Automation of the Prerequisite Waiver Process using Machine Learning

PROJECT NATURE

We will be completing an implementation-focused project. The paper we intend to implement is *A Comparison of Semantic Similarity Methods for Maximum Human Interpretability* authored by Pinky Sitikhu, Kritish Pahi, Pujan Thapa, and Subarna Shakya (Department of Electronics and Computer Engineering, Tribhuwan University), published by IEEE in the 2019 Artificial Intelligence for Transforming Business and Society (AITB) conference.

GROUP MEMBERS

Nisha Shibu, Varad Abhyankar, Atharva Kulkarni, Jayasri Alla, Tejas Chaudhari

PROBLEM STATEMENT

Incoming and transfer masters' students at the university are given prerequisite courses based on their program, undergraduate transcripts and work experience to ensure students have a sufficient background in their prospective program. Incoming students can appeal these prerequisites based on their undergraduate coursework or work experience. 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 large number of incoming and transfer students request a prerequisite waiver. We aim to automate this process by using machine learning based text matching algorithms.

INTRODUCTION

We aim to create a prototype of an automated prerequisite waiver system. We intend to create a system that will determine whether one's previous undergrad coursework is sufficient to opt out of taking corresponding prerequisite classes. Currently this process is manual, requiring intervention by academic advisors. However, by utilizing text matching machine learning algorithms, we can determine whether the student's previous coursework sufficiently matches the prerequisite course's syllabus. The prototype version is currently designed for the prerequisites CS 5333 (Discrete structures), CS 5348 (Operating Systems) and CS 5343 (Data Structures and Algorithms) which are commonly assigned to incoming graduate students at UTD.

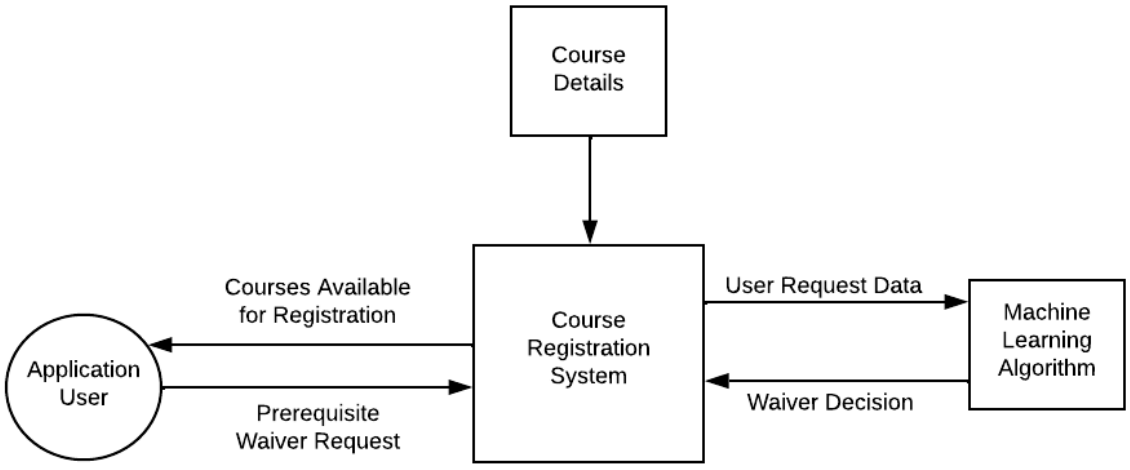

Fig 1 – Flowchart of the proposed system

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 - The first step in the preprocessing pipeline involves transforming the text's characters into lower case letters and removing unnecessary special characters, such as punctuation. Through text normalization, the non-standard words in a document can be converted into a standard format, such as in a string or plain text format. During normalization all the text is converted to lower case letters and any special characters are removed in the text. For example: "Hi man, How are You?" will get converted to - "hi man how are you (Normalized text)"

ii) Tokenization - In tokenization, the normalized text produced in the first step is split into a list of tokens. Basic tokenization refers to strings being separated into simple processing units, to be consumed by various NLP algorithms. In addition, these tokens can be further interpreted and combined to form high-level conceptual units. Three high-level steps are involved in the tokenization process [1]. First, the text document is converted into a series of word counts which equals a predefined bag of word (BOW) unit. The bag of words method refers to the process of identifying the important topics in a text document by creating the tokens of a paragraph. Second, the text is cleaned and filtered through the removal of empty sequences. This involves multiple filtering steps such as collapsing whitespaces and stripping unnecessary special control characters. Third, the filtered text document is sectioned into a series, or list, of features, also known as tokens or words.
For example - Text: Three high level steps are involved in the tokenization process.
BOW:{ Three: 1 high :1 level: 1 steps: 1 are: 1 involved: 1 in: 1 tokenization:1 process: 1}
Cleaned text: {Three high level steps are involved in the tokenization process.
Tokens: {'Three', 'high', 'level', 'steps', 'are', 'involved', 'in', 'the', 'tokenization', 'process}

