vxa210030
Venkata Jayasri Alla
CS6320.002

Automation of the Prerequisite Waiver Process

PROBLEM STATEMENT

For any graduate course the students will be given prerequisites based on their undergraduate courses. Students can waive their prerequisites by submitting the related work experience or transcripts to the advisor for the waiver. To register in higher-level courses, students must first obtain a temporary waiver, then a permanent waiver, following a committee's evaluation on a case-by-case basis. It's a lengthy process as many incoming and transfer students request a prerequisite waiver. We aim to automate this process by using machine learning based text matching algorithms.

INTRODUCTION

The primary goal of this project is to create a prototype for an automated prerequisite waiver system. This system is designed to determine if a student's previous undergraduate coursework covers the material required for specific prerequisite courses, thus allowing them to bypass these courses. Currently, this process relies on manual intervention from academic advisors. However, by utilizing text matching machine learning algorithms, we aim to automate the evaluation process.

This project builds upon previous work done by my team in database design. I have enhanced the project by implementing a more robust code structure and integrating security measures into the web application. Additionally, I have improved the accuracy of the model by incorporating concepts from natural language processing (NLP) that I have learned during this semester. These modifications enhance the overall functionality and effectiveness of the system, providing a more reliable and efficient solution for evaluating prerequisite waivers.

The prototype is tailored for assessing eligibility for prerequisite courses commonly assigned to incoming graduate students at the University of Texas, Dallas. Specifically, the system targets courses such as CS 5333 (Discrete Structures), CS 5348 (Operating Systems), and CS 5343 (Data Structures and Algorithms). By analysing the syllabus and content of these courses alongside a student's academic history, our system determines whether the student's prior coursework sufficiently aligns with the requirements of the targeted prerequisites.

vxa210030
Venkata Jayasri Alla
CS6320.002

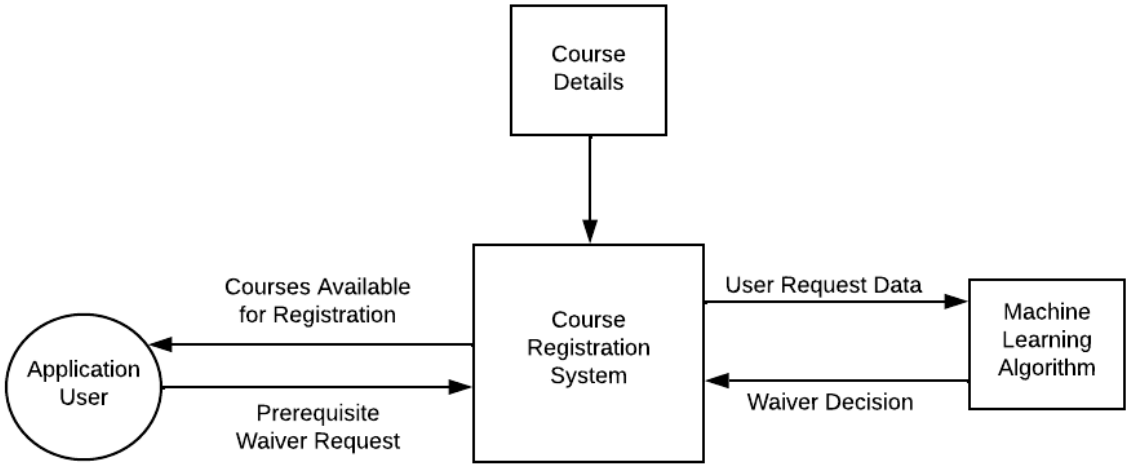

Fig 1 – Flowchart of the proposed system

RELATED WORKS

I. Preprocessing - Most natural language processing (NLP) algorithms require preprocessing to be done prior to the text being transformed. In order for unstructured text data to be analysed, it must be cleaned and transformed. This preprocessing pipeline involves multiple steps including normalization, tokenization, stop-word removal, stemming, and lemmatization. The goal of text preprocessing is to be able to represent each text document into a feature vector.

