RepoLens - Github Repository Recommendation

Adithya Sunil IMT2021068
Nimish Khandeparkar IMT2021077
Mayank Sharma IMT2021086
Ayush Singh IMT2021096

I. Abstract

Open source developers, pivotal in driving technological innovation, often face challenges in locating suitable repositories to contribute their expertise. To address this, our project, RepoLens, develops a robust recommendation system that assists developers in discovering repositories that align closely with their interests and skills. Utilizing GitHub's API, our system gathers extensive data on user activities and repository characteristics from the 100 most popular repositories between 2008 and 2023. By employing advanced Natural Language Processing (NLP) techniques such as TF-IDF, Doc2Vec, and BERT, alongside similarity metrics like cosine and Jaccard similarities, RepoLens meticulously crafts personalized repository recommendations. This system not only enhances developer engagement by connecting them with compatible projects but also augments the overall efficiency and output of the open source community. Our evaluation shows significant improvement in matching developers with relevant repositories, leveraging both content-based metrics and user-specific data to mitigate common issues like the cold start problem in recommendation systems.

II. Data-set Creation

To construct a robust dataset for our GitHub repository recommendation system, we utilized the GitHub API, which provides extensive access to both user and repository data. This access is secured through an authentication token, ensuring that our requests are both safe and compliant with GitHub's usage policies.

Our dataset curation focused on selecting the 100 most popular repositories annually, ranked by stars, from 2008 to 2023. This strategy resulted in a dataset comprising approximately 1,600 repositories. These repositories were chosen for their broad appeal and relevance across various sectors within the technology community. This selection not only offers a snapshot of evolving trends in repository popularity over a significant period but also covers a diverse range of programming languages and development practices.

To efficiently handle the large volume of API requests needed to compile this data, we employed an asynchronous fetching mechanism. This approach allows for multiple data requests to be processed concurrently, significantly speeding up the data collection process without overloading the server. By using asynchronous methods, we were able to reduce the response time for each request, enabling a more streamlined and efficient gathering of data.

The following data was obtained for each repository:

- Name and Owner
- Description
- Stars, Watchers, and Forks
- Topics
- Programming Languages
- README content

III. Pre-processing

In the development of our GitHub repository recommendation system, the selection and preparation of features from the dataset are crucial for effectively training the model to provide personalized recommendations. Although metrics like stars, watchers, and forks are straightforward indicators of a repository's popularity, they do not necessarily reflect the nuanced interests or expertise of an individual developer. Consequently, while these features might suggest general appeal, they do not lend themselves well to personalization. Thus, for our model, which aims to match repositories to users based on individual alignment rather than broad popularity, these features were excluded from the dataset.

Text-based features such as the repository's description and its README file are rich sources of semantic information that can significantly aid in understanding the content and purpose of a repository. These features are particularly useful for identifying thematic and contextual similarities between repositories. To extract meaningful patterns from these textual features, Natural Language Processing (NLP) algorithms are employed. For instance, methods like TF-IDF (Term Frequency-Inverse Document Frequency) or more advanced embeddings from models like Doc2Vec or BERT could be used to convert raw text into a structured, high-dimensional space where textual similarities can be quantified.

However, the README files were cluttered with URLs and HTML tags which would not work well with the NLP models, so the following pre-processing steps were performed.

- Conversion to lowercase
- Removal of text enclosed in <> (HTML Tags)
- Removal of text beginning with http (URLs)
- Replacing any punctuation or special characters such as newline with spaces
- Removing numbers and hexadecimals

Topics associated with a repository provide a categorical sense of the areas covered by the project, such as 'machine learning', 'web development', or 'data analysis'. These can be effectively represented as one-hot encoded vectors, where each vector has a dimension for each possible topic and a binary

indicator (0 or 1) signifying the absence or presence of the respective topic in a repository.

The usage of programming languages in a repository can also inform the recommendation system about the technical nature of the projects. Programming languages in repositories are often quantified by the fraction of the repository's codebase written in each language. These fractions can be represented as float vectors, with each entry representing the proportion of the codebase written in a specific programming language, normalized between 0 and 1. This quantification allows for nuanced comparisons between the programming profiles of different repositories.

