TABLE OF CONTENT

1.ABSTRACT

The goal of the project is to develop a concept positioning system that recommends the next topics or skills to learn based on the topics you are already familiar with. Using data from Stack Overflow, where each topic (or concept) is represented by tags, the system will analyze the learning patterns of users and suggest new topics that are typically learned after similar ones. This approach aims to create a personalized learning path for users, guiding them through a linear progression of knowledge acquisition

2.INTRODUCTION

This project, the Concept Positioning System, aims to build a recommendation system that provides concepts in a linear order of difficulty to users. The system is divided into two parts: Knowledge Estimation and Bridging. Concepts are sourced from Stack Overflow tags, along with their descriptions and post scores. The underlying hypothesis is that if a user has answered a question, they possess knowledge of the relevant set of concepts.

In the Knowledge Estimation phase, the goal is to develop a recommendation system for implicit feedback that can suggest a set of concepts. For the Bridging phase, the aim is to order these recommended tags based on their prerequisite relationships.For Knowledge Estimation, we experimented with several recommendation algorithms, including Alternating Least Squares, Bayesian Personalized Ranking, and Neural Collaborative Filtering. The F1 score was our chosen metric for evaluating performance, and we compared the results of these different approaches.

In the Bridging phase, we utilized a prerequisite graph. We also experimented with random walks on this graph to enhance its utility as a recommendation system. The prerequisite graph is based on data from Stack Overflow.

AIVS Camp: This project requires a strong foundational knowledge base. To build this foundation, we participated in the AIVS camp, where we attended valuable scholar sessions and learned various essential topics.

Kaggle Challenge: Developing this recommendation system involves numerous trial-and-error methods. To foster the development of superior recommendation systems, we organized a Kaggle challenge to encourage multiple approaches and innovative solutions.

3.Data Sources for the Concept Positioning System

3.1 Techniques:

1. Stack Overflow

Stack Overflow data is recommended by my mentor due to its rich capture of user interactions. This data includes upvotes, downvotes, and scores, which provide valuable insights into user engagement and preferences. Stack Overflow offers both metadata and user data, making it highly suitable for a recommendation system. Additionally, it provides semantic meanings for items through detailed tag descriptions, enhancing the understanding of each topic.

2. Wikipedia

Similar to Stack Overflow, Wikipedia provides both metadata and user data. The metadata can be accessed through links, and user data is captured via clickstream analysis. However, Wikipedia data tends to be quite noisy, necessitating extensive preprocessing to extract meaningful information.

3. Books

Books are rich in the ordering of tags and related metadata. They offer valuable semantic information, but lack user data, making it challenging to understand user interactions and preferences. Moreover, books require significant preprocessing to extract and structure the necessary data.

4. MOOCs

Massive Open Online Courses (MOOCs) present another valuable data source, offering extensive metadata and user data. These online courses provide a wealth of information about user learning patterns and preferences. However, like Wikipedia and books, MOOCs data requires considerable processing to be useful for our purposes.

3.2 Inference

Among these data sources, Stack Overflow is preferred for its balance of metadata and user data, along with its relatively structured format and semantic richness. While other sources like Wikipedia, books, and MOOCs offer valuable information, they require extensive preprocessing to mitigate noise and extract meaningful insights. Stack Overflow’s detailed user interaction data and semantic tag descriptions make it a superior choice for developing an effective concept positioning system.

4.Data Retrieval

4.1Stack Overflow:

1. Stack Exchange Data Explorer (SEDE):

SEDE allows you to easily query the data you need. You can select the domain, such as Mathematics or Ubuntu, and explicitly write the SQL query to retrieve the desired data. You can then download the queried rows as a CSV file. However, SEDE does not allow you to run queries locally and can only be used manually. [Link](https://data.stackexchange.com/)

2. Stack Exchange API:

The Stack Exchange API allows you to query some aspects of Stack Overflow using REST calls. You can use the API in Python to retrieve data programmatically.

3. Public Release Data Dump:

You can download a public release of the Stack Overflow data dump and use historical data. The dataset is approximately 96 GB and can be downloaded as a ZIP file from the Internet Archive. The data is in XML format and needs to be converted to CSV. [Link](https://archive.org/details/stackexchange)

4. Google BigQuery:

The Stack Exchange public dataset is also available on Google BigQuery. You can query the public data maintained by Google, which is free up to 1 TB, but there is a charge for querying beyond that. This is useful for obtaining large datasets, such as those with more than 50,000 entries. [Link](https://cloud.google.com/bigquery/public-data)

5. Kaggle:

While Kaggle is an option, it is not preferred because it contains smaller and less frequently updated datasets. Using Kaggle would narrow the application's scope.

For our project, we preferred downloading the data dump from the Internet Archive, which includes multiple tables like Posts.csv, Users.csv, and Tags.csv. These files are in XML format and need to be converted to CSV.

4.2 Wikipedia:

1. Data Dump:

We downloaded an SQL file of 250 GB from the Wikipedia data dump. This file includes articles, links, and clickstream data for multiple domains. We specifically obtained data for computer science and are working on a prerequisite graph.