i) Normalization - In the initial stage of the preprocessing pipeline, we begin by converting all the characters in the text to lowercase and eliminating any unnecessary special characters, such as punctuation marks. This process, known as text normalization, aims to standardize the format of words within a document. By converting non-standard words into a common format, such as plain text, we ensure consistency and facilitate further analysis. For instance, a sentence like "Hi man, How are You?" would be transformed to "hi man how are you" as the normalized text.

ii) Tokenization -

Tokenization is a crucial step in natural language processing where we break down normalized text into smaller units called tokens. These tokens serve as the basic units for further analysis by NLP algorithms. The process involves three key steps. First, the text is converted into word counts, forming a bag of words (BOW) representation that helps identify significant topics. Then, the text undergoes cleaning and filtering to remove empty sequences and unnecessary characters, ensuring a more streamlined dataset. Finally, the cleaned text is segmented into a list of features or tokens, representing individual words, which forms the foundation for subsequent analysis and interpretation.

iii) Stop-Word Removal - Stop-word removal involves getting rid of common words in text that don't contribute much to its overall meaning. These words, like "the," "is," and "are," are often used frequently but don't add significant information. Removing

stop-words is important for NLP tasks because it helps algorithms focus on the more meaningful content of the text. There are different methods for identifying and removing stop-words, which can be broadly categorized into static and dynamic approaches. For instance, in the sentence "there are three high-level steps involved in the tokenization process," stop-word removal would result in "three high-level steps involved tokenization process."

iv) Stemming - Stemming is like finding the basic form of a word by removing extra bits like prefixes and suffixes. It's handy in uni when you're dealing with lots of words that are related but have different endings. By stemming them, you make it easier for the computer to understand and analyze them. For instance, if you have a sentence like "running races is funnier than running," stemming would turn it into "run race is funnier than run."

v) Lemmatization - Lemmatization involves assembling the inflected parts of a word into a single element, namely the word's lemma or underlying form. Although this process is quite similar to that done in stemming, there is additional complexity as the meaning or sentiment of words are also being considered. Lemmatization attempts to identify the root word while stemming attempts to identify the root stem. Lemmatization attempts to connect various words which are similar in their meaning but appear different on the surface into a singular root word. Through lemmatization, an accurate word is produced regardless of any original word derivative or tense. For example; "The quick brown foxes jumped over the lazy dogs." will be converted to "The quick brown fox jump over the lazy dog."

II. Transforming text into embeddings - Transforming text into embeddings is a fundamental process that involves representing words, phrases, sentences, or entire documents as continuous numerical vectors called word embeddings or word vectors.

i) Count vectorizer - Count vectorizer determines the frequency of a word in a document. A list of numbers reflecting the count of each word in the text makes up the resultant vector [3]. The advantage of count vectorizer is that it produces less sparse vectors than one-hot encoders but is more efficient. However, it ignores the connections between words' semantic meanings and has a bias towards frequently appearing words which can be a disadvantage.

ii) Term frequency-inverse document frequency (TF-IDF) - TF-IDF considers both a word's frequency inside a text and its frequency throughout the whole corpus. Compared to vectors generated by count vectorizers, this method yields vectors that are less sparse and more informative [4]. This results from TF-IDF taking into consideration a word's frequency in both the text and the full corpus. Although more difficult to build than count vectorizers, it nonetheless ignores the semantic connections between words. Additionally, TF-IDF has slow processing for large vocabularies.

iii) Word2vec - Word2vec is a family of neural network models that acquire the ability to vectorize words. With the help of a large corpus of text, Word2vec models are trained to represent words in a manner that accurately reflects their semantic associations [5].

They require a huge corpus of text to train, which may be computationally costly. Word2vec is also unable to handle words which are unknown or out-of-vocabulary (OOV), leading to a random vector being assigned.

iv) GloVe - GloVe is an additional neural network model that picks up word representation as vectors. Although GloVe is trained on a distinct goal function, it is comparable to Word2vec. Word2vec models are often combined with GloVe models to provide even more sophisticated word embeddings [6]. In terms of benefits and performance, this method is comparable to Word2vec, but may be more effective to train. However, hyperparameter tuning is more challenging than with Word2vec. Additionally, memory costs are also higher due to the usage of a co-occurrence matrix and global information.

v) BERT - BERT is a more modern neural network model that has produced cutting edge outcomes for several NLP jobs. Because BERT is a bidirectional encoder model, it acquires the ability to represent words while accounting for context. BERT models may be used for a range of tasks, like text categorization, natural language inference etc. The BERT model is trained on a vast corpus of text [7]. Due to its large size, many computations are required, which can be more expensive compared to other models.