IV. Models and Similarity Measurement

A. Similarity Measurement

The core idea behind our recommendation engine is to measure the similarity between two repositories, which is accomplished by constructing feature vectors from the Description and README sections using Natural Language Processing (NLP) models. Topics and programming languages associated with repositories are treated as one-hot encoded binary and float vectors, respectively. Jaccard similarity (intersection / union) is used to calculate the similarity between two binary vectors in the topic representation. For the other vectors, a basic cosine similarity (normalized dot product) is used. The final "recommendation score" is a weighted sum of these individual similarity scores.

B. Keyword Similarity: Using TF-IDF

The Term Frequency-Inverse Document Frequency (TF-IDF) method is particularly adept at the task of identifying keywords. TF-IDF quantifies the importance of a word within a repository's textual content relative to a collection of repositories. This is beneficial for our project as it helps to highlight repositories that use specific technical terms or technologies more frequently than others. By transforming the 'Description' and 'README' texts of repositories into TF-IDF vectors, we can efficiently calculate the similarity between repositories based on the presence and importance of common keywords. This approach ensures that repositories are recommended to users based on shared technical terms and tools, which are often crucial indicators of a developer's interest and expertise areas.

C. Semantic Similarity: Leveraging Doc2Vec and BERT

Beyond mere keywords, understanding the context and semantic meaning of texts within repositories is vital for a more nuanced recommendation. Doc2Vec and BERT are powerful tools for capturing the semantic structures of text. Doc2Vec extends the idea of word embeddings to entire documents, enabling our system to consider the overall context of repository README files, rather than just isolated words. BERT, which utilizes a deep learning model based on the Transformer architecture, takes this further by understanding the nuances and relations of words in sentences, making it highly effective for semantic analysis.

For our project, employing Doc2Vec and BERT allows the recommendation system to not only match repositories based on shared topics but also on the subtler contextual similarities in their documentation. These models were only used on the README and not the Description, as most descriptions were single sentences without much context. Using combinations of keyword similarities and semantic similarities proved to give the best results as seen in the upcoming sections.

V. Test Data Generation and Evaluation

A. Test Data

To validate our recommendation system, we handpicked 60 renowned repositories and manually grouped them into similar categories using human intuition. For example the "django" and "angular" repositories were grouped together as they both were Web Frameworks. The idea behind using such a method of testing is that most people would prefer getting recommendations from repositories belonging to the same group.

B. Evaluation

The effectiveness of the GitHub repository recommendation system was assessed using the precision metric, a common measure in the field of information retrieval. Precision in this context is defined as the proportion of relevant recommendations within the top five suggestions provided by the system, relative to a specific query repository.

To conduct the evaluation, each repository in our test dataset was processed as a query. For each query, the system calculated similarity scores with all other repositories in the dataset, and the top five repositories were selected as recommendations. Precision for each query was then calculated by determining the fraction of these recommendations that belonged to the same group as mentioned above.

The final precision score for the system was obtained by averaging these individual precision scores across all test cases. This method offers a clear measure of the system's ability to deliver relevant results. For comparative analysis, it is noteworthy to mention that a purely random recommendation approach is expected to yield a precision score of approximately 8%. A relatively "good" precision score would be around 50 to 60%.

VI. Test Results

The first step of testing was to evaluate each model on each feature independently. The following precision scores were obtained:

A. Description-Based Similarity

The TF-IDF model, trained solely on the descriptions consisting mainly of short sentences, demonstrated a precision of approximately 36%. This indicated that keyword-based features were significantly impactful, suggesting that simple descriptions often contain enough signal to drive useful recommendations.

B. Topic-Based Similarity

When evaluating topic-based similarity alone (with Jaccard similarity), we achieved a precision of 31%. While topics are a strong indicator of repository relevance, their absence in many repositories limits their utility as the sole feature for recommendation.