iii) Stop-Word Removal - Stop-word removal refers to the elimination of frequently used words in text which have minimal information associated with it. Stop-words should be removed if they are not essential for the underlying meaning of the document as they can inhibit analysis conducted by NLP algorithms, especially in regards to information retrieval (IR) tasks. A variety of stop-word identification and removal techniques have been explored and analysed, and can be classified into static and dynamic approaches [2]. For example; "there are three high level steps are involved in the tokenization process" will get converted to "three high level steps involved tokenization process"

iv) Stemming - Stemming is the process of getting to the root of a word and removing prefixes and suffixes from their features. When text has different forms of features being stemmed into one feature, the stemming process is taken advantage of so that the performance of the NLP algorithm's classifier can be improved.
For example; "running races is funnier than running" will get converted to "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 how many times a word occurs 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. But 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) - It considers both a word's frequency inside a text and its frequency throughout the whole corpus. Compared to vectors generated by count vectorizers, they yield vectors that are less sparse and more informative [4]. It produces less sparse and more informative vectors than count vectorizers by 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. Also, 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 - 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 better outcomes [6]. In terms of benefits and performance, this method is comparable

to Word2vec, but may be more effective to train. Hyperparameter tuning is more challenging than with Word2vec. Memory costs are also higher due to the usage of a co-occurrence matrix and global information.

v) BERT - It 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. The definition of it is 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. The number of points at which the corresponding symbols differ is its definition. 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 [12] in the paper "Attention Is All You Need" by Vaswani et al. 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 pre-trained 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 and 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 – We have used 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 – In order for unstructured text data to be analysed, it must be cleaned and transformed. The preprocessing here included extracting text from the pdf file uploaded by the student, removing the stop words and lemmatizing the text. For extracting text from the pdf, we used the ‘Pypdf’ python library. Functions using the ‘NLTK’ (natural language toolkit) were used for removing stop words and lemmatization.

ii) Implementing Word2Vec – Word2Vec are a family of neural networks 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. For implementing Word2Vec we have used the ‘Gensim’ library in python and the pretrained ‘glove-wiki-gigaword-300’ model. The model has been trained on 5.6 billion tokens and over 400,000 unique words from Wikipedia 2014 and Gigaword dataset. After vectorization we used cosine similarity and soft-cosine similarity in our experiment to get the similarity between the two texts. Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors are arrays containing the word counts of two documents. A soft cosine similarity between two vectors considers the semantic similarities between pairs of features too. We used the ‘Scipy’ python library for implementing the cosine

similarity and the ‘Gensim' library for aiding the implementation of the Soft-cosine similarity.

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. We used the ‘Scikit-Learn’ library for implementing TF-IDF vectorizer and then used cosine similarity 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 of a student gets approved, he/she will be able to view the updated list of courses he/she can take. We created 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. MySQL was used for building this mock dataset.

IV. Building a web application – We built 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. 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.