III. Text similarity matching

i) Cosine Similarity - Cosine similarity is a useful tool for capturing semantic meaning between documents. Cosine similarity compares term vectors to determine how similar text documents are, although it has difficulty appropriately capturing semantic subtleties. The article [8] explores the shortcomings of the conventional cosine similarity approach and suggests an improvement.

ii) Levenshtein distance - String metric called the Levenshtein distance is used to calculate how different two strings are from one another. The Levenshtein distance calculates the difference between two texts using a matrix and dynamic programming. Similarities between sections of the compared texts can be found by drawing a diagonal line in the matrix based on the Levenshtein distance. [9]

iii) Jaccard Index - The Jaccard similarity coefficient, often known as the Jaccard index, calculates how similar two sets are to one another. It is defined as the ratio between the sizes of the sets' union and intersection. In order to determine the Jaccard similarity coefficient, the research article [10] examined a method that measures similarity between correct grammatical syntax and error-related similarity.

iv) Hamming Distance - The distance between two strings of the same length is measured using the Hamming distance. It is defined as the number of points at which the corresponding symbols differ. In order to build similarity search based on user profiles, the research [11] introduced a new approach in which profile vectors are used to characterize users. The ability of these profile identifiers to preserve vector similarity in the Hamming distance between them was demonstrated.

IV. Transformers - Transformers are a class of deep learning models first introduced in the paper "Attention Is All You Need" by Vaswani et al [12]. Some well-known transformer

models like BERT, RoBERTa and GPT-3 show extraordinary performance on a wide range of NLP tasks. Transformers use attention layers to catch the long–range dependencies in text and are based on encoder and decoder architecture. Some of the features of transformers are -

i) Attention Mechanism: Transformers use "self-attention" mechanism. This enables the model to consider the entire context when making predictions, capturing long-range dependencies in the data.

ii) Parallelism: Transformers are highly parallelizable, which means they can process input sequences in parallel, making them much faster to train and evaluate compared to earlier sequential models like LSTMs and RNNs.

iii) Layered Architecture: Transformers are typically composed of multiple layers, with each layer consisting of an attention mechanism and feed-forward neural networks.

iv) Bidirectional Processing: Transformers can process text in both directions helping to capture contextual information more effectively.

v) Transfer Learning: Transformers excel at transfer learning. You can take a pretrained transformer model and fine-tune it on a specific NLP task with a relatively small amount of task-specific data.

IMPLEMENTATION

The implementation details of the various subsections of the project are as follows-

I. Dataset collection – A test dataset of syllabus of various undergrad courses at multiple universities related to the subjects Operating Systems, Discrete Maths and Data Structures & Algorithms was created. Each file was labelled “Yes" or “No,” indicating whether a prerequisite waiver would have been granted if a student had uploaded that file. This dataset was used for calculating the accuracy of the three text matching algorithms implemented.

II. Text matching – Three major machine learning based text matching techniques in this project – TF-IDF and Cosine Similarity, Word2Vec and Cosine Similarity, Word2Vec and Soft-cosine Similarity.

i) Preprocessing the data – Before analyzing unstructured text data, it undergoes a process of cleaning and transformation. In preprocessing steps, the text is first extracted from the PDF file submitted by the student. This extraction process is facilitated by the 'Pypdf' Python library. Next, functions from the 'NLTK' (Natural Language Toolkit) are used to remove stop words and perform lemmatization on the extracted text. This ensures that the text is in a suitable format for further analysis and processing.
ii) Implementing Word2Vec – Word2Vec represents a family of neural networks designed to convert words into vectors, capturing their semantic meanings. To implement Word2Vec, the 'Gensim' library in Python is used along with the pretrained 'glove-wiki-gigaword-300' model. This model was trained on an extensive corpus of text, comprising 5.6 billion tokens and over 400,000 unique words from Wikipedia

2014 and the Gigaword dataset. The training process enables the model to effectively encode words based on their contextual relationships.