C. README-Based Similarity

For README files, which are typically more detailed and complex, a combination of keyword and semantic analysis was more informative. We evaluated three models:

- TF-IDF alone yielded a precision of 25%.
- Doc2Vec, which captures semantic document relationships, came close with a precision of 23%.
- BERT, despite its advanced capability to understand deep semantic meanings, only achieved a precision of 20%. Given its computational intensity and lower performance, BERT was subsequently excluded from the final model ensemble.

VII. Hyperparameter Tuning

The formula used to compute the final recommendation score (RS) is represented as follows:

$$\begin{aligned} RS = {} & w_1 \cdot \text{sim(desc\_TF-IDF)} + \\ & w_2 \cdot \text{sim(topic)} + \\ & w_3 \cdot \text{sim(readme\_TF-IDF)} + \\ & w_4 \cdot \text{sim(readme\_Doc2Vec)} \end{aligned} \tag{1}$$

Here w_1, w_2, w_3, and w_4 represent the weights for the similarity scores derived from different features. The tuning process involved plotting graphs to visually analyze how changes in these weights affected the precision. Upon increasing each weight from zero, the precision would initially increase until an optimal value, and then decrease. Using the test results, it was concluded that w_1 plays the most important role in precision score, followed by w_2, w_3, and finally w_4. Due to this ordering, we tried two approaches to approximating the optimal value of hyperparameters.

A. Setting w_1 to Unity

Since only the relative scale of weights with respect to each other matters, we initialize w_1 to a default value of 1 and assume this is the optimal value. By fixing the value of w_1 and setting w_3 and w_4 to 0, we plot a graph to find the optimal value of w_2 with respect to $w_1 = 1$ (Fig 1). Next, w_2 is also fixed to its predicted optimal value, and the graph of w_3 is plotted (Fig 2). Finally, w_4 is calculated using the fixed values of all the other weights (Fig 3). In this approach the weights are adjusted in **decreasing** order of importance and the approximate optimal precision score is:

- $w_1 = 1$
- $w_2 = 1.2$
- $w_3 = 0.8$
- $w_4 = 0.12$
- $P = 0.497$

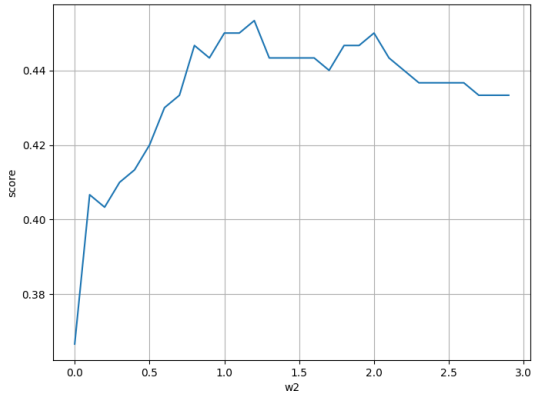

Fig. 1: Optimal value of $w_2 = 1.2$

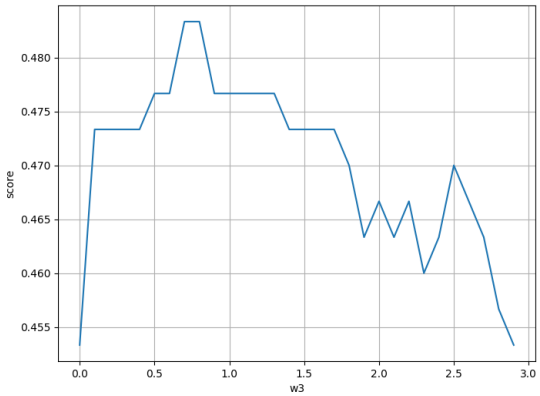

Fig. 2: Optimal value of $w_3 = 0.8$

B. Setting w_4 to Unity

This time, the reverse approach is followed as w_4 is assumed to be equal to 1, and the weights are adjusted in **increasing** order of importance. This allows w_1 to be the last weight to get fine-tuned. The corresponding graphs are shown in Fig 4, 5, and 6. The approximate optimal precision score is:

- $w_1 = 11.0$
- $w_2 = 11.5$
- $w_3 = 10.5$
- $w_4 = 1.0$
- $P = 0.503$

C. Testing Conclusion

By iteratively adjusting the weights and features, we optimized the model to achieve a precision score of approximately 50%, indicating that our system successfully recommended two to three relevant repositories out of five. These same weights would later be used in the predictions involving the training data.