2. Web Crawling:

Another method for data retrieval is web crawling. We provided a list of article names, and the crawler found the links between these articles, identifying the relationships among them.

5.Big Data Handling

1. **XML to CSV Conversion:** This conversion was handled by my teammate, who successfully retrieved users.csv, posts.csv, and tags.csv.
2. **Cleaning Datasets:** The CSV files were further cleaned to remove unwanted columns and rows. This was done in chunks using parallel processing with Dask, a very helpful Python tool. My teammate cleaned posts.csv and tags.csv, while I handled the cleaning of users.csv.
3. **Further Breakdown:** The posts table was further broken down to include only the relevant rows for the set of tags we considered. Based on postType and parentId, we divided the posts into questions and answers. The questions.csv contained the tags, while the answers.csv contained the parentId for which it was an answer. Our hypothesis was that if a person answered a question, they would know all the tags related to that question. Therefore, we mapped answers to questions and created a new table with columns ID, OwnerUserID, Tags, CreationDate, and Score.
4. **Further Purification:** This table was further purified to include only the tags of interest, ultimately selecting 231 tags.

```
tags_list=["boolean","boolean-operations","boolean-logic","boolean-expression","boolean-algebra","boolean-polynomials","proof","induction","proof-of correctness","probability","probability-theory","abstract-data-type","arrays","static-array","array-comparison","array-reverse","array-address","array-pointer","dynamic-arrays","dynamic-allocation","array-merge","sparse-array", "arrayofarrays","multidimensional-array","sparse-matrix","matrix-multiplication","array-multisort", "variable-length-array","linked-list", "list", "singly-linked-list","doubly-linked-list","string","string-operations", "bitstring", "substring", "string-concatenation","string-substitution","string-comparison","string-matching","algorithm","array-algorithms","quicksort","mergesort","radix-sort", "linear-search", "bucket-sort", "ternary-search", "counting-sort","lexicographic-ordering","combinatorics","binary-search", "bubble-sort", "selection-sort","sorting", "insertion-sort", "stable-sort","data-structures","stack","queue","queueing","enqueue","heap","heapalloc","heap-memory","heap-table","heapsort","binary-heap","fibonacci-heap", "priority-queue", "circular-queue", "fifo", "stack-memory", "binomial-heap", "lifo", "stack-allocation","min-heap","max-heap","minmax-heap","heap-size","heaps-algorithm","tree","treenode","treepath","tree-search","tree-balancing","tree-rotation", "tree-traversal","breadth-first-search", "depth-first-search", "best-first-search", "treesort", "binary-tree", "ternary-tree","multiway-tree", "binary-search-tree", "balanced-binary-search-tree", "ternary-search-tree", "avl-tree", "b-tree","b-plus-tree", "red-black-tree", "red-black-tree-insertion", "splay-tree", "hashtree", "n-ary-tree", "huffman-tree","huffman-code","string-algorithm", "longest-substring", "lcs", "greedy", "optimization", "mathematical-optimization","linear-
```

programming","knuth-morris-pratt", "rabin-karp","graph", "graph-theory", "strongly-connected-graph", "adjacency-list", "adjacency-matrix", "directed-graph","undirected-graph", "digraphs", "subgraph", "minimum-spanning-tree", "minimum-spanning-forest","directed-acyclic-graphs", "graph-traversal", "a-star", "Dijkstra", "kruskals-algorithm", "prims-algorithm","hungarian-algorithm", "simplex", "simplex-algorithm", "strassen", "clique", "clique-problem","graph-algorithm", "tarjans-algorithm", "shortest-path", "planar-graph", "floyd-warshall","traveling-salesman", "path-finding", "bipartite", "hamiltonian-cycle", "topological-sort", "graph-coloring","spanning-tree", "bellman-ford", "kosaraju-algorithm", "euler-path", "graph-query", "dependency-graph","vertex-cover", "hamiltonian-path", "cyclic-graph","hash", "hashmap", "hash-function", "hashset","double-hashing","hashtable", "hashcode", "hash-collision","consistent-hashing","dynamic-programming", "knapsack-problem", "code-complexity", "big-o", "tail-recursion","partitioning","time-complexity","space-complexity","memory-management", "network-flow", "branch-and-bound", "max-flow", "master-theorem", "recursive-backtracking","fibonacci", "towers-of-hanoi", "recursion", "divide-and-conquer", "asymptotic-complexity","greatest-common-divisor", "partition-problem", "ford-fulkerson", "edmonds-karp", "set", "set-cover","amortized-analysis", "set-intersection", "set-union", "set-theory", "set-difference", "set-operations","set-cover","disjoint-union","disjoint-sets","union-find","complexity-theory", "recursive-datastructures", "p-np", "np", "np-hard", "np-complete", "sat","satisfiability", "2-satisfiability", "stable-marriage", "heuristics", "artificial-intelligence", "automata","automata-theory", "finite-automata","non-deterministic", "subset-sum","minimax", "turing-machines","turing-complete", "halting-problem", "backtracking", "formal-languages", "stability", "computability","probing", "linear-probing", "quadratic-probing"]