After vectorizing the words,two similarity measures are used: cosine similarity and soft-cosine similarity. Cosine similarity calculates the cosine of the angle between two vectors, providing a measure of their similarity in a multi-dimensional space. Soft-cosine similarity, on the other hand, takes into account semantic similarities between pairs of features in addition to their direct similarities.

To implement cosine similarity, 'Scipy' Python library is used, while for soft-cosine similarity,functionalities from the 'Gensim' library are used. These similarity measures play a crucial role in determining the resemblance between two texts, aiding in various natural language processing tasks.

iii) Implementing TF-IDF – TF-IDF considers a word's frequency inside a text and its frequency throughout the whole corpus and measures how important a term is within a document. 'Scikit-Learn' library is used for implementing TF-IDF vectorizer and then cosine similarity is used to get the similarity between documents.

III. Building a mock dataset of students – A mock dataset of students was built with a separate login id and password for each student, prerequisite courses assigned to each student, and prerequisite courses required for different courses. This dataset was used to display prerequisite courses assigned to a student and courses a student can select in the prototype. If a prerequisite waiver for a student gets approved, he/she will be able to view the updated list of courses he/she can take. A table with student name, student id and login details; a table for course details of all the courses; a table for the prerequisites assigned to each student and a table describing the prerequisites for each course are designed. MySQL was used for building this mock dataset.

IV. Building a web application – A prototype web application where a student will see the courses he/she can take next semester and the prerequisite courses assigned to him/her. The student will have the option to request a prerequisite waiver by uploading the completed relevant coursework and will see the result of the prerequisite waiver immediately. The Word2vec with soft-cosine method is used to determine if a student's uploaded coursework is sufficiently similar to the corresponding UTD syllabus. If the similarity value computed passes a predetermined threshold, a prerequisite waiver is granted. HTML, CSS and JavaScript was used for building the frontend of the application. PHP and the Flask library in Python were used for connecting the application to the database and executing the machine learning code for getting results for prerequisite waiver.