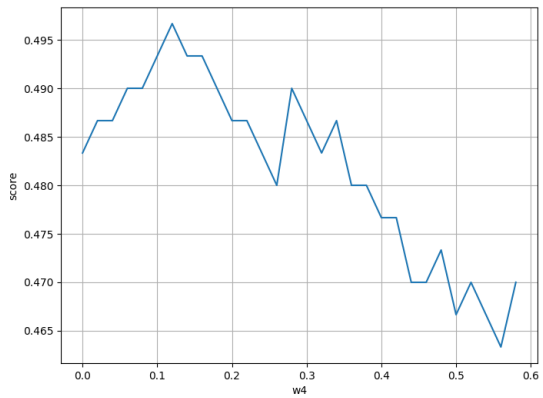

Fig. 3: Optimal value of $w_4 = 0.12$

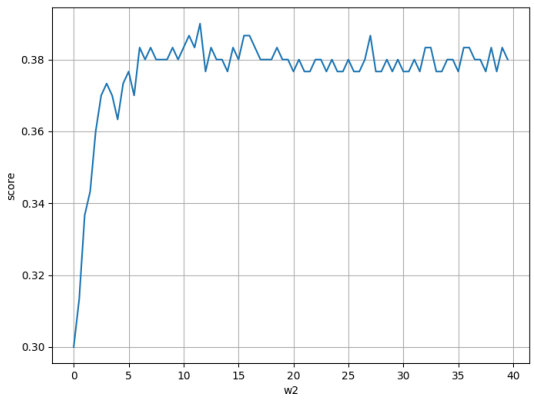

Fig. 5: Optimal value of $w_2 = 11.5$

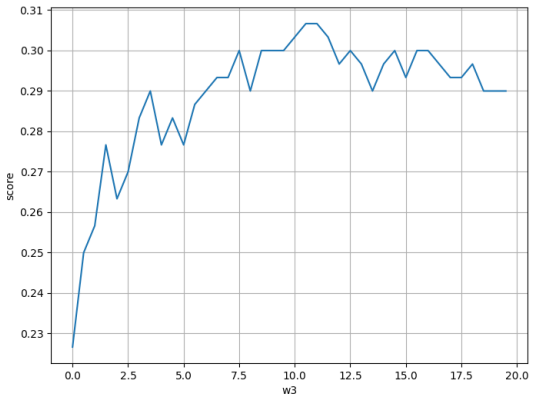

Fig. 4: Optimal value of $w_3 = 10.5$

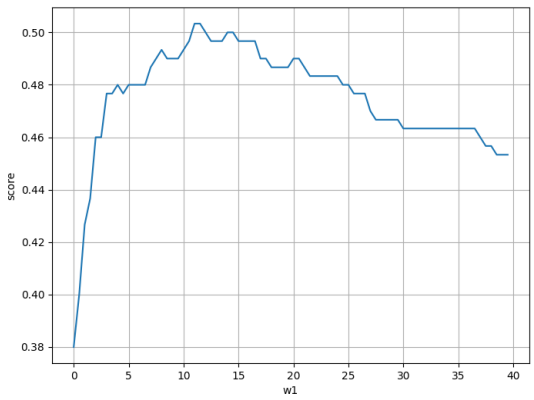

Fig. 6: Optimal value of $w_1 = 11.0$

VIII. Content Based Recommendation

A. User Profile Creation

Content based recommendation involves constructing a user profile in the same vector space as the items (repositories). For this, a list of the repositories which the user has interacted with is considered, which includes:

- Repositories created by the User
- Repositories starred by the User

B. Aggregating Repository Data

Constructing a user profile in a content-based recommendation system involves aggregating the vectors of repositories that a user has interacted with. This profile acts as a basis for identifying new repositories that match the user's interests. Three aggregation methods were considered:

1) Average: The simplest approach is to calculate the arithmetic mean of the repository vectors. This method is straightforward to implement and computationally efficient. However, it has significant drawbacks, particularly for users with diverse interests. Averaging does not differentiate between the importance or relevance of different interactions, potentially diluting distinct interests into a generic profile that does not accurately represent any specific preferences.