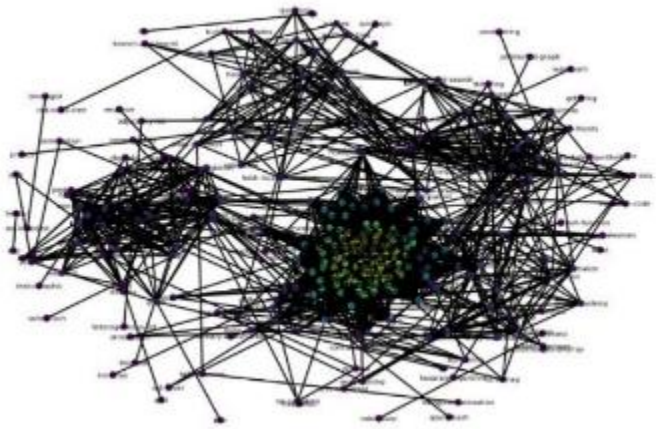
Fig 1: Above are the 231 Tags of interest in our research

Prerequisite Graph Construction and Experiments

To enhance our recommendation system by understanding the hierarchy of concepts, we aimed to create a prerequisite graph using tag correlations. This graph would help us recommend tags in increasing order of difficulty. We utilized NetworkX for visualization and graph creation.

Experiment 1: Used the BERT base model to generate embeddings for each tag and calculated cosine similarity. This approach yielded fewer effective results.
Experiment 2: Trained the BERT model on Wikipedia pages related to given tags using unsupervised fine-tuning, which provided better results. This phase enhanced our understanding of Plotly and network Python libraries, as well as different BERT models and fine-tuning techniques, including BERT for Sequence Classification and BERT for Masked Language Modeling.

Experiment 3: Explored unsupervised training on large corpora from both Wikipedia and books. This approach was less effective. The graph had a high level noise.



6.Wikipedia-Based Prerequisite Graph

Utilizing Wikipedia, we aimed to obtain the entire hierarchy required for the project, despite the considerable noise present in the data. While Stack Overflow data represented user behavior comprehensively, Wikipedia data included both user behavior (clickstreams) and page links, offering a richer context for creating the graph.

- **Process:** We created a graph that, although effective, still contained noise. Previous work on this project addressed 347 tags, and we refined this set of tags to improve the graph's quality. Ultimately, we selected a more focused set of tags, resulting in a better graph with persistent, albeit reduced, noise.
- **Significance:** The prerequisite graph is crucial as it assists the recommender system in suggesting concepts in order of difficulty, thereby enhancing the learning experience for users. This phase provided valuable insights into handling different types of data from Wikipedia and improved our methodology for creating and refining prerequisite graphs.

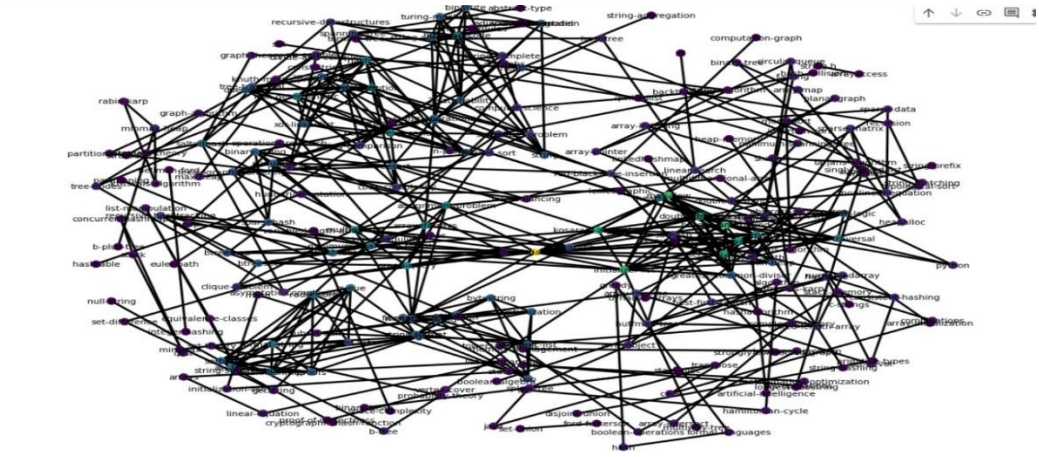


Fig 2: Prerequiste graph using wikipedia article links

Fig 3: Concept of Flipping a image

7.User-Tag Matrix:

We constructed the User-Tag matrix using Stack Overflow data, which provides rich insights into user behavior. For a given list of tags and a set of users who answered questions tagged with these tags, we hypothesize that if User 1 knows concepts 1 and 2 and wants to learn concept 5, and another user knows all 5 concepts, then User 1 can follow the learning path of the other user. The sequence may vary based on difficulty and prerequisite order.

Two matrices were created:

- A binary (implicit feedback) matrix.
- A score matrix, where the scores are the weighted sum of the scores for that user who answered the tag multiple times.

