation system. But the work did not address the issue of the longer time for processing the threads data to synthesize recommendation data. So, this project will attempt to address that issue by redesigning the core logic of processing the threads data and efficiently synthesize recommendation data.

III. PRELIMINARY WORK

This section is an overview of the work done during the first half of this project which contributed towards enhancing the experience of users while interacting with Yioop. The preliminary work helped the author to familiarize with the Yioop's functionality and its code base. This section contains brief details regarding the new features added to Yioop and ends with a discussion of what was learned about Yioop's recommendation system and how that served as the starting point for later half of the project.

A. Emoji Picker Tool

The first task during the preliminary work implemented an Emoji Picker tool into Yioop's direct messaging functionality. Currently there are more than 1500 emojis supported by the tool, divided into 8 different categories allowing users to easily locate an emoji. Moreover, implementation also supports feature of replacing textual shortcuts of emojis in the messages with the corresponding emoji character. In order to implement this feature, it was necessary to gain familiarity with the Unicode emoji characters and their code points. Every emoji is a Unicode character which has either a specific code point or a sequence of code points [4]. To display an emoji on a web page one could use combination of Hyper Text Markup Language (HTML) codes equivalent to Unicode code point. Based on this information the Emoji Picker tool was designed without any dependency on external code libraries.

B. UI Testing Project

The second task was to create a UI testing project for Yioop as UI is the primary way through which a user can interact with any web application. To ensure rich user experience for a

wiki system, a suite of UI test cases is developed that detect issues in UI elements. The outcome of this deliverable is a separate project based on Selenium with node.js programming language which can test the Yioop's UI in Firefox and Google Chrome web browsers for desktop and mobile version of Yioop. Selenium is an open-source UI testing framework which provides various drivers for different web browsers and language bindings for different programming languages to use it [5]. A shell script was developed which ensures all the requirements of testing project are met before executing the test cases. A HTML report is generated after all the test cases are executed which shows the result of every test cases along with the screen shots captured.

C. Advertisement Credits Redeem

Yioop has a keyword advertisement facility which allows users with business roles to create text-based advertisement campaign for certain number of days. Whenever a user uses Yioop search engine then the appropriate advertisement campaign will be displayed based on the words in search query. In order to start an advertisement campaign user needs to bid for the keywords using *credits*. A credit is worth one US penny and user can purchase it using credit cards in a bundle of 1K, 2K, 5K and 10K credits.

The third task during preliminary work was to allow users to convert their unused credits back to US dollars. This implementation leverages Stripe's Connected Account services which allows to create Stripe accounts for the users. Stripe is a company that provides services to businesses for processing online payments [6]. This feature allowed users to setup the Stripe connected account through Yioop and redeem credits by providing details of their debit card which are handled on Stripe servers with leaving any impression on Yioop server.

D. Understand Existing Recommendation System

The last task was to get familiarized with the existing recommendation system in Yioop as described in background section. This task also involved understanding the work done by Anirudh as described in [3]. The outcome of this task was increased familiarity with the working of existing recommendation system and knowledge of Hash2Vec word embedding technique. This helped to find the issues within existing system and motivation to use Hash2Vec method for improving the existing recommendation mechanism for threads and groups and extend it to generate recommendations for wiki resources.

IV. THE WIKI SYSTEM IN YIOOP

A. Wiki Pages

Yioop has a wiki system which allows users to create, edit and read wiki pages within a group. As shown in figure 2, a standard wiki page resembles a web page in read mode that contains text data, hyperlinks, resources like images or videos, tables and bar-graphs. However, all the data except resources is in textual format organized in a specific syntax which can be parsed during the read mode in order to display it as a web page. Figure 3 shows the wiki page in edit mode where the raw text data is displayed inside a text area which allows to edit the data. Additionally in edit mode, there are options to change the settings of wiki page and upload or create resources in the page.