2) Weighted Average by Date of Interaction: To address the limitations of simple averaging, a weighted average approach was adopted, where more recent interactions are given greater importance. This method adjusts the weights of the repository vectors based on the date of interaction, thereby providing a temporal bias that reflects the user's evolving interests. This approach is especially beneficial in dynamic fields like software development, where newer interactions are often more indicative of current interests and expertise.

3) Clustering: Clustering involves grouping repository vectors into clusters that represent different interest areas. This method could potentially capture diverse interests more effectively by treating each cluster as a separate component of the user profile. However, the implementation complexity increases significantly, particularly when dealing with multiple feature vectors and the need to determine the optimal number of clusters dynamically.

C. Temporal Decay

The second aggregation function was chosen due to its simple implementation and ability to dynamically provide bias to the user's current interests. The weight α_i of a repository is modelled as an exponential decay function with respect to the time elapsed from date of creation.

$$\alpha_i = e^{-\lambda t_i} \tag{2}$$

In the final implementation, λ is chosen to give each repository a half life of two years. The weights are also normalized by dividing each weight by the total sum.

D. Recommendation

We train the NLP models, and obtain topic and language data from the training dataset consisting of 1600 popular repositories over the years. These same models are used to obtain the feature vectors for the user-interacted repositories. The feature vectors are then aggregated using the weighted average and now represent the user profile. To provide recommendations, the user vector is compared with each repository in the dataset and a resultant Recommendation Score is obtained using equation (1). The repositories with the top 10 recommendation scores are displayed to the user.

E. Mitigating the Cold Start Problem

For new users with no interaction history, the system requests manual input to create an initial topic vector. Three topics are requested by the system, and these are matched to the three most closest topics (by Levenshtein distance) belonging to the dataset. This helps in generating relevant recommendations by using this initial data to find repositories with similar topics.

F. Programming Language Filtering

Normally, programming languages would not play a large role in recommendations. Consider the situation where a library has a binding for another language. Both the library and its binding would be considered as similar repositories despite its language vectors having no similarity. Therefore, we have kept the ability to filter languages as a separate option. The language vectors are also compared using cosine similarity, and this adds an extra term in the Recommendation Score Calculation:

$$\begin{aligned} RS = {} & w_1 \cdot \text{sim(desc\_TF-IDF)}+ \\ & w_2 \cdot \text{sim(topic)}+ \\ & w_3 \cdot \text{sim(readme\_TF-IDF)}+ \\ & w_4 \cdot \text{sim(readme\_Doc2Vec)}+ \\ & w_5 \cdot \text{sim(language)} \end{aligned} \tag{3}$$

By default, the value of w_5 is set to zero, however is the language filtering option is enabled, then w_5 is set equal to w_2

IX. Dynamic Dataset

Until now, the dataset used for recommendation is static, it is a list of popular repositories over the years. Unfortunately, this may not be very relevant to the user. Users may show preference towards repositories which are trending, or connected to them (in terms of the github following mechanism). In order to fix this, we introduce dynamic datasets, where each repository belongs to three possible categories.

A. Popular Repositories:

These repositories are often stable, well-maintained, and highly reliable, making them attractive choices for users seeking established projects. By incorporating popular repositories into the recommendation pool, the system ensures that users are exposed to proven and widely respected projects, which can serve as benchmarks or learning tools. All repositories belonging to the training dataset are classified as "Popular".

B. Trending Repositories:

Trending repositories are identified through a more dynamic metric, typically based on a sharp increase in stars or forks over a recent period. This category reflects the current interests and technological trends within the developer community. Trending repositories are obtained by requesting the top repositories created in the last month, ranked by stars. The feature vectors for these repositories are obtained using the training dataset, and are added to the list of recommendation options.