OwnerUse	recursion	hashtable	hashmap	boolean-p	digraphs	probing	stack-alloc	heap-size	cyclic-grap	linear-proc	string-algo	min
1	0	1	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	
13	1	1	1	0	0	0	0	0	0	0	0	
17	1	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	
22	0	1	1	0	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	0	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	0	
29	1	0	0	0	0	0	0	0	0	0	0	
30	0	0	0	0	0	0	0	0	0	0	0	
33	0	0	0	0	0	0	0	0	0	0	0	
34	0	0	0	0	0	0	0	0	0	0	0	

Fig 4: User-Tag-Binary Matrix

OwnerUse	boolean	boolean-o	boolean-lc	boolean-e	boolean-a	boolean-p	proof	induction	proof-of-c	probability	probability	abstract-d	arrays	sta
1	0	0	0	0	0	0	0	0	0	0	0	0	497	
3	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	3	
5	0	0	0	0	0	0	0	0	0	0	0	0	1	
13	116	2	0	3	0	0	0	0	0	0	0	0	496	
17	18	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	2	
23	0	0	0	0	0	0	0	0	0	0	0	0	19	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	0	0	10	
29	13	0	0	0	0	0	0	0	0	0	0	0	15	
30	0	0	0	0	0	0	0	0	0	0	0	0	0	

Fig 5: User-Tag-With Scores Matrix

8.Kaggle Challenge Preparation

To develop a more accurate recommendation system, Dr. Sudarshan Iyengar suggested creating a Kaggle challenge. This challenge aims to encourage the development of better recommendation systems and ultimately build a superior product. The concept of this challenge is similar to the Netflix challenge. The dataset will be divided into training and testing sets, with the test set being hidden. Participants will be required to train a model using the training data, and their models will be evaluated based on specific metrics.

Due to the sparsity and large size of the user-tag matrix, it cannot be provided directly. Additionally, since the matrix is derived from Stack Overflow data, there is a risk that it could be reverse-engineered, which poses privacy concerns. Therefore, my teammates and I decided to encrypt or morph the UserID, and one of my teammates suggested adding synthetic data to further protect the information.

The user-tag matrix was then converted into tuples and stored in a format similar to the MovieLens dataset. While converting, we created two CSV files. The first CSV file contains three columns: UserID (obtained from the OwnerUserID), itemID (obtained by getting the TagIndex of the corresponding Tag column in the user-tag matrix), and a Score or Rating column. This format ensures that the data is more manageable and secures user privacy while still being useful for the challenge participants.

OwnerUse	Tag	Score	TagIndex
1	algorithm	37	40
1	arrays	497	12
1	hash	6	160
1	hashtable	1	165
1	optimizati	7	112
1	string	25	32
3	optimizati	8	112
4	algorithm	8	40
4	arrays	3	12
4	heuristics	6	213
4	list	2	29
4	memory-m	4	177
4	optimizati	5	112
4	recursion	3	185

Fig 6: Kaggle Dataset format

Further Data Observations

If we give OwnerUserID and a few tags that they know, people can easily reverse engineer it. That's why we used TagIndex, which is in the order that we have, and also encrypting the OwnerUserID would be better.

We didn’t consider time because it didn’t represent any learning sequences. For example, a person answered a problem related to optimization in 2008 and then in 2016 answered a problem addressed by tags such as arrays and strings. Then again, they answered another question related to hash maps.

Optimization -> array -> string -> hash maps

This doesn’t make any learning sequence. Another observed problem was in the learning sequences where the tags get repeated. The same user again answered a post related to arrays, thus:

Optimization -> array -> string -> hash maps -> array

This relation can’t be pruned. Even in some cases, their co-occurrence with another tag may give some learning path. For the user-tag matrix to tuple conversion, we considered only the entries for which it is nonzero (>0). Another problem that was found was there were negative scores for a few answers. Those scores should not be considered, and thus while converting, we considered only the scores (>0).

We also encountered another problem: there were a few entries where the user interacted, but the scores assigned were 0. In such cases, even though they interacted, they were automatically considered as not having interacted. This would cause us to lose valuable information.

We also plotted a 'number of users' vs 'tag' plot and found that it was an exponential distribution. We had a total of 499,546 rows in the user-tag matrix after cleaning and filtering. Among these, 279k rows were users with one tag, which doesn’t represent any learning sequence at all. Thus, we removed them.