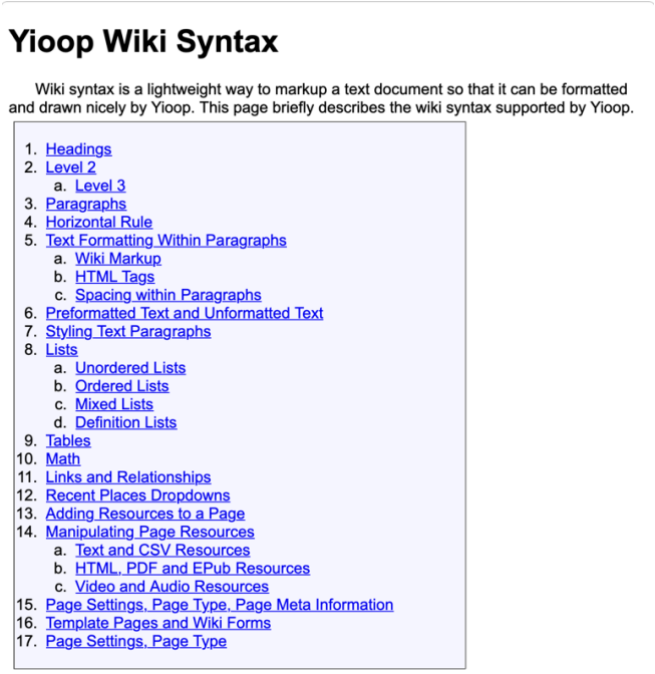

Fig 2. *Wiki page in read mode*

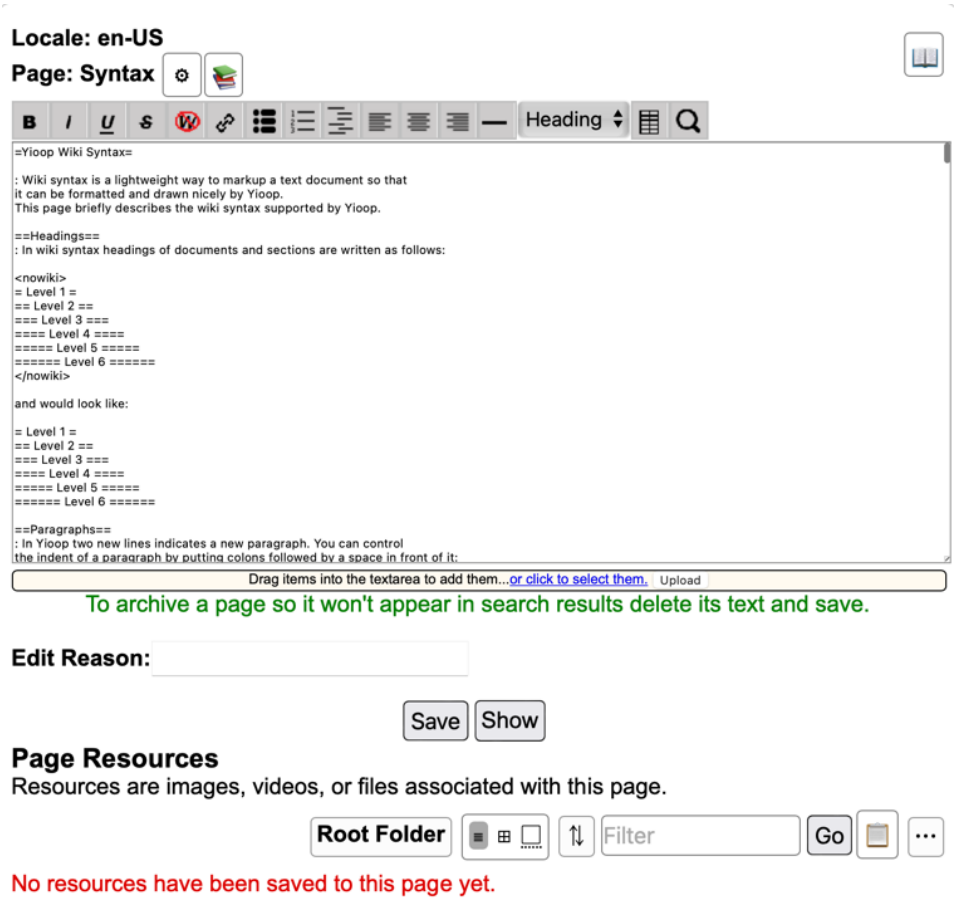

Fig 3. *Wiki page in edit mode*

Yioop site uses database management system (DBMS) to persist data that is created by users while using Yioop. The database of Yioop contains 49 tables and 21 indices to optimize query for faster data retrieval. The raw textual data of wiki pages is stored in GROUP_PAGE table under PAGE column. Yioop also has the feature to preserve the history of a wiki page and allows to go back to any previous version of that page. This is done by storing the current data of wiki page in GROUP_PAGE_HISTORY table when an edit operation is performed on that page and the latest data will be updated in GROUP_PAGE table.

B. Wiki Resources

The resources in a wiki page are stored in local storage file system as shown in figure 4. The root folder for storing resources of all the wiki pages is the “resources” subfolder inside app directory created during the installation of Yioop instance. Within the resources folder a subfolder for each wiki page is created when a resource is created or uploaded for the first time. The name of this subfolder is the first 3 characters of a hash operation on the id of group and id of page mixed with string constant “group” and numeric constant AUTH_KEY that can be configured to any value.

