IDENTIFYING FUNDAMENTALLY SIMILAR COURSES ON TECHTREE

Similaritywiz

Advisor:-
Prof. Dhruv Kumar
Contributors:-
Dhruv Mishra(2020296),
Hunar Kaur(2020303),
Arsh Gupta(2020285)

Index

Serial No.	Title	Page Number
1.	Motivation	3
2.	Problem statement	3-4
3.	Aim of the project	4-5
4.	Methodology	5-9
5.	Dependencies	10
6.	Interface	10-11
7.	Usage	12-13
8.	Example Queries	13-14
9.	Results	14
10.	Future Work	15-16
11.	References and Github Link to Project Repository	16

1. MOTIVATION:

The availability of a large number of courses in colleges and universities is a common phenomenon. Indraprastha Institute of Information Technology, also known as IIITD, is one such institution that offers over 100 courses to students every year. This vast array of courses can be both exciting and overwhelming for students, who often find it challenging to select the courses that best suit their academic interests and career goals.

Due to the sheer number and diversity of courses, students may face difficulty in determining which ones to enroll in. This can lead to confusion and, at times, even misjudgment on their part when it comes to choosing the right course load for a semester. The importance of choosing the correct courses cannot be understated since it directly affects the students' academic and career prospects.

Another issue that arises from having multiple courses with similar deliverables is that it creates redundancy in the course offerings. This redundancy may lead to a reduction in student registration for these courses since students may opt to take only one of the similar courses. In turn, this can make it difficult for the faculty to sustain these courses in the long run. It also splits the student registration between two courses when it could have been concentrated in a single course, which can impact the effectiveness of the courses and may lead to less satisfactory learning outcomes for students.

Overall, while the availability of a vast range of courses can be advantageous for students, it also poses challenges for them and the faculty. The institution needs to strike a balance between offering a diverse range of courses and ensuring the effectiveness of the courses offered, taking into account the interests and preferences of students, and the sustainability of the courses in the long run.

2. PROBLEM STATEMENT:

The project's main objective is to create a system that can automatically identify courses offered at IIITD (Indraprastha Institute of Information Technology Delhi) that share similar topics and learning objectives. The system would achieve this by analyzing publicly available data from IIITD's techtree, which is a repository of course-related information.

The system would use advanced data analysis techniques to process the techtree data and identify courses that are related in terms of topics and goals. It would then present

this information to the user in an easily understandable format, enabling them to make informed decisions about which courses to take based on their interests and career aspirations.

Overall, the project aims to simplify the course selection process for students at IIITD and make it easier for them to find courses that align with their academic and professional goals.

3. AIM OF THE PROJECT:

(1) This system is mainly designed for helping the students. It will help many kinds of students in different ways.

(a) **Students who want to complete their graduation credits**:- If a student is good at a certain course or topic and wants to take similar courses in order to get good grades and completed his/her credits, then they can use this system to identify similar courses which will help the student to complete their credits and even get a good grade.

(b) **Students who have an interest in a particular field of study**:- If a student has an interest in a particular field of study, they can search for courses which are similar to the field and study the courses which interest them. For example:- If a student has an interest in the field of “Machine Learning”. They can take courses which are similar to “Machine Learning”.

(c) **Students who are willing to diversify their resume**:- If a student wants to study different disciplines, then also our system finds its way of helping the student. If a student has already done a specific course. Then they can find courses which are similar to that course and opt not to take those courses. For example:- A student has already studied “Machine Learning” and they want to learn new disciplines in the field of computer science. Then they can use the system to find courses similar to “Machine Learning” and do not opt for these courses and choose courses of other disciplines.

(d) **Students who are not good/interested in a particular course**:- If a student does not like a course that they have already taken, then they can also filter out the courses that are similar to the said course and do not take similar courses.

(2) This system will also help the faculty and academic department to remove courses which are redundant and similar and reduce the effort put in by the professors and help to diversify the set of disciplines taught at IIITD.

4. METHODOLOGY:

(i) **Analysis of Dataset:**

For this project, we aimed to analyze and compare the course descriptions of various courses offered on the IIITD tech tree. To do this, we collected a dataset consisting of Excel files containing information about each course, such as its course code, pre-requisites, course outcomes, weekly lecture plans, grading distribution, and references.