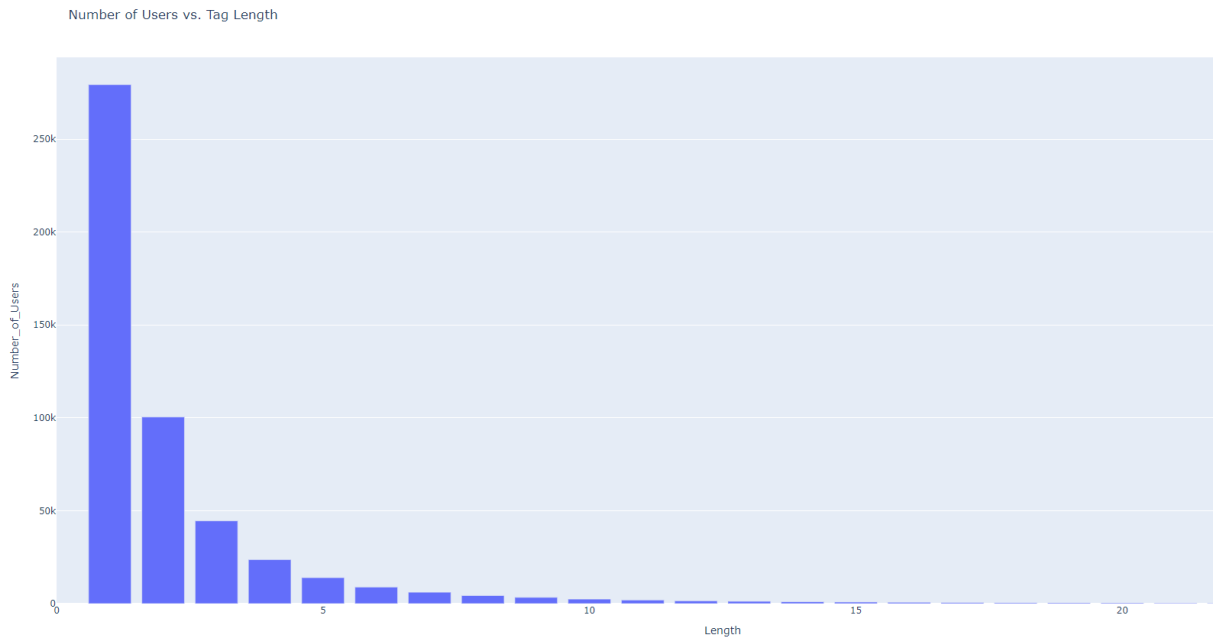


Fig 7: Exponential distribution – No_of Users vs Tags

9.Stack OverFlow Based Prerequiste Graph

In order to perform bridging after knowledge estimation, we require the prerequisite graph. Due to differing naming conventions between Stack Overflow and Wikipedia, their existing graphs cannot be used directly. I plotted the user tag co-occurrence in such a way that:

$$\text{Co-occurrence}(t1,t2)=\sum(wt1,2)$$

To determine the direction of the relationship, instead of changing the weight magnitude, I verified the direction using the logic that

If t1 -> t2 > t2 -> t1 then t1 -> t2

Else the other case

After obtaining the graph, I extracted the maximal spanning tree using the NetworkX library in Python. I then visualized this tree using the interactive plotting tool, Plotly.

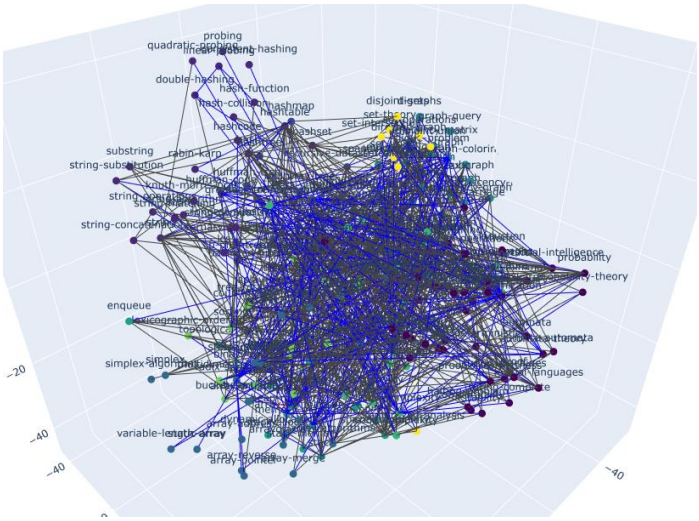


Fig 8: Cooccurrence graph over the communities

Here communities are represented by different colors . My teammate extracted the communities using Nearest Neighbour. Also i did few manual changes to the maximal spanning tree to make it look more accurate. I have provided both the graphs below .