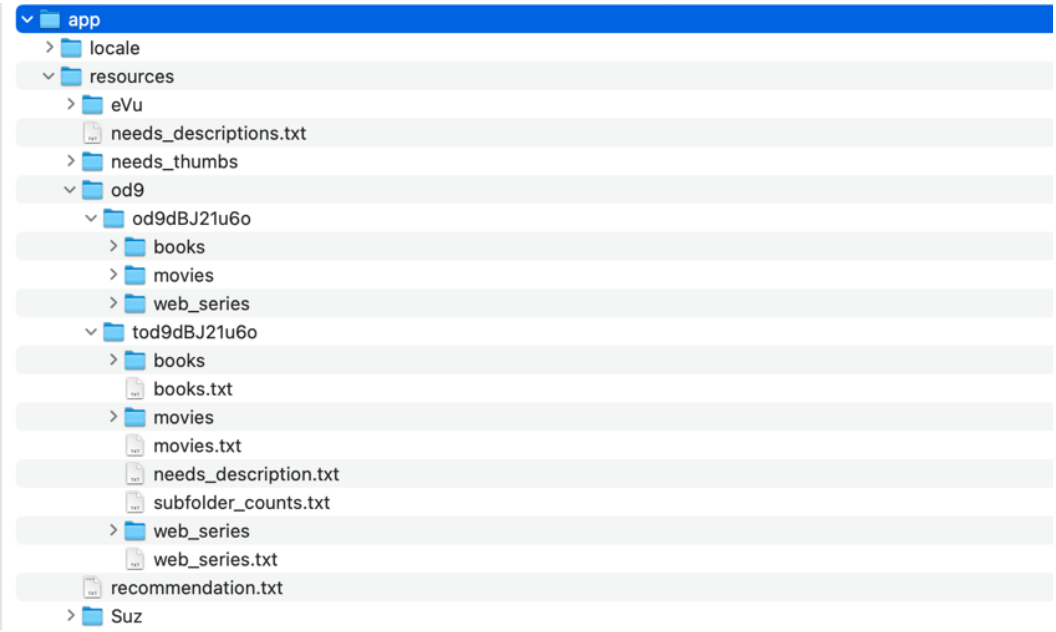

Fig 4. *Resources in file system*

Inside a page subfolder there are 2 folders, one folder named using the hash operation as discussed before contains the actual data files of resource and serves as the root folder for all the sub-folders created within a wiki page. The second folder name is same as the first one with extra “t” character at the start and same sub-folders structure as the other folder and is used to store description for resources in a separate text file per resource. Figure 5 shows resources in detail mode inside a sub-folder of a *media list* type wiki page. A media list wiki page just displays the

resources in that page as a thumbnail and a hyperlink to the web page where the resource data will be displayed depending upon the file format of a resource.

Fig 5. *Detail mode in media list*

C. Description Source

The proposed recommendation system processes descriptions of resources for synthesizing recommendation data for resources. While uploading or creating a new resource, user is allowed to enter description in the detail view mode of the resource. Since adding description for a resource is optional, we cannot rely only on the users as the source for descriptions. To address this concern, a mechanism is proposed to gather descriptions for resources through web scrapping using just the name of resources as the search information.

Yioop provides ability to configure a Search Source as shown in figure 6. There are different types of search source based on the type of information to be processed. To add a search source, a user needs to provide various information depending upon the type of search source. For the use case of gathering resource description, a new search source type *Description Source* was created which has the information field as shown in figure 6.