Our dataset originally included over 400 Excel files, each corresponding to a different course. However, we encountered several discrepancies while working with the dataset, such as odd column and row formats, missing descriptions or course names, and same course codes for different courses to name a few.

We identified the following problems when analyzing the initial dataset:

- Odd column, row format(Some rows are columns, some columns are rows)
- Different Formats for Different Subjects(Introduction to spatial computing, CSE-555)
- Missing Description(CSE-503, Computational Metagenomics)
- Missing Course Name(Advanced Sociology Theory)
- Same Course Code(ECE-570; Control Theory and Linear Systems Theory)
- Empty Files with just course names (BIO361)
- Multi-Cell Formatting(CSE-523 Randomized Algorithm)
- Multi-Line Course Description(ECE-321, SSH-222)
- Empty Columns(ENT-411)

To ensure the accuracy and reliability of our analysis, we decided to drop the files containing discrepancies from our dataset. After this, our final dataset consisted of 390 Excel files, each representing a unique course with complete and consistent information. We then used this dataset to train and evaluate our models for analyzing and comparing course descriptions.

Overall, while we faced some challenges in working with our dataset, we were able to address these issues and obtain a high-quality dataset for our project. This dataset

enabled us to gain insights into the similarities and differences between various courses offered on the tech tree and provided a foundation for future research and analysis in this area.

(ii) **Data Extraction: (cell by cell searching)**

To ensure consistency in our dataset, we spent significant time and effort rectifying the discrepancies that we encountered. However, even after rectification, we still faced issues with the inconsistent formatting of the excel files, which posed a challenge for reading data into our code.

To tackle this problem, we developed an innovative approach that involved searching for markers within the excel files that indicated the type of information contained in a particular cell. We then used these markers to identify and extract the relevant data from each file.

For example, we used markers such as "Course Name," "Course Description," "Course Code," and "Lecture Plan" to locate the cells containing the corresponding information. Once we identified the relevant cells, we used the Openpyxl library to read the data into a Pandas dataframe, which could then be used for further analysis.

The resulting code is highly flexible and can automatically extract information from any excel file containing course information, regardless of the formatting inconsistencies present in the file. This approach has proven to be highly effective in efficiently and accurately extracting the relevant data from a large number of excel files, enabling us to create a comprehensive dataset for our project.

(iii) **Data Preprocessing:**

Once we had extracted the relevant information from each excel file and stored it in a Pandas dataframe, the next step was to preprocess the text data for analysis. This involved a series of steps to remove any irrelevant information and prepare the text for input into our model.

The first step in the preprocessing pipeline was to convert all the text to lowercase. This helped to ensure consistency in the data and eliminated any inconsistencies that may have been introduced due to the use of upper and lowercase letters.

Next, we removed any extra spaces that may have been present in the text. This was done to ensure that the text was uniform and easier to analyze. We also removed any irrelevant characters and unreadable characters that may have been present in the text.

Handling stopwords and punctuations was also an important step in the preprocessing pipeline. We used Python's Natural Language Toolkit (NLTK) library to remove common stopwords and punctuations from the text while preserving the context of the text.

We chose not to stem or lemmatize the words as their verb form is useful for preserving their context in a sentence, and context preservation is essential to us. To further ensure that our model was sensitive to context, we also designed a few custom helper functions that were used to preprocess the text data. These functions were designed to handle specific cases and ensure that the text was ready for input into our model.

Overall, the cleaning and preprocessing of the text data was a critical step in our project, as it helped to ensure that our model could accurately analyze and compare the course descriptions. By carefully removing irrelevant information and preserving the context of the text, we were able to create a clean and accurate dataset that could be used to train our model.

(iv) **Feature Extraction (Generating Embeddings):**

Extracting the relevant features to generate an appropriate embedding was a crucial aspect of this project as it heavily impacted the accuracy of the course similarity matching. There are a plethora of approaches to tackle this problem, but we wanted something robust. In our initial versions, we experimented with the following:

a. Term Frequency-Inverse Document Frequency (TF-IDF): This method calculates the importance of each word in a document relative to all other documents in the dataset. It assigns a higher weight to words that are more unique to a specific document and less weight to words that are common across documents.

b. Word2Vec: This is a neural network-based technique that generates vector representations of words in a high-dimensional space. These vectors capture the semantic meaning and relationships between words, allowing for more accurate comparisons between documents.