vxa210030
Venkata Jayasri Alla
CS6320.002

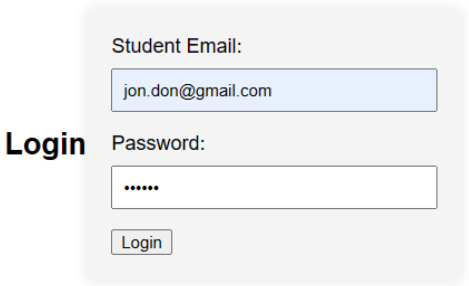

Fig 1 - Screenshot of the home page of the prototype application

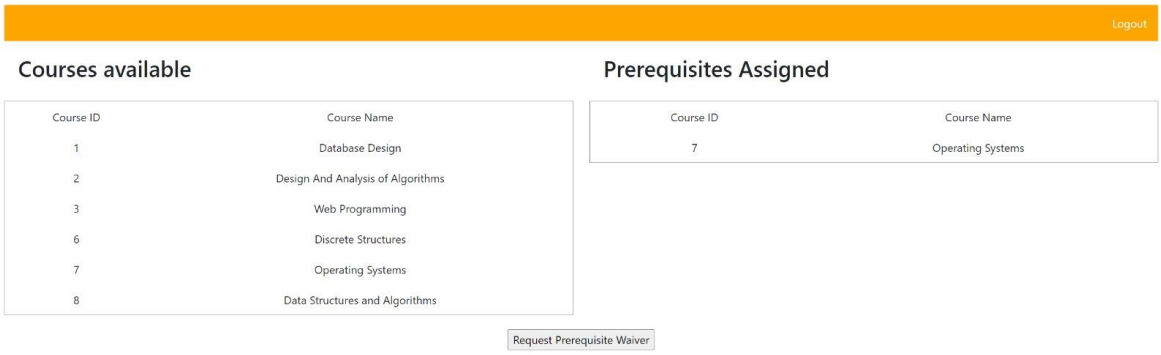

Fig 2 - Screenshot of the home page of the prototype application

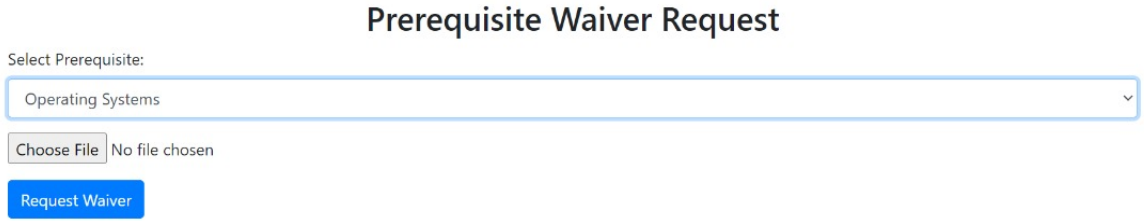

Fig 3 - Screenshot of the waiver page of the prototype application

KEY TECHNIQUES USED

I. Stop Word Removal - Stop-words are the frequently used words which have minimal information associated with it. For example, ‘a’, ‘the’, ‘be’, ‘in’ etc. Stop-words are removed as they are not essential to the meaning of the document and can inhibit its analysis. NLTK has a corpus of stop words in English, which we have used to identify the stop words which need to be removed from the text. Stop word removal is done by iterating through the text and filtering out each stop word which exists in the NLTK English stopword set.

II. Lemmatization - Lemmatization aims to reduce inflected words to their root word form, through the process of identifying an inflected word's lemma based on its intended meaning. Based on the context, the intended part-of-speech and meaning of the word need to be accurately determined. For example, the word "walking" or "walked" is converted to "walk." We have implemented lemmatization by utilizing the NLTK WordNetLemmatizer. WordNetLemmatizer creates a mapping between a word and its semantic relations, or synonyms. Synonyms are grouped into synsets such that words which are semantically equivalent belong to the same synset.

III. Word2Vec - Word2Vec is a family of model architectures which can be used to learn word embeddings from a large corpus of text, allowing for many applications such as text similarity, sentiment analysis, etc. The output of Word2Vec models is a set of embeddings which are associated with each unique word passed through the algorithm. The generation of word embeddings involves transforming each individual word into a numerical representation, namely a vector, which is then learned in a way that resembles a neural network. In the context of the entire corpus of text, each vector aims to capture various characteristics of the word such as the semantic relationship, context, definitions, etc. For example, a simple one-hot encoding of text data is as follows: have = [1,0,0,0], a = [0,1,0,0], good = [0,0,1,0], day = [0,0,0,1]. To implement Word2Vec, we have utilized the 'glove-wiki-gigaword-300' model. The GloVe model is based on matrix factorization on a word context matrix. A large matrix is constructed of co-occurrence information, and for each word, we count how frequently that word is seen in some context throughout the large corpus. A major principle of GloVe is to learn ratios of cooccurrence probabilities. For each word, the frequency distribution of words that occur near them is encoded. [13]

IV. TF-IDF (Term Frequency-Inverse Document Frequency) – It is a statistical measure which computes the importance of a term within a document relative to a corpus. Through a text vectorization process, words in a body of text are transformed into importance numbers. TF-IDF vectorizes a word by multiplying the word's term frequency by its inverse document frequency. Term frequency is the number of times a word appears in the document relative to the total number of words in the document. TF = number of times a word appears in the document / total number of words in the document. Inverse document frequency is the proportion of documents in the corpus which contain that specific word. IDF = log(number of documents in the corpus / number of documents in corpus which contain the specific word). TF-IDF can be calculated using the following formula: TF-IDF = TF * IDF. The TF-IDF value calculated indicates that the importance of a word is high when it appears frequently in a given document but infrequently in others. To implement TF-IDF we have used the 'Scikit-Learn' library which can convert a collection of raw documents to a matrix of TF-IDF features. [14]