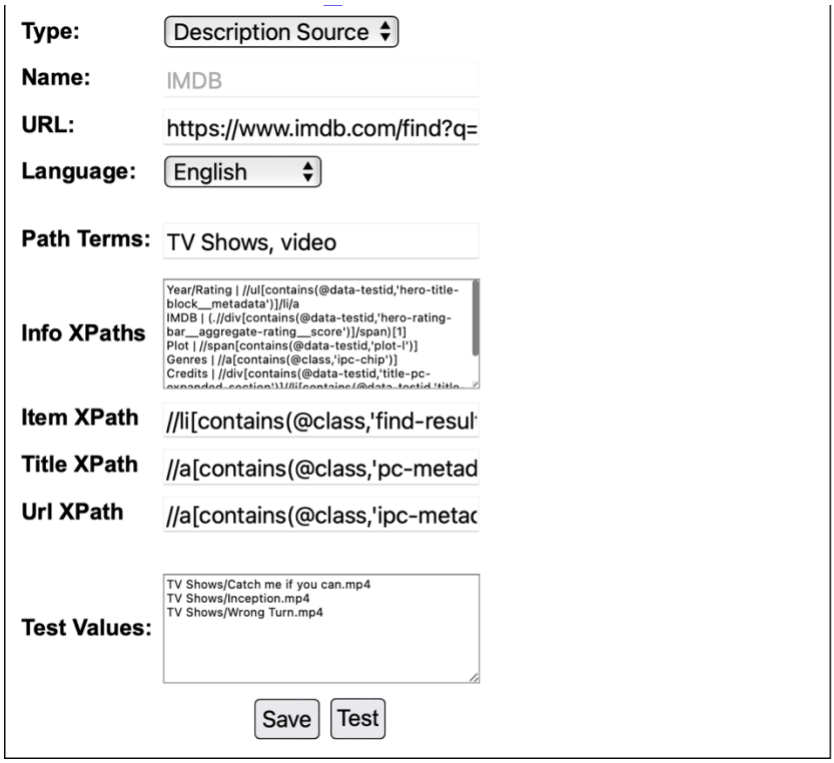

Fig 6. *Description search source*

The URL field is used to specify the URL of a website where the information about a resource can be found using the name of resource. Path Terms field contains a comma separated values which could either be a term in the file system path of a resource or mime type of the resource file either partial or full. The purpose of this field is to specify *media job* covered later, which resources description can potentially be gathered using this search source.

XPath represent either an absolute or relative path of an element with a html page. A search within most of the websites output research results in a html table or list element where each item has similar html structure which makes it easy to process them using a xpath that matches with all the research results, Item XPath field accepts this common xpath value. Title XPath field contains the xpath value of html element that represents the title of a single search result item and URL XPath field contains the xpath of an anchor html element which has hyperlink to the webpage that

contains details of a search result. Info XPath field contains xpaths in the detail webpage, of certain information field that can be part of a resource description.

D. Description Update Media Job

Configuring a description search source is only first task towards gathering the descriptions using that source. It is saved in database under MEDIA_SOURCE table and is not responsible for actual task of performing web scrapping. The main task of performing web search by using a search source is done by a media job. A media job in Yioop represents a background process that is scheduled to run repeatedly at specified time interval. To update the descriptions of resources, a new media job *Description Update* is created which runs at every 1 hour and attempts to fetch description for a resource using the details of configured search sources.

To avoid reprocessing the same resource at every run of this media job, text files named "needs_description" are maintained within each wiki folder and within each sub-folders as shown in figure 4. When a wiki page or a sub-folder within that page is viewed, the information about resources at that path are retrieved from file system. Using the retrieved details, it can be decided if a resource is having a description already or if not then the name of that resource is added to the needs_description file. To further optimize the process, separate needs_descriptions file is maintained at resources folder as shown in figure 4. Each line in this file contains the absolute path of needs_description file inside sub-folders of a wiki page.

During each run of description update media job, the lines from top level needs_descriptions file are read. Then the needs_description file located at the path represented by any line is read and for every name of resources inside that file, suitable search sources decided using the Path Terms field value of search sources are used to sequentially until the description for

the resource is not found. If the job fails to find description after using all the search sources, then a dummy description like “Description search sources failed to find description.” is used to avoid reprocessing that resource during next run of description update media job.

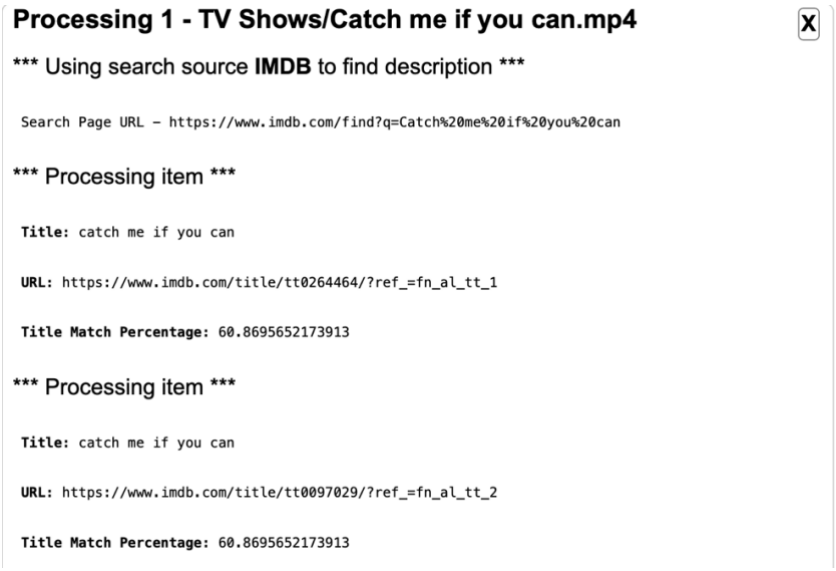

Fig 7. *Title match in Description Update*

As shown in Figure 7, the titles of search results retrieved using a particular search source for a particular resource are matched against the name of resource to calculate similarity percentage and the search result with the highest score is used for fetching the descriptions on the details webpage. Figure 7 shows a part of title matching process during the test mode of the description search source. After processing all the search result items, the URL of the item with the highest score is used to fetch the html content of webpage that contains potential description information. The xpath values in Info XPath are evaluated against the fetched html content and the retrieved data is stored in the description file for the resource under processing. Figure 8 shows an example of fetched description.