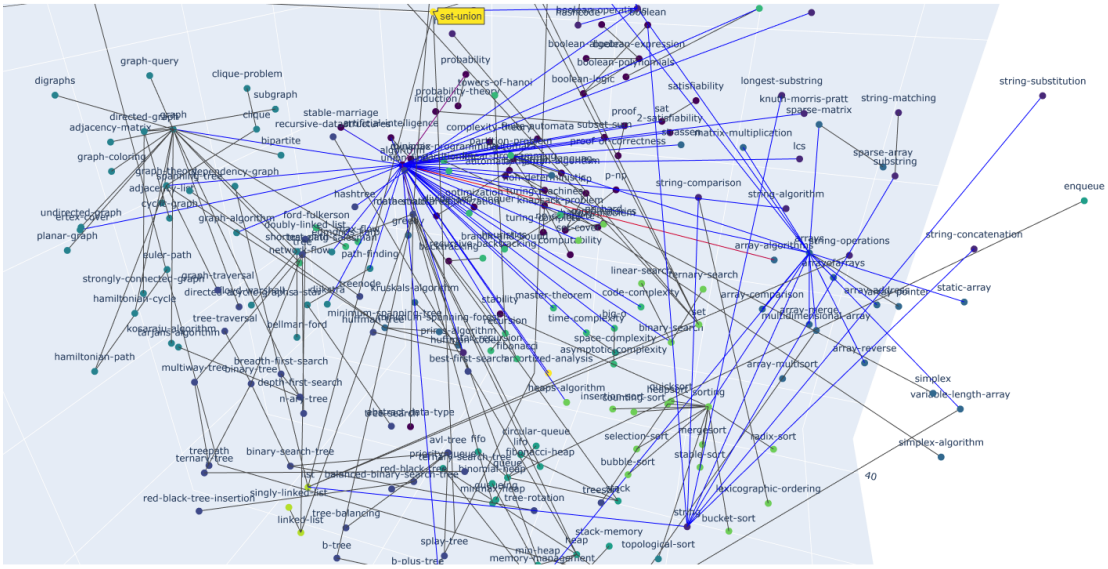


Fig 9: maximal spanning tree

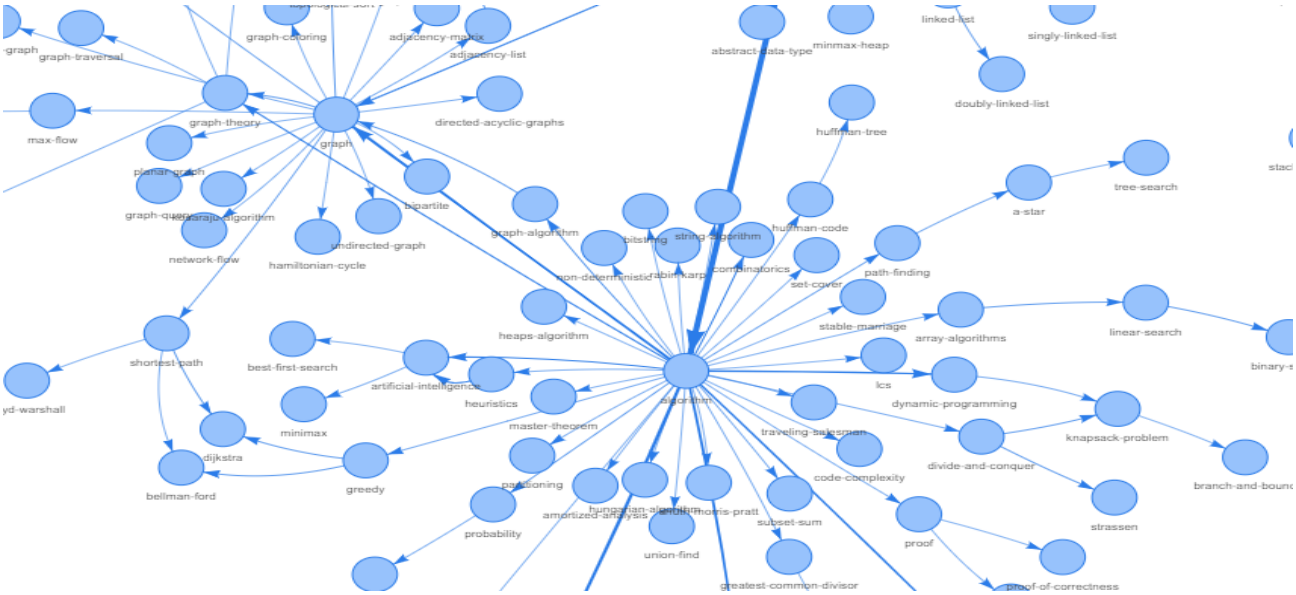


Fig 10: the acyclic graph after changing few links and plotted using PYVIZ

These Prerequiste graph is used for Bridging and i also tried random walk on this graph to check the results and in the next section i have given the explanation.

10.1 Random Walk on the graph:

I tried random walk on the prerequiste graph. I did three experiments

$$PR=\alpha \cdot P \cdot PR+(1-\alpha) \cdot v$$

PR : Page Rank

A : Damping factor

P: Probability transition matrix

Experiment 1:

I tried random walk using the inbuilt function given by networkx . I tried it on three different graphs item interation graph , maximal spanning tree graph and graph with few manual changes . The results obtained are given below

```
CASE: 1
Graph loaded with 231 nodes and 4487 edges.
PageRank computed successfully.
Top recommendations based on PageRank:
Node: kosaraju-algorithm, PageRank: 0.01521582209846155
Node: recursive-backtracking, PageRank: 0.013724212428604853
Node: longest-substring, PageRank: 0.01154506229542819
Node: big-o, PageRank: 0.01144588473536216
Node: strongly-connected-graph, PageRank: 0.011194739477045634
CASE: 2
Graph loaded with 231 nodes and 230 edges.
PageRank computed successfully.
Top recommendations based on PageRank:
Node: boolean-operations, PageRank: 0.00895717289288754
Node: binomial-heap, PageRank: 0.008470445078241379
Node: red-black-tree-insertion, PageRank: 0.008329286889406545
Node: kruskals-algorithm, PageRank: 0.008221430847573393
Node: np-hard, PageRank: 0.008111633533455767
CASE: 3
Graph loaded with 231 nodes and 242 edges.
PageRank computed successfully.
Top recommendations based on PageRank:
Node: binomial-heap, PageRank: 0.009143744857206476
Node: branch-and-bound, PageRank: 0.009095629558014056
Node: ternary-search, PageRank: 0.008475820536925117
Node: boolean-operations, PageRank: 0.00844719883781316
Node: 2-satisfiability, PageRank: 0.008166324115024921
```

Fig 11: Random walk using inbuilt function

Experiment 2:

I tried random walk but i handled the sinkhole case differently The result is given below

- **Initialization**: Start with the initial state vector and an empty stack.
- **Random Walk with Backtracking**:
- Perform the random walk with a probability determined by the damping factor.
- When reaching a node with no outgoing edges, backtrack using the stack and choose the next node based on transition probabilities.