After experimenting with these, we concluded that we needed a more robust embedding technique that could capture the academic context of the text accurately. This is because academic texts are more complex and have specific terminology and sentence structure which require more advanced techniques to capture the meaning.

To address this, we turned to tensorflow hub's Google US encoder-4, which is a pre-trained model capable of generating embeddings that can capture the meaning of the text accurately. The embeddings generated from this model are of size 512 and contain float values ranging from 0-1, which makes them very efficient and easy to work with.

Another significant advantage of using these embeddings is that their size is independent of the vocabulary size, which means that if a new term is introduced in the dataset, we don't have to recalculate embeddings for all the strings to maintain the same size of the resulting vectors. Instead, we can simply generate an embedding for the new entry and use it to calculate the cosine similarity, which saves a lot of time and computational resources.

While the embeddings generated by the Google US encoder-4 model gave us satisfactory results, we acknowledge that there are other models and techniques available, and it is possible that some of them may provide even better results. In the future, we plan to explore other models and techniques like RoBERTa, BERT, and many others available in the Hugging Face library, to see if we can improve our results. Overall, finding an appropriate embedding technique is a crucial step in natural language processing and plays a significant role in the accuracy of the model.

(v) **Similarity Metrics:**

After feature extraction, the similarity between courses is calculated using a similarity metric. While various similarity metrics exist, the choice of metric is equally important as the choice of encoder. In our case, we chose the cosine similarity metric as it is robust to the varying lengths of the course descriptions and only considers the angle between two vectors. It is often used in natural language processing tasks as it can capture the semantic similarities between documents.
Cosine similarity is calculated by taking the dot product of two vectors and dividing it by the product of their magnitudes. A smaller angle between two vectors denotes high similarity between the two courses.

Apart from cosine similarity, we experimented with other distance metrics such as Pearson Correlation, Jaccard similarity, Euclidean distance, and Manhattan Distance. Pearson correlation measures the linear relationship between two variables, and Jaccard similarity measures the similarity between two sets. Euclidean distance measures the distance between two points in a Euclidean space, while Manhattan distance measures the distance between two points in a grid-like path.

However, after verifying the results with human subjects, we found that cosine similarity consistently provided us with the most accurate and reliable results. It is also computationally easier to calculate than other metrics, making it an efficient choice for our project.

(vi) **Scoring and Results:**

After the process of extracting features and calculating similarities between courses is complete, the final step is to present the results in a visually appealing and easy-to-understand format. To achieve this, we organize the results by listing the recommended courses in ranked order of their similarity to the input course. This ranking is arranged in the descending order of similarity to the input course, with the most similar course at the top of the list.

When generating recommendations for multiple input courses, we calculate the similarity of each input course with all courses in the dataset and take the unweighted mean of these similarities. The resulting unweighted mean similarity value is then used to rank the recommended courses in descending order. This allows us to provide recommendations that take into account the similarities between multiple input courses, giving a more comprehensive list of recommendations that cover all the input courses.

Overall, presenting the results in a visually appealing and easy-to-understand format is important for users to quickly and easily interpret the recommendations. By listing the recommended courses in a ranked order of similarity and taking the unweighted mean of similarities with respect to multiple input courses, we provide users with an intuitive way to understand and explore the recommendations.

5. DEPENDENCIES:

For running this code, you need Python 3 (3.7.9 recommended) installed on your system.

You must have the the following dependencies installed:

- Pandas
- Numpy
- Tensorflow
- Tensorflow Hub
- Sklearn
- Difflib
- Pickle
- Openpyxl

Ensure that you have atleast 50 MB of free space on your system.(Although tensorflow hub's US encoder may take upto 1 GB space in temporary files).