C. Connected Repositories:

Connected repositories are those linked to a user's social directed graph on GitHub, which includes repositories maintained by users they follow. This dimension of recommendations leverages the social connections and network effects within GitHub, positing that users are likely to be interested in projects that people in their network are working on or endorsing.

To implement this, a BFS traversal is initiated from the main user. The graph contains edges between two users a and b if a follows b. A BFS traversal ensures that the users are visited in order of distance, so that users directly followed by the main user are prioritized. For each user visited in the traversal, their most recent repositories are obtained and added to the list of recommendation options using the training dataset.

D. Distance Metric:

In order to describe how relevant a repository is to a user, we introduce a new value called the distance metric. A repository with smaller distance is more relevant to a user. The distance metric is calculated as follows:

- For Popular Repositories, the distance is Infinity (Set to 1000 in the implementation).
- For Trending Repositories, the distance is $1 + \text{rank}/50$ rounded down.
- For Connected Repositories, the distance is the length of the shortest path from the main user to the owner of the repository in the directed graph. (This is already obtained from the BFS).

The new recommendation score is calculated as follows:

$$\text{RS'} = \text{RS} * (1 + 0.5/\text{distance}) \tag{4}$$

This equation essentially multiplies the recommendation scores of relevant repositories by a factor in the range of 1 to 1.5. The constant 0.5 was selected through human evaluation to give a diverse mix of Popular, Trending, and Connected Repositories.

Table I: Users involved in the demonstration

Username	Interests	Languages	Repository Interactions	Follows
FinalTrack	Games	HTML/Javascript, Python	5	raj-bunsha
raj-bunsha	Web Development, Machine Learning	Python	51	FinalTrack
NimishEmpty	3d, Visualization, Graphics	None	0	None

```
Top Recommendations:

CONNECTED
raj-bunsha/NetFarmer
A game for net Farming

Recommendation Score: 0.5174671241993276
1.0

---

POPULAR
leereilly/games
:video_game: A list of popular/awesome video games, add-ons, maps, etc. hos
ted on GitHub. Any genre. Any platform. Any engine.
game;game-development;game-engine;gamedev;games;html5-games;platform-game;p
uzzle-game;sandbox-game;strategy-game
Recommendation Score: 0.49314557816604865
1000.0

---

CONNECTED
drwrose/quarto
An experimental AI to play the board game Quarto

Recommendation Score: 0.39278689916366166
3.0

---

CONNECTED
rdb/angiogenesis
Procedural space horror runner game set in a tubular megastructure - PyWeek
 37 entry
horror-game;panda3d;pyweek
Recommendation Score: 0.36760828377487226
2.0
```

Fig. 7: Recommended repositories for FinalTrack

X. Implementation and Results

The implementation of the project is divided into a Test Version and Final Version. The Test Version contains the pre-trained models on the test dataset, along with evaluation and hyperparameter tuning codes. The Final Version contains the pre-trained models on the train dataset along with the dynamic dataset extension. For the sake of demonstration, we have included the final version results of three users as shown in Table 1.

Fig. 7 shows the first four recommendations for the user FinalTrack. Each recommendation is accompanied by its Tag (Popular/Trending/Connected), Name, Topics, Recommendation Score, and Distance. All of the recommendations are related to games and most of the recommendations are connected to the user.

Fig. 8 similarly shows the recommendations for the second user. The results show a mix of connected repositories, and popular web frameworks. Most of the results involve python despite not explicitly using language filtering.

A. Cold Start Demonstration

This time we will look into the user NimishEmpty, who does not have any repository interactions. The system prompts the user to enter three topics. The topics which interest the user are given in Table 1, and used as the inputs (3d, graphics, visualizations). Fig. 9 shows the corresponding recommendations. This time, a trending repository is suggested which involves

```
Top Recommendations:

CONNECTED
DaFluffyPotato/pygame-shaderlib
My WIP code for using shaders with Pygame

Recommendation Score: 0.9749448590719123
1.0

---

POPULAR
react-bootstrap/react-bootstrap
Bootstrap components built with React
bootstrap;hacktoberfest;javascript;react;react-components;typescript
Recommendation Score: 0.840471742585178
1000.0

---

POPULAR
google/python-fire
Python Fire is a library for automatically generating command line interfac
es (CLIs) from absolutely any Python object.
cli;python
Recommendation Score: 0.7770677334316509
1000.0

---

POPULAR
vercel/hyper
A terminal built on web technologies
css;html;hyper;javascript;linux;macos;react;terminal;terminal-emulators
Recommendation Score: 0.7446940901721332
1000.0

---
```