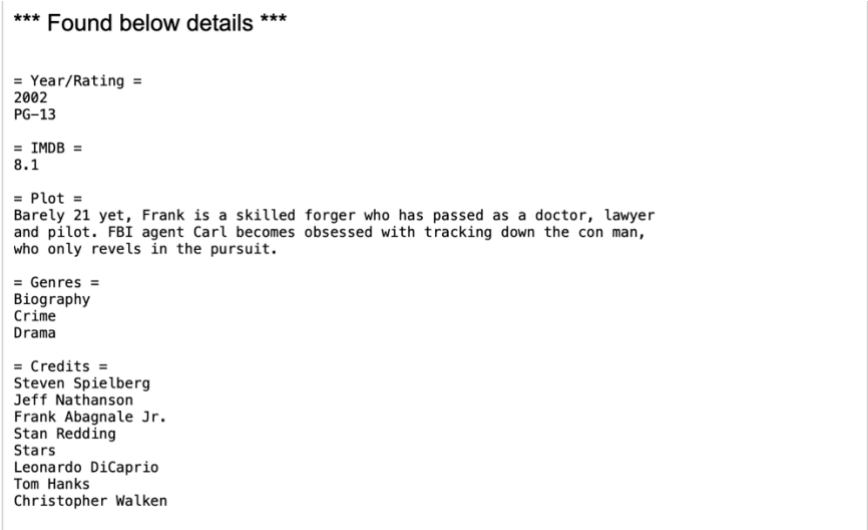

Fig 8. *Example description fetched*

The above proposed mechanism of fetching description through web scrapping using name of a resource was the most novel and challenging aspect for this project as there is no related work where such approach was taken.

V. ENHANCED RECOMMENDATION SYSTEM

This section describes the details of proposed mechanism to enhance the existing recommendation system in terms of quality of generated recommendations and the time required to synthesize recommendation data. The concept of Hash2Vec word embedding technique will be introduced and the results of comparison with other word embedding techniques will be discussed.

A. Word Embedding

In natural language processing (NLP), word embedding refers to the numerical vectors representing a word or term in text corpus [7]. Word embedding is one of the important steps in various NLP tasks as they allow mathematical calculations to be carried out on word embedding vectors of raw terms for achieving various insights. There are various techniques for creating word embeddings like Word2Vec or Glove, TF-IDF, BERT, CBOW, etc. [7]. The less advanced techniques like TF-IDF or CBOW are simple to implement and takes less processing time. However, these techniques do not preserve the contextual information between the terms in the corpus [8] and the generated embedding vectors are sparse which require more space or memory during processing.

More advanced word embedding techniques like Glove or BERT use neural networks for calculating the embedding vectors. These techniques preserve contextual information between terms and produces less sparse embedding vectors. But the neural networks require high end processing servers which have advanced processing units like graphics processing unit (GPU) and even with GPU their processing time are very longer as compared to techniques like TF-IDF. In order to use such techniques in Yioop, a very complex implementation in PHP will be required and the server specification required for running Yioop would increase.

Considering the described issues, the enhanced recommendation system proposes to use Hash2Vec word embedding technique which is not backed by neural networks and as described in [8], the embedding vectors generated using Hash2Vec are able to preserve the semantics or contextual information between the terms.

B. Hash2Vec Word Embedding

Hash2Vec technique utilizes feature hashing to generate embedding vectors for terms in text corpus. Feature hashing refers to the process of applying hash operation to the features in higher dimension to find a new dimension for feature in reduced space. In Hash2Vec, higher dimension refers to the number of unique terms in the text corpus, features are terms itself and lower dimension space represents the resulting embedding vectors which have fixed and smaller size than original vocabulary size. In [9] feature hashing was successfully used to reduce the size of BOW vectors and describes how feature hashing was used to create a classifier to label a mail as spam or ham.