6. INTERFACE:

We designed the interface of our project on figma. Following are the High Fidelity prototypes, once completed, our system should look like this:

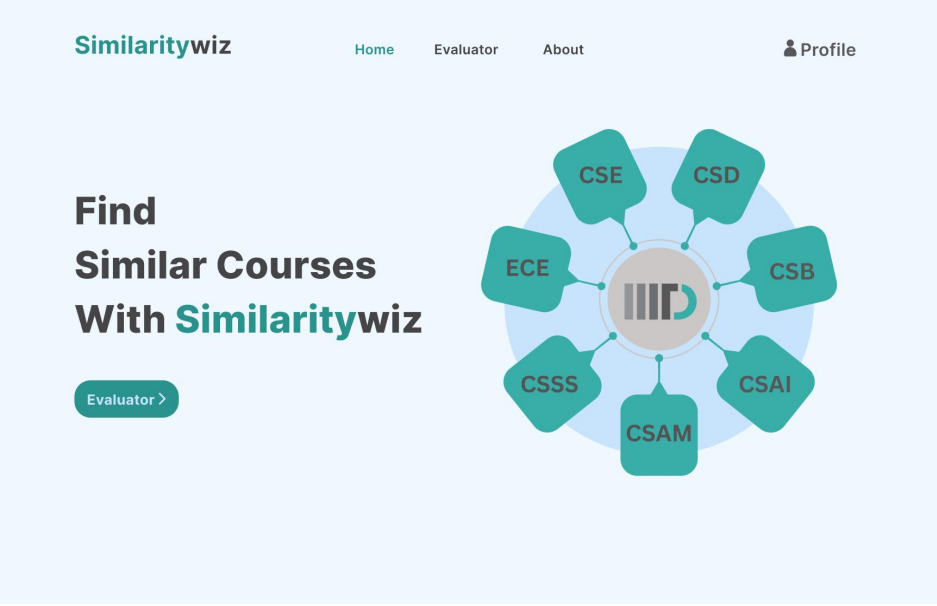

Fig 1. The above image represents the layout of our application's homepage, which serves as the primary hub for users to navigate through the various features and functionalities of the platform.

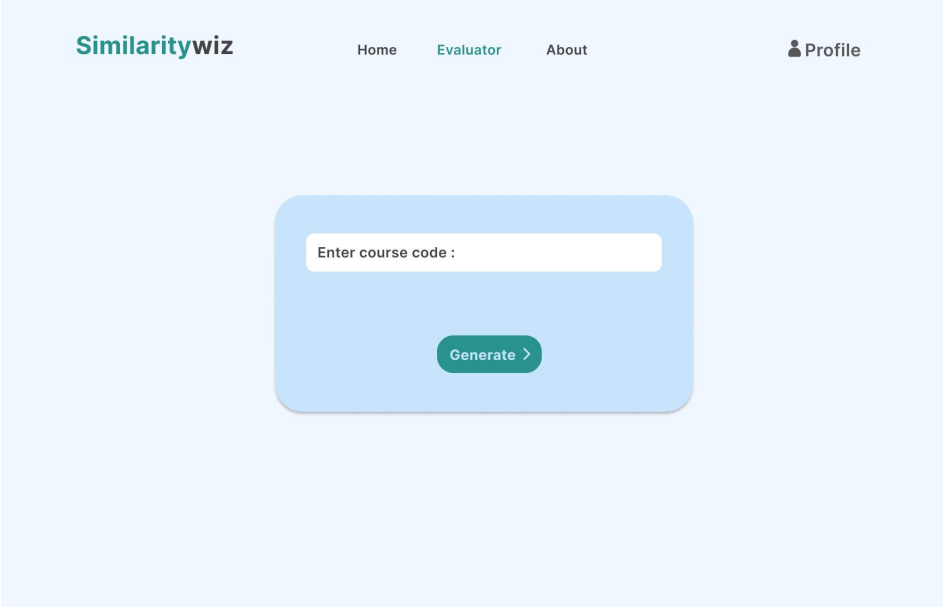

Fig 2. The above page represents the layout of the evaluator page of our application. The users enter their queries in the provided text field.