Fig. 8: Recommended repositories for raj-bunsha

```
Top Recommendations:

TRENDING
danielchasehooper/ShapeUp-public
A 3D Modeler Made in a Week
c;graphics;raymarching
Recommendation Score: 0.6123724356957945
2.0

---

POPULAR
motion-canvas/motion-canvas
Visualize Your Ideas With Code
animation;presentation;visualization
Recommendation Score: 0.40020000000000006
1000.0

---

POPULAR
algorithm-visualizer/algorithm-visualizer
:fireworks:Interactive Online Platform that Visualizes Algorithms from Code
algorithm;animation;data-structure;visualization
Recommendation Score: 0.34658336659453237
1000.0

---

POPULAR
jgraph/drawio-desktop
Official electron build of draw.io
diagram-editor;electron-app;graphics;javascript-applications
Recommendation Score: 0.34658336659453237
1000.0
```

Fig. 9: Recommended repositories for NimishEmpty

the user's preferred topics. Some other popular repositories are also recommended which involve graphics and visualizations.

B. Language Filtering Demonstration

For demonstration of language filtering, the first user is again given as input. The new recommendations are given in (Fig. 10). This time, most of the results are filtered to give repositories which primarily contain HTML, python, or both.

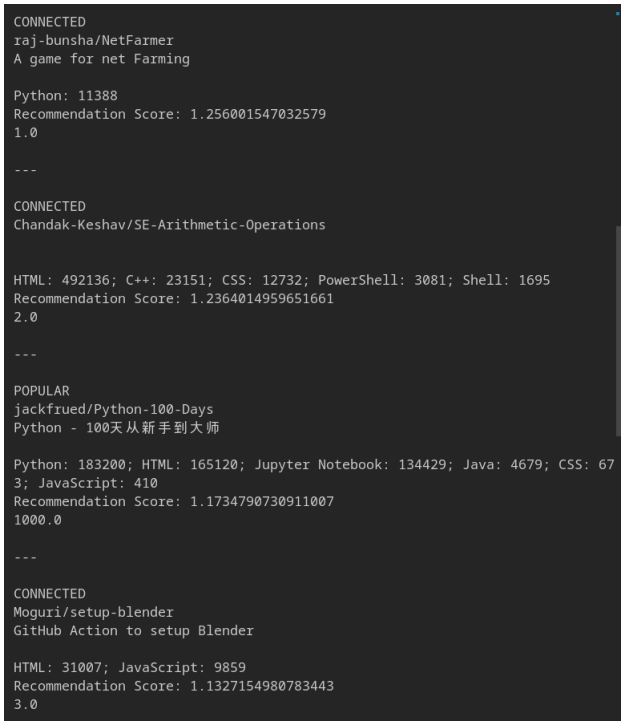

Fig. 10: Language Filtering for FinalTrack

XI. CONCLUSION

The development of RepoLens has significantly enhanced GitHub repository recommendations, enabling developers to find projects that align closely with their interests and technical expertise. By integrating sophisticated data analysis, natural language processing, and user-specific customization features, RepoLens offers personalized and relevant recommendations.

Our system effectively combines user interaction data, content-based analysis using advanced NLP models, and incorporates popular, trending, and connected repositories to provide a rich, multi-dimensional recommendation experience. Additionally, RepoLens addresses the cold-start problem and allows for adjustable recommendation parameters, ensuring adaptability to diverse user preferences.

As we continue to refine RepoLens, future updates will focus on improving algorithm accuracy and expanding features to better serve the GitHub community. The project demonstrates the potential of machine learning in enhancing user engagement and fostering a more connected open-source ecosystem.