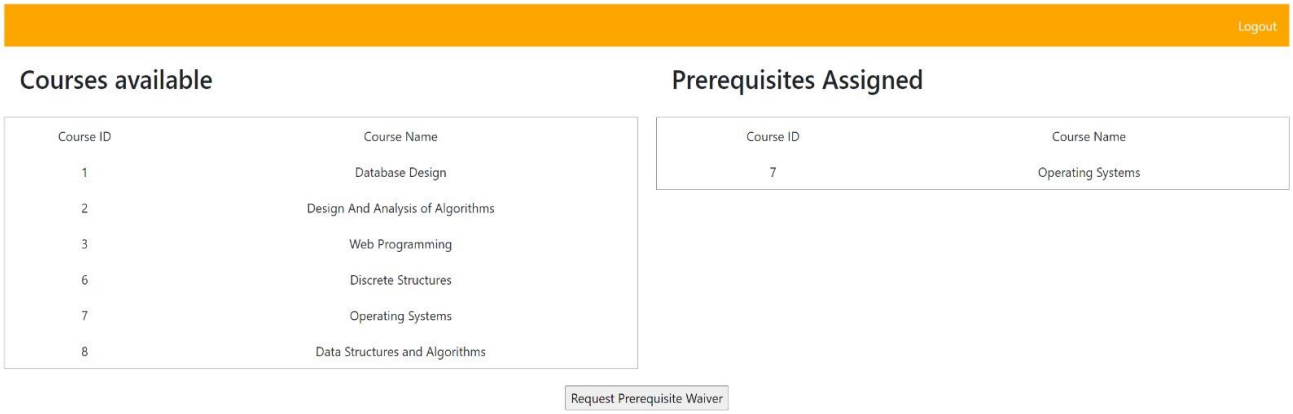

Fig 1 - Screenshot of the home page of the prototype application

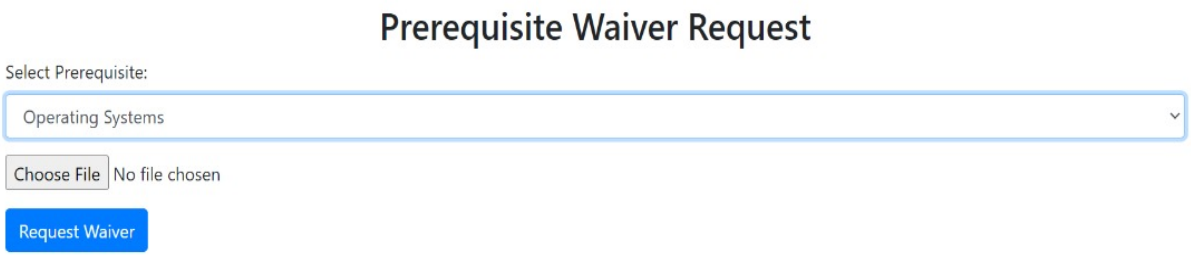

Fig 2 - Screenshot of the waiver page of the prototype application

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 co-occurrence probabilities. For each word, the frequency distribution of words that occur near them is encoded. [13]

IV. TF-IDF - Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure which evaluates the importance or relevance 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 the 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

We collected a dataset of 35 university syllabi related to the courses Operating Systems, Discrete Maths and Data Structures & Algorithms. Each syllabus in the dataset was compared to the corresponding UTD syllabus for that course. We labeled each syllabus 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. We analyzed the performance of the three text matching methods (Word2Vec with Cosine Similarity, Word2Vec with Soft-cosine Similarity, TF-IDF with Cosine Similarity) 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 we tested the mentioned three text matching techniques on the test dataset, we obtained the following results.

Text Matching Method	Accuracy
Word2Vec with Cosine Similarity	74.3%
Word2Vec with Soft-cosine Similarity	85.7%
TF-IDF with Cosine Similarity	80%

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

Our experimental investigation shows that all three text matching methods performed relatively well in accurately classifying syllabi. The text matching method which achieved the highest accuracy was Word2Vec with Soft-cosine Similarity. When considering the three methods relatively, Word2Vec with Cosine Similarity performed the worst. As we got the best results with Word2Vec with Soft-cosine similarity, we used that method in our prototype web application.

FUTURE SCOPE

There are multiple aspects of this project which can be improved like:

I. Authentication of the file provided by the student – In this project we have not touched upon the authentication process which needs to be followed to make sure a student is uploading an unedited version of the syllabus of a course he/she has completed earlier. A possible solution includes getting a link from the student to his/her university course catalogue and downloading the related file directly from the university website. Also, the course completion must be verified from the students' transcripts.

II. More effective pdf parsing – A more effective pdf parser can be built which extracts only the data related to the course description and learning outcomes from the file uploaded by the student and ignores irrelevant details like grading structure and deadlines based on the headings.

III. Building a better test dataset – We can try to access the official prerequisite waiver data of any university to build a better test dataset.

IV. Improving the accuracy of text matching – Better models for converting text into embeddings and transformers can be used to try to improve the accuracy of text matching.

CONCLUSION – In this project we have created a prototype of an automated prerequisite waiver system which leverages text matching machine learning models. We have implemented and compared the performance of three text similarity matching methods: Word2Vec with Cosine Similarity, Word2Vec with Soft-cosine Similarity, and TF-IDF with Cosine Similarity. Upon conducting an investigation using a dataset of university syllabi related to the courses Operating Systems, Discrete Maths and Data Structures & Algorithms, we determined that the Word2Vec with Soft-cosine Similarity method achieved the highest accuracy. This text matching method was used in our prototype to determine whether a syllabus a student has uploaded is sufficient to grant the student a prerequisite waiver. In order to improve our prototype, we can take various actions such as authenticating the file provided by the student, building a more comprehensive and accurate test dataset, and utilizing a more accurate text matching model.

CONTRIBUTIONS - Varad implemented the text matching methods and conducted the experimental investigation. Nisha helped construct/clean the dataset and connect the frontend to the text matching code via Flask. Atharva implemented the application UI and created a MySQL database to store university course and student information. Jaya helped implement the application UI. Tejas helped construct/clean the dataset. All group members helped write the project report.

GITHUB URL: https://github.com/varadabhyankar/Automated-Prerequisite-Waiver-Project

DEMO VIDEO: Included in the GitHub repository.

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 Frequency-Inverse Document Frequency.” *Learn Data Science - Tutorials, Books, Courses, and More*.