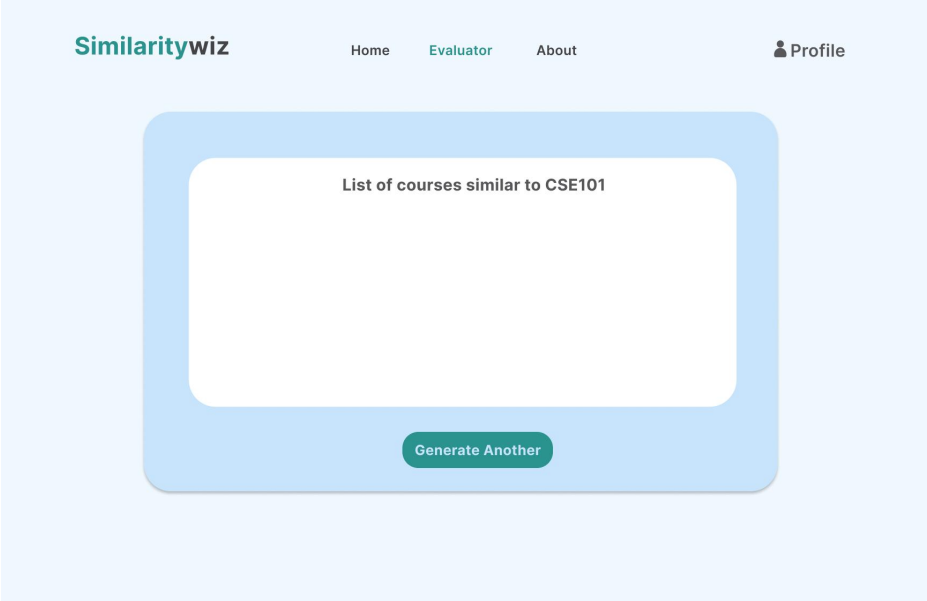

Fig 3. The above page represents the layout of the search results page of our application. The top results for the query will be displayed in the white region

Link to High-fi prototype:
[Figma Link]

7. USAGE:

The project aims to provide a comprehensive system that can assist college students in selecting the courses that best align with their interests and academic goals. Students often face challenges when choosing courses due to the vast and diverse range of options available. As a result, students may choose courses that do not align with their interests or fail to consider courses that would benefit their academic and career development. Moreover, our project can also help students avoid redundancy in their course structures by identifying courses with similar course structure and syllabus. By doing so, students can make informed decisions about their course load and avoid taking courses that may have similar content and learning objectives, leading to redundant coursework.

In addition to assisting students, our project can also benefit professors in identifying courses with similar course structures and syllabus. This system would enable professors to avoid redundancy in course offerings and ensure that the courses provided are relevant and effective.

Following are the steps to use our project:

1. To start, you need to initialize a Course_Loader object. This is where all the data will be stored, processed and similarity will be calculated. It can be initialized in the following way:

   ```
   c = Course_Loader()
   ```

2. Once the object is loaded, you now have to add all the xlsx files containing the information about the courses. To add a new course, you simply have to call the addCourse function of your Course_Loader object. For this you need to retrive the directory of the .xlsx files containing your data. In our case, the files are located in the Data folder in the same directory as our python file.

   ```
   path = os.getcwd()
   path = str(path) + "\\Data\\"
   csv_files = glob.glob(os.path.join(path, "*.xlsx"))    #Retrieving the directory of the data

   for f in csv_files:
           c.addCourse(f)    #Adding each course using the addCourse function
   ```

3. Now that all the courses have been added, you can use the system. Note that you may add a new course at any given time and the system will update itself to include this in the results. Now, to make a query, you have to call the combine_recommendations function of your Course_Loader object. The input to this function will be a list of strings, here each string denotes the course that you wish to input (see the examples for more clarity).
   It would look something like this:

   query = ["Course-1","Course-2","Course-3", ...] #The courses which you wish to search
   c.combine_recommendations(query)

4. Upon running this, you would get the desired output in the form of a pandas dataframe. The output would contain the top 10 courses similar to your entered courses, in the ranked order with first being the most similar.

8. EXAMPLE QUERIES:

(i) Demonstrating Single Input Query:

```
query = ["machine learning"]
c.combine_recommendations(query).head()
```

✓ 0.0s

	Course Name	Course Code	Course Description
177	bayesianmachinelearning	ece551/cse515	In this course, we will discuss the foundation...
54	statisticalmachinelearning	cse342/cse542	This course will introduce students to salient...
105	collaborativefiltering	cse640	Recommender systems have around for sometime. ...
107	advancedmachinelearning	cse642	This is an advanced course on Machine Learning...
278	categoricaldataanalysis(cda)	mth576	The study of categorical data sets involve com...