Algorithm 1 Hash2Vec

Parameters: n the embedding size, k the context size, h hash function, ξ hash sign-function and f aging function.

words $\leftarrow$ *Dictionary*()
for every word w in text **do**
  if $w \notin$ keys(words) **then** words[w] $\leftarrow$ Array(n)
  for every context word cw with distance d **do**
    weight $\leftarrow f(d)$
    sign $\leftarrow \xi(cw)$
    words[w][$h(cw)$] $\leftarrow$ words[w][$h(cw)$] + sign $\times$ weight

Fig 9. *Hash2Vec simplified algorithm [5]*

Figure 9 shows the simplified version of Hash2Vec algorithm. Parameter n refers to the size of resultant embedding vectors, k is the size of context window for a term. Context window for a term consists of past k and next k words considered during feature hashing, this way Hash2Vec embeddings preserve semantic information of a term by considering the contextual information. The hash function *h* could be any hash function that converts a text term into numeric values bounded between certain range, few examples of hash functions are MD5, SHA-256, etc. The other hash function is used for resolving collision for *h* hash function. The weight or aging function *f* is used to assign weight to the context term, it can be as simple as assigning constant weight to each context term or change depending upon the distance between the term and the context term.

For each term in text corpus, a new embedding vector of size n is created if the term was not processed previously otherwise the previously created embedding vector will be used. For each context term in the context window, weight is calculated using the weight function *f* and the hash is calculated using hash function *h* which decides the index in the embedding vector where the value of weight should be adjusted. At the end, each term would have an associated embedding vector and as shown in [11], the vectors of semantically similar words would be closer in vector space.

C. Enhanced Recommendation System

The proposed recommendation system for wiki resources leverages Hash2Vec technique for creating word embeddings for the descriptions of resources. The processing of all the data required for recommendation is done by recommendation media job. As shown in figure 4, a text file named “recommendation” is used to keep track of paths of the resources that have a description

file, this optimizes the process by avoiding a look up for all the resources that do not have descriptions. Based on this file, the recommendation job retrieves descriptions of resources from the file system.

A preprocessing of descriptions is done before starting Hash2Vec algorithm. This step split the description of a resource into individual lines based on new line character "\n". For each line, the individual terms are extracted using regular expression and any non-alphanumeric characters are removed from the terms. The stop terms like "a", "the", "is", etc. which occurs very frequently and do not have importance are removed. Also as shown in figure 8, there are certain terms like "Plot", "Genres", etc. used as labels are also removed from description. Finally, the characters in remaining terms are changed to lower case to avoid considering similar terms different due to character case difference.

After preprocessing if the remaining number of terms are less than the defined context window size of Hash2Vec algorithm then those terms are considered as metadata terms and are not considered during Hash2Vec phase. The intuition behind this is to avoid considering contextual information for list of terms that represent certain metadata details like name. Then Hash2Vec algorithm is performed on the remaining terms where context window size k was set to 5 and the size of embedding vector n was set to 200. The hash function h used was md5 and the weight function $f(d) = e^{-(d/\sigma)2}$ was used where d is the distance between current term and the context term and σ is the standard deviation of distances of all context terms.

ID	ITEM_TYPE	VECTOR
4267481446	2	vwGVpXOrAbo+53Ix746s+AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAvudyMe+OrPgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA+93Ix746s+L7ncjHvjqz4vudyMe+OrPg/...

Fig 10. *RECOMMENDATION_TERM_EMBEDDING*

The term embeddings are stored under RECOMMENDATION_TERM_EMBEDDING table in database as shown in figure 10. The ID column represents id unique to the term which is calculated using md5 hash function. The ITEM_TYPE column is used to represent whether the term embedding is for wiki resources description or group threads description. The embedding vector is normalized and stored under VECTOR column in a base64 encoding string.

The embedding vector for a resource is calculated by adding the embedding vectors of all the terms in the processed description of resource. In order to account the significance of metadata words separated during preprocessing step, a constant weight 1 is adjusted at the proper index in embedding vector of resource determined using md5 hash on a metadata term. This adjustment allowed to have embedding vectors of resources closer to each other if their metadata is similar for example two movie resources with same genre or star cast will be related closely to each other. Like term embedding vectors, resource embedding vectors are normalized and stored under RECOMMENDATION_ITEM_EMBEDDING table with similar schema as shown in figure 10.

	USER_ID	ITEM_ID	ITEM_TYPE	VIEW_DATE
	Filter	Filter	Filter	Filter
1	1	3510	2	1669963...
2	2	3510	2	1669963...
3	1	10	3	1669963...
4	2	10	3	1669963...
5	1	3510	2	1669963...
6	2	3510	2	1669963...
7	1	10	3	1669963...
8	2	10	3	1669963...
9	1	46313	6	1669963...
10	2	46313	6	1669963...

Fig 11. *ITEM_IMPRESSION*

To recommend resources to users, user embedding vectors are calculated based on the embedding vectors of resources which are already viewed by a user. This information is logged in ITEM_IMPRESSION table as shown in figure 11 whenever a user interacts with a resource. The

ITEM_TYPE column denotes the type of item user interacted with which can be a thread, group, wiki resource, etc. The ITEM_ID denotes the ID of item with which user interacted. For wiki resource item id is calculated by md5 hash of resource path, page id in which contains the resource and group id of the group containing the page.