- **Maintain Stack for Backtracking**: Keep track of visited nodes in a stack to enable backtracking.

```
CASE: 1
Graph loaded with 231 nodes and 4487 edges.
Top recommendations based on PageRank:
Node: kosaraju-algorithm, PageRank: 0.01112843482518261
Node: recursive-backtracking, PageRank: 0.009604723795514294
Node: longest-substring, PageRank: 0.007981070941146152
Node: strongly-connected-graph, PageRank: 0.007674607877229069
Node: graph-coloring, PageRank: 0.007394048359299946
CASE: 2
Graph loaded with 231 nodes and 230 edges.
Top recommendations based on PageRank:
Node: binomial-heap, PageRank: 0.003683236580031438
Node: edmonds-karp, PageRank: 0.003613977272497515
Node: branch-and-bound, PageRank: 0.003595295810394316
Node: tree-search, PageRank: 0.003595295810394316
Node: ternary-search-tree, PageRank: 0.0032200624999770243
CASE: 3
Graph loaded with 231 nodes and 242 edges.
Top recommendations based on PageRank:
Node: quadratic-probing, PageRank: 0.004132511639835858
Node: ternary-search, PageRank: 0.00405166262672061
Node: 2-satisfiability, PageRank: 0.003929527387732551
Node: branch-and-bound, PageRank: 0.003909602309718276
Node: binomial-heap, PageRank: 0.003810004032019413
```

Fig 12: Random walk with modification .

The problem in our case is frequent sinkholes. When the surfer reaches a random end node, instead of randomly jumping to another node, I backtrack to the previous node and randomly choose another immediate node.

11.Recommendar systems

Our case based on data we concluded that we are performing implicit feedback recommendar systems. We began from very basic level and thought of slowly increasing to high level model

11.1.**Matrix Factorization**

This approach is based on UV decomposition. We did not use SVD because our matrix is sparse, and SVD is prone to overfitting in such cases. Hence, we opted for UV decomposition. Here, we randomly initialized two matrices, U and V, with dimensions $U \in R^{n \times k}$ and $V \in R^{k \times m}$, whose product gives the matrix $A' \in R^{n \times m}$, where A' is a dense approximation of the original sparse matrix $A \in R^{n \times m}$. We first randomly initialize the entries of the U and V matrices and use gradient

descent to optimize or minimize the error. In my experiment, I did not calculate any metrics like precision, recall, MAP, NDCG, or F1 score. For gradient descent i used Adam optimizer .

Objective function:

- A is an m×n matrix.
- U is an m×k matrix (representing the feature vectors for the rows of A).
- V is a k×n matrix (representing the feature vectors for the columns of A).
- $A'=UV$ is the approximated matrix.

The cost function JJJ for the UV decomposition can be defined as:

$$J(U,V) = \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - A'_{ij})^2 + \lambda \left(\sum_{i=1}^{m} \sum_{k=1}^{K} U_{ik}^2 + \sum_{j=1}^{n} \sum_{k=1}^{K} V_{kj}^2 \right)$$

Result:

```
linked-list: 0.7242
c-strings: 0.7239
singly-linked-list: 0.7229
python: 0.7228
linear-programming: 0.7227
list: 0.7222
insertion-sort: 0.7218
recursion: 0.7218
selection-sort: 0.7217
data-structures: 0.7215
```

Fig 13: Recommendation given by UV when user knows ['oop','sorting','math','big-o','arrays']

Train/Test Split:

To measure the performance of the model, the dataset was divided into two parts: train and test. The logic for this split is as follows:

- Users who have answered fewer than 1 tag were removed.
- Users who have answered between 2 and 8 tags were put in the train set.
- For users who answered more than 8 tags, 5 random tags were taken for the test set, and the remaining tags were put in the train set.

11.2 Metrics

We considered four metrics for evaluating our system:

- **MAP (Mean Average Precision):** This metric evaluates the ranking quality by averaging the precision scores at the ranks where relevant items are found. In an implicit recommendation system, MAP measures how well the system ranks the relevant concepts for a user.
- **NDCG (Normalized Discounted Cumulative Gain):** This metric assesses the ranking quality by considering the positions of the relevant items in the recommendation list, with higher-ranked items given more weight. In an implicit recommendation system, NDCG helps evaluate the usefulness of the ranking by penalizing relevant concepts that appear lower in the list.
- **Precision:** This metric measures the proportion of recommended concepts that are relevant to the user. In the context of an implicit recommendation system, it indicates the accuracy of the recommendations.

$$\text{Precision} = \frac{|\{\text{relevant items}\} \cap \{\text{recommended items}\}|}{|\{\text{recommended items}\}|}$$

- **Recall:** This metric measures the proportion of relevant concepts that are successfully recommended to the user. In an implicit recommendation system, it reflects the system's ability to capture all relevant concepts for a user.

$$\text{Recall} = \frac{|\{\text{relevant items}\} \cap \{\text{recommended items}\}|}{|\{\text{relevant items}\}|}$$

In the Knowledge Estimation phase, ranking the concepts is not necessary. Therefore, the first two metrics, MAP and NDCG, are used to assess the quality of the ranking. Instead, we focused on Precision, Recall, and **F1-Score** as our final metrics.