(ii) Demonstrating Multi Input Query:

```
query = ["mechine learning", "artifical inteligence"] #Demonstrating the Spell Correction Ability of the System
c.combine_recommendations(query).head() #Making a query
```

✓ 0.0s

	Course Name	Course Code	Course Description
55	machinelearning	cse343/cse543/ece563	This is an introductory course on Machine Lear...
54	statisticalmachinelearning	cse342/cse542	This course will introduce students to salient...
177	bayesianmachinelearning	ece551/cse515	In this course, we will discuss the foundation...
107	advancedmachinelearning	cse642	This is an advanced course on Machine Learning...
105	collaborativefiltering	cse640	Recommender systems have around for sometime. ...

(iii) Demonstrating Similarity Calculation for Two Inputs:

```
query = ["machine learning","artifical inteligence"]
print("Cosine Similarity of "+query[0]+" and "+query[1],c.caclulate_similarity(query))
```

✓ 0.0s

```
Cosine Similarity of machine learning and artifical inteligence 0.59952796
```

9. RESULTS:

The use of a deep learning approach allowed us to capture the semantic meaning of the course description and plan, which helped in finding similarities not only based on terminology but also based on the overall meaning of the text. This resulted in a more diverse set of recommendations, which was appreciated by the users.

Due to the inclusion of the course plan along with the description, the model was able to detect similarities between courses of different disciplines as well. We consulted human subjects to evaluate the usability of our system, and the feedback was overwhelmingly positive. From the user reviews and upon manual inspection of course material, we found our outputs to be very relevant to the given inputs.

To verify the system's performance with performance metrics like accuracy, precision, recall, and F1 score we would require a dataset which contains information about similar courses. Unfortunately, at the time of writing this, such a dataset does not exist for our college.

Overall, our system proved to be an effective tool in helping students find courses that match their interests and requirements, and we believe that it can be further improved by incorporating more advanced deep learning techniques and expanding the dataset to include more diverse academic disciplines.

10. FUTURE WORK:

If we had more time on our hands, we could have explored several additional ways to enhance this project. One important aspect that can be added is to develop a fully functioning user interface and make this project into a web application. A simple and intuitive design can be created that consists of an IIITD Gmail login page. Once the user logs in, they can easily search for courses by entering a course name or code, and the application can display similar courses.

Furthermore, the project can be extended to include more advanced features, such as identifying the most dissimilar courses. This can be achieved by clustering courses into different disciplines and then querying for courses that are dissimilar to the user's selected discipline. For instance, if a student wants to study a course on Artificial Intelligence, they can enter a query for similar courses that belong to the discipline of "Data Science." Similarly, if they want to find courses that are similar to Artificial Intelligence but are in a different computer science discipline, they can specify that option.

In addition, we can explore different embedding models from various transformers such as RoBERTa, BERT, and others, which can help improve the results. The Hugging Face interface provides access to hundreds of transformers that can be used to generate text embeddings. These embeddings consider the context in which a word appears in a sentence, making them more informative and accurate than traditional embedding methods.

There are several similarity metrics that can be evaluated, such as Euclidean distance, Manhattan distance, Minkowski distance, and Jaccard similarity. Although cosine similarity is the most commonly used metric for identifying similarity between string embeddings, other metrics can also be explored. Since cosine similarity is insensitive to the difference in magnitude and can handle sparse data effectively, it was the ideal choice for our project. However, other metrics may provide better results in certain situations and could be investigated further.

The UI/UX of the system is very basic at the moment, therefore, we plan to enhance our application’s interface by incorporating design principles into it to make it more visually appealing while also being functional and intuitive to use.

Lastly, we can incorporate additional information available in the excel files provided on Techtree, such as course outcomes and more detailed course descriptions. This would make the model more robust and informative, allowing for more accurate recommendations. Despite our best efforts and time constraints, we believe that there is still considerable scope for future work in this project. By incorporating these enhancements, we can make this project more comprehensive and provide better recommendations to students at IIITD.

11.REFERENCES:

[1]http://techtree.iiitd.edu.in/
[2]https://towardsdatascience.com/nlp-preprocessing-with-nltk-3c04ee00edc0
[3]https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder
[4]https://tfhub.dev/google/universal-sentence-encoder/4
[5]https://towardsdatascience.com/calculate-similarity-the-most-relevant-metrics-in-a-nutshell-9a43564f533e

Project Github Repository:
https://github.com/Dhruv-Mishra/Course-Similarity-Evaluator