EVALUATION

A dataset of 35 university syllabus related to the courses Operating Systems, Discrete Maths and Data Structures & Algorithms was collected. Each syllabus in the dataset was compared to the corresponding UTD syllabus for that course. Each syllabus was labelled with either a "Yes" or "No", indicating whether that syllabus sufficiently matches the UTD syllabus for that course and a prerequisite waiver should be granted. The performance of the three text matching methods (Word2Vec with Cosine Similarity, Word2Vec with Soft-cosine Similarity, TF-IDF with Cosine Similarity) is analyzed upon the dataset. If the text matching method produced the

correct prerequisite waiver response, it is noted as being correctly classified. The accuracy of each text matching method is calculated as follows:
$accuracy = (total\ no.\ of\ correctly\ classified\ syllabi/total\ number\ of\ syllabi) * 100$.
When the mentioned three text matching techniques are tested on the test dataset, the following results are obtained.

Text Matching Method	Accuracy
Word2Vec with Cosine Similarity	86%
Word2Vec with Soft-cosine Similarity	90.76%
TF-IDF with Cosine Similarity	82.29%

Table 1 - Accuracy of different text matching techniques on the test dataset

CONCLUSION – We developed a prototype for an automated prerequisite waiver system, employing text matching machine learning models. Our project involved implementing and comparing the performance of three text similarity matching methods: Word2Vec with Cosine Similarity, Word2Vec with Soft-cosine Similarity, and TF-IDF with Cosine Similarity. After analyzing a dataset of university syllabi for courses like Operating Systems, Discrete Maths, and Data Structures & Algorithms, we found that the Word2Vec with Soft-cosine Similarity method achieved the highest accuracy. This method was then used in our prototype to determine whether a student's uploaded syllabus is adequate for a prerequisite waiver. To enhance the prototype further, we took several actions. These included implementing file authentication for student uploads, building a more comprehensive and accurate test dataset, and using a more precise text matching model. These improvements aim to make the system more reliable and effective in evaluating prerequisite waivers.

GITHUB URL: Automated-Prerequisite-Waiver

DEMO VIDEO: Demo Link

REFERENCES

[1] P. M. Rahate and M. Chandak, ‘‘An experimental technique on text normal1013 ization and its role in speech synthesis,’’ Int. J. Innov. Technol. Exploring 1014 Eng., vol. 8, no. 8S3, pp. 1–4, Jun. 2019.

[2] D. J. Ladani and N. P. Desai, "Stopword Identification and Removal Techniques on TC and IR applications: A Survey," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 466-472, doi: 10.1109/ICACCS48705.2020.9074166.

[3] Burges, Christopher J.C., Robert Ragno, and Quoc Le. "Learning to Rank for Information Retrieval" (2005).

[4] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. "Statistical Methods in Information Retrieval" (2008).

[5] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space" (2013).

[6] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation" (2014).

[7] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (2018). [10] Rahutomo, Faisal, Teruaki Kitasuka, and Masayoshi Aritsugi. "Semantic cosine similarity." The 7th international student conference on advanced science and technology ICAST. Vol. 4. No. 1. 2012.

[8] Rahutomo, Faisal, Teruaki Kitasuka, and Masayoshi Aritsugi. "Semantic cosine similarity." The 7th international student conference on advanced science and technology ICAST. Vol. 4. No. 1. 2012.

[9] Zhang, Shengnan, Yan Hu, and Guangrong Bian. "Research on string similarity algorithm based on Levenshtein Distance." 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). IEEE, 2017.

[10] Niwattanakul, Suphakit, et al. "Using of Jaccard coefficient for keywords similarity." Proceedings of the international multiconference of engineers and computer scientists. Vol. 1. No. 6. 2013.

[11] Apostolico, Alberto, et al. "Sequence similarity measures based on bounded hamming distance." Theoretical Computer Science 638 (2016): 76-90.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” arXiv:1706.03762 [cs], Dec. 2017. arXiv: 1706.03762.

[13] CR, Aravind. “Word Embeddings in NLP: Word2vec: Glove: Fasttext.” *Medium*, Analytics Vidhya, 10 Sept. 2020.

[14] Fatih Karabiber Ph.D. in Computer Engineering, et al. “TF-Idf - Term FrequencyInverse Document Frequency.” *Learn Data Science - Tutorials, Books, Courses, and More.*