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

11.3 Bayesian Personalized ranking

After the Implementation of UV decomposition I tried BPR on the dataset . the code for this model was available at github which can be accessed with recommenders library . Optimizes for pairwise ranking, focusing on the relative ranking of items rather than their absolute ratings. It aims to maximize the difference in predicted scores between a pair of items, ensuring that a user’s preferred items are ranked higher.

Objective function:

$$J(\Theta) = \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) - \lambda_\Theta ||\Theta||^2$$

Where DS={(u,i,j)|(u,i)∈S and j∈I\{i}} ,x^ui is the predicted score for user u and item I and x^uj is the predicted score for user u and item j.

```
Random UserID: 364696
Known Tags for User: ['a-star', 'algorithm', 'arrays', 'big-o', 'binary-search', 'boolean', 'boolean-expression',
Top 10 Predicted Tags for User:
1: matrix-multiplication (Prediction: 1.3800346851348877)
2: stack (Prediction: 1.2922627925872803)
3: insertion-sort (Prediction: 1.215512990951538)
4: binary-tree (Prediction: 1.1537493467330933)
5: greatest-common-divisor (Prediction: 0.8571100234985352)
6: tree (Prediction: 0.596876323223114)
7: radix-sort (Prediction: 0.011361122131347656)
8: huffman-code (Prediction: -0.2430262565612793)
9: hashmap (Prediction: -0.4059211015701294)
10: tail-recursion (Prediction: -0.5129590630531311)

User Test Set:
       userID                   Tag  rating  itemID
27800  364696                 stack       8      58
27801  364696  variable-length-array       2      27
27802  364696          network-flow       0     178
27803  364696               hashmap       9     161
27804  364696   string-concatenation     175      36
['matrix-multiplication', 'stack', 'insertion-sort', 'binary-tree', 'greatest-common-divisor', 'tree', 'radix-sort
```

Fig 14: Predictions using BPR

Results based on metrics:

```
MAP:    0.435804
NDCG:   0.575710
Precision@K:    0.505710
Recall@K:       0.505710
```

Fig 15: Metrics Result for BPR on only Real data

12.Conclusion:

We had an comparative study of all the models that we tried like Alternating Least Square ,Bayesian Personalized ranking , Neural Collaborative Filtering . In that my model performance (F1 -score) was about 0.50 . after getting results with prerequisite graph bridging is done to arrange in an linear order of learning .further these models can be improved if semantic informations are also included. Alternating least square resulted with an accuracy of 0.29.

Dataset	ALS (F1 Score @k)	BPR (F1 Score @k)
Real dataset	0.25	0.50
Augmented dataset	0.15	0.46

Fig:16 : Comparison of F1 Score @k values for ALS and BPR on Real and Augmented datasets

13.Future Scope:

- Identifying an alternative to Stack Overflow.
- Prerequisites for graph-based recommender systems.
- The LightGCN model can also be very helpful for implicit feedback recommender systems.
- Implementation of reinforcement learning-based recommender systems.

14 . Bibilography

[1] https://arxiv.org/pdf/1205.2618 : BPR: Bayesian Personalized Ranking from Implicit Feedback

[2] https://math.dartmouth.edu/~doyle/docs/cat/cat.pdf : Random Walks on Weighted Graphs

[3] K. Xiao, Y. Fu, J. Zhang and W. Tianji, "A Hybrid Approach for Discovering Concept Prerequisite Relations in Wikipedia," *2022 9th International Conference on Behavioural and Social Computing (BESC)*, Matsuyama, Japan, 2022

[4] https://cs229.stanford.edu/proj2013/SchusterZhuCheng-PredictingTagsforStackOverflowQuestions.pdf : Predicting Tags for StackOverflow Questions

[5] https://arxiv.org/pdf/2302.02579 : Recommender Systems: A Primer

[6] https://dl.acm.org/doi/pdf/10.1145/2939672.2939673 : Collaborative Knowledge Base Embedding for Recommender Systems

[7] https://dl.acm.org/doi/pdf/10.1145/3289600.3291016 : RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation

[8] https://sudarshansudarshan.github.io/aicamp/ : AI Vicharana Shala

[9] https://everdark.github.io/k9/notebooks/ml/matrix_factorization/matrix_factorization.nb.html:Matrix factorization for Recommender systems

[10] Hui Chen, John Coogle, Kostadin Damevski,Modeling stack overflow tags and topics as a hierarchy of concepts,Journal of Systems and Software,Volume 156,2019,Pages 283-299,ISSN 0164-1212