The cosine similarity score is calculated between user embedding vector and embedding vectors of resource not viewed by the user. This score is stored under SCORE column of GROUP_RESOURCE_RECOMMENDATION table as shown in figure 12.

	USER_ID	GROUP_ID	PAGE_ID	RESOURCE_PATH	SCORE	TIMESTAMP	RESOURCE_ID
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	10	3510	/Users/admin/Desktop/CS298/yioop/work_directory/app/...	0.99444505505917	1669952640	9620
2	1	10	3510	/Users/admin/Desktop/CS298/yioop/work_directory/app/...	0.99705336595812	1669952640	21721
3	1	10	3510	/Users/admin/Desktop/CS298/yioop/work_directory/app/...	0.99685127799373	1669952640	40566

Fig 12. *GROUP_PAGE_RECOMMENDATION*

Data in other columns is used to create hyperlink for page where the resource can be viewed. Resources with top 3 scores are recommended to the user on the account home page as shown in figure 13.

Fig 13. *Recommendations in Yioop*

The cold start problem for users with no data in ITEM_IMPRESSION table is handled by recommending the popular resources to such users. The popular resources are determined by sum of recommendation scores of a resource for the users having impression data. The resources with top 3 scores are recommended to new users.

D. Threads and Groups Recommendation

In the enhanced recommendation system, threads recommendations are generated in a similar way of generating resource recommendations. Threads are stored in GROUP_ITEM table as shown in figure 14. Only the title and description of main threads are fetched and not for comment threads, a main thread has same PARENT_ID and ID. The recommendations are generated for threads not viewed by the user and belong to groups where the user is a member.

ID	PARENT_ID	GROUP_ID	USER_ID	URL	TITLE	DESCRIPTION	PUBDATE	EDIT_DATE	UPS	DOWNS	TYPE
Filter	Filter	Filter	Filter	Filt...	Filter	Filter	Filter	Filter	Filter	Filter	Filter
3542	3542	11	1		Crime Updates	Here you can find the updates regarding the crimes ...	1669103062	1669103062	0	0	0
3541	3540	10	1		-- This is the new thread	What about this.	1669091075	1669091075	0	0	0
3540	3540	10	1		This is the new thread	Please join in here	16690910...	16690910...	0	0	0

Fig 14. *Threads in GROUP_ITEM*

To generate recommendations for groups, group embedding vectors are calculated by adding the embedding vectors of threads in a group. The remaining steps of calculating user embeddings for groups and ultimately for recommendations are same as steps for calculating resource recommendations. The recommendation data for threads and groups is stored under GROUP_ITEM_RECOMMENDATION table as shown in figure 15. ITEM_TYPE column is used to differentiate between threads and groups.

	USER_ID	ITEM_ID	ITEM_TYPE	SCORE	TIMESTAMP
	Filter	Filter	Filter	Filter	Filter
1	1	3516	2	0.54530925642216	1669952640
2	1	3233	2	0.68199376790027	1669952640
3	1	3234	2	0.68202241865801	1669952640

Fig 15. *GROUP_ITEM_RECOMMENDATION*

VI. EXPERIMENTS

This section describes the details and results of experiments conducted to measure the effectiveness of proposed description update media job and enhanced recommendation system. The comparison details between the results of new and old recommendation system for groups and threads will be given.

A. Experiment for Description Update Media Job

The experiment was conducted on 2 different collections of wiki resources. In first collection there were 137 resources containing movies, TV shows and books released in year 2022. The second collection had 78 resources containing TV shows released back in year as far as the 1950s. Two description search sources were configured to fetch descriptions from IMDb [12] and Goodreads [13] sites. The description update media job was able to find descriptions for 130 resources in collection 1, 43 resources were missing values for some detail fields configured in the search source. For the other collection, descriptions for all the resources were fetched with 45 resources having correct descriptions. The details of remaining 33 resources were the descriptions of completely different TV shows with names closely matching the names of resource in the collection.

B. Experiment for Resources Recommendation

This experiment included 137 resources described in above section and 5 test users were created. Four users interacted with 15 resources each and the fifth user did not view any of the resources. A profile was assigned to a user based on the type of resources viewed by the user which can be used to measure the relevance of recommended resources for the user. Table 1 summarizes

the results of experiment. The system recommended more relevant recommendation to users that viewed resources of similar category. Fifth user was recommended resources that were frequently viewed by other four users.

User	Profile	No. of relevant recommendations	No. of irrelevant recommendations	Accuracy
1	Action	2	1	0.67
2	Science, Fiction, History and Romance	1	2	0.33
3	Drama	3	0	1
4	Thriller, Comedy, Family, Anime	1	2	0.33
Total		7	5	0.58

Table 1. *Results for resource recommendation experiment*

C. Experiment for Threads and Groups Recommendation

An experiment was conducted to compare the performance of proposed recommendation system with the existing recommendation system for threads and groups. Two instances of Yioop server were created on the same machine, first instance had the code for enhanced recommendation media job while the second instance had the code for existing recommendation media job. Same testing data was created on both the instances which included 5 groups and 10 threads within each group. The threads contained descriptions for 5 different categories – software development, gaming, political affairs, stock market and sports. Table 2 summarizes the threads of each category in a group and a profile is given to group based on threads in that group.

Group	No. of threads about software development	No. of threads about gaming	No. of threads about political affairs	No. of threads about stock market	No. of threads about sports	Profile
1	3	3	1	2	1	Mixed
2	1	5	0	0	4	Gaming / Sports
3	0	0	6	4	0	Political / Stock market
4	7	0	1	1	1	Software development
5	0	3	3	1	3	Mixed

Table 2. *Groups and threads experiment data*

On both Yioop instances, 5 test users were created and each user was a member of same group and viewed same 5 threads within group. The recommendation media job was started at the same time on both the instance. The job finished processing in 8 seconds on first instance with enhanced recommendation system code while the job on second instance took 24 seconds to complete processing. Table 3 shows the accuracy of recommendation results defined as ratio of relevant recommendations to total recommendations displayed for new and existing systems.

User	Thread recommendation accuracy for enhanced system	Thread recommendation accuracy for existing system	Group recommendation accuracy for enhanced system	Group recommendation accuracy for existing system
1	0.67	0.33	1	0.33
2	1	0.67	1	1
3	0.33	0	0.67	0
4	0.67	0.67	1	0.67
5	1	0.67	1	1
Average	0.73	0.47	0.93	0.6

Table 3. *Results for threads and groups experiment*

VII. CONCLUSION

For any wiki systems like Yioop it is important to ensure a total positive experience for its users. This project implemented new functionalities in Yioop like emoji picker tool, UI testing mechanism and advertisement credits redeeming into US dollars which enhanced experiences of user while using Yioop site. Hash2Vec word embedding technique was studied and used to enhance existing recommendation system in Yioop. A novel approach was used to extract descriptions for wiki resources by introducing description search sources and implementing a description update media job. Various experiments were conducted to compare the performance of proposed recommendation system with existing system and result showed higher accuracy for the recommendations generating by enhanced recommendation system. The processing time for synthesizing recommendation data was reduced by enhanced recommendation system.

There are many areas for future work building upon this project. The accuracy of description update media job can be significantly improved by considering best description across multiple description search sources. A weight function can be used to assign different weights to term embedding vectors while calculating embeddings vectors for threads, groups and resources which may adjust the importance of a term within the entire description increasing the contextual information retained by embedding vectors.

REFERENCES

[1]. B. Corinna, G. Bela, Langer and Stefan, "Research-paper recommender systems: a literature survey", *International Journal on Digital Libraries*. 17 (4): 305–338. doi:10.1007/s00799-015-0156-0

[2]. J. Ramos, et al., "Using TF-IDF to determine word relevance in document queries." *Proc. of the First Instructional Conf. on Machine Learning*, 2003, vol. 242, pp. 133–142

[3]. M. Anirudh, "An open source direct messaging and enhanced recommendation system for Yioop", San José State University, Dec 2021

[4]. *Full Emoji List, v15.0*. (n.d.). https://unicode.org/emoji/charts/full-emoji-list.html

[5]. *Selenium.* (n.d.). https://www.selenium.dev

[6]. *Stripe | A platform for processing online payments*. (n.d.). https://stripe.com

[7]. T. Gerth, "A Comparison of Word Embedding Techniques for Similarity Analysis.", Computer Science and Computer Engineering Undergraduate Honors Theses, 2021. Available: https://scholarworks.uark.edu/csceuht/85

[8]. L. Argerich, M. J. Cano and J. T. Zaffaroni, "Hash2Vec: Feature Hashing for Word Embeddings", 2016

[9]. Q. Shi, J. Petterson, G. Dror, J. Langford, A. Strehl, L. Smola, et al., "Hash kernels." *International Conference on Artificial Intelligence and Statistics*, 2009, 496–503

[10]. *IMDb | Ratings, Reviews, and Where to Watch the Best Movies & TV Shows*. (n.d.).

https://www.imdb.com

[11]. *Goodreads | Meet your next favorite book*. (n.d.). https://www.goodreads.com