Natural Language Processing & Information Retrieval

Kwaku Ofosu-Tuffour

Spring 2024

Abstract

This paper serves as an introduction to Natural Language Processing (NLP), an important and widely used concept in Machine Learning. Background information on Natural Language Processing begins the paper, including discussions on commonly used programming languages/frameworks and commonly used technical terms. Afterwards, the paper introduces the concept of Information Retrieval, a widely used application of Natural Language Processing. Lastly, the paper includes an active application of information introduced within the paper in a mobile application currently in development. In short, the paper serves as a personal investigation on how to apply Machine Learning and mathematics to solve a real-world problem within technology.

1 Introduction

Modern technology continues to be fueled and powered by developments in Artificial Intelligence (AI). Companies and organizations worldwide are rushing to incorporate Artificial Intelligence and Machine Learning into their solutions and services. With the term Artificial Intelligence, important terms such as *Machine Learning* (ML) and *Deep Learning* and often used almost interchangeably. There is, however, an important distinction between these three terms.

The relationship between these three terms can be visualized with a Venn Diagram. In essence, Artificial Intelligence entails programming machines to perform jobs that mimic human behavior. Machine Learning involves the process of machines improving at various tasks without explicit programming. Lastly, Deep Learning involves machines that incorporate artificial neural networks, systems that mirror the human brain, to solve complex problems. Generally, Artificial Intelligence encompasses both Machine Learning and Deep Learning, but is also composed of other important scientific aspects.

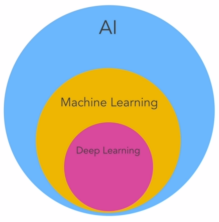

Figure 1: Venn Diagram of Artificial Intelligence, Machine Learning, and Deep Learning [Lab20]

An important topic in Machine Learning is Natural Language Processing (NLP), a field of study that gives computers the ability to interpret, manipulate, and comprehend natural (human) language. Natural Language Processing can ultimately be viewed as an interdisciplinary study between computer science and linguistics.

2 Important Tools

There are a plethora of tools that are used perform key functionalities within Natural Language Processing and Machine Learning. While there are numerous programming languages that can be used for machine learning, the top choice and most common amongst companies, organizations, and educational institutions is *Python*. This choice is a result of Python's simplicity, versatility, and it's extensive ecosystem of important libraries and frameworks used to optimize mathematical computation [W3S].

NumPy, short for Numerical Python, is a library within the Python programming language, is a foundational package for scientific computation. NumPy was first created by Travis Oliphant in 2005 as an open-source project. This well-established library allows its users to perform various important mathematical operations on an n-dimensional array object. NumPy is commonly used within the domains of linear algebra, Fourier transformations, and matrices. NumPy was built for computational optimization. Mathematical operations using NumPy arrays, *ndarray*, performs up to 50x faster than traditional Python lists [W3S].

Tensors are algebraic objects that describe multi-linear relationships between objects within a vector space. In other words, Tensors are mathematical objects used to describe physical properties []. Tensors are used to generalize the concept of scalars, vectors and matrices to higher dimensions. These characteristics are generalized through the rank $\mathbf{R}$ of tensors, where $\mathbf{R} = 0$ represents a scalar, $\mathbf{R} = 1$ represents a vector, and $\mathbf{R} = 2$ represents a matrix. Tensors are an important tool for ML frameworks, as they provide the ability to run mathematical operations on Graphics Processing Units (GPUs) for faster computation [oIftPoMS].

Pandas a python library that provides powerful, flexible open-source data analytics and manipulation tool. Pandas, the name which references both "Panel Data" and "Python Data Analysis", was created by Wes McKinney in 2008. Pandas is used to analyze big data and form conclusions on analysis. Pandas can be used to clean and organize messy data sets, giving its users tools to make data more readable [W3S].

3 Important Terms

Developers use Natural Language Processing to develop algorithms for machines to learn, comprehend, and produce natural (human) language. The challenge, however, is allowing computers that act upon numerical data, specifically binary numbers, to process human

text. For several NLP procedures, the goal is to represent units in natural language like words, phrases, and even paragraphs, as mathematical vectors within a multi-dimensional vector space, which a computer processes and analyzes. In Natural Language processing, *preprocessing* is a necessary step which allows a machine to clean and extract important information from natural language, and further to transform natural language into numerical data. Dan Jurafsky and James H. Martin of Stanford University provide the following overview of prepossessing (and many other features of Natural Language Processing) in their textbook *Speech and Language Processing* [DJ22]

Tokenization is a preprocessing procedure, one of the preliminary steps for transforming text to mathematical vectors. Tokenization simply involves splitting input text into individual *tokens*. Here is an example of tokenization:

Input: "I study mathematics at W&M"
Output: ["I", "study", "mathematics", "at", "W&M"]

When preprocessing natural language, developers commonly wish to extract and remove *Stop Words* from input text. Stop words are words that are not considered during preprocessing due to lack of semantic significance. An example of stop words includes propositions such as a, of, on, I, for, and with.

Stemming and *Lemmatization* are also important concepts in Natural Language preprocessing. Stemming simply involves removing word endings. Lemmatization is the process of reducing a word to its root form, also known as its *lemma*. Here is an example of preprocessing that incorporates both stemming and lemmatization:

Input: The boy's cars are different colors
Output: The boy car be differ color

3.1 Normalization

Normalization can play a crucial role in Natural Language Processing, as Normalization is used when developers wish to confine a dataset that is skewed in behavior. As explained by the Google Developer platform, there are numerous types of normalization techniques including scaling, clipping, log scaling, and z-score [fD22b].

Scaling is used to convert data from their initial range $[a, b], a, b \in \mathbb{R}$ into a standard range like $[0, 1]$ or [-1,1]. Scaling can be used when the input data follows a uniform distribution and the approximate upper and lower bounds of the initial data is known. The following formula can be used for all points x within a data set, with x_{min} and x_{max} representing the minimum and maximum of the data, respectively.

$$x' = \frac{x - x_{mix}}{x_{max} - x_{min}}$$

Log Scaling is used when a data set mirrors a logarithmic distribution, or when the majority of data points are confined within a small portion of the initial range. Log scaling ultimately changes the distribution, creating a greater distinction between data and improving the linear model performance.

$$x' = log(x)$$

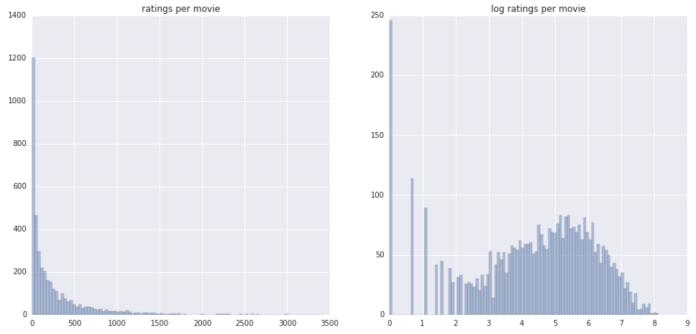

Figure 2: Before-and-after of normalization of data for movie ratings using log scaling [fD22b]

Z-Score normalization represents the number of standard deviations away from the mean. Z-score can be used for confining the range of a data set when *no* extreme outliers are present. Z-Score normalization uses both the mean and standard deviation of a data set.

$$x' = \frac{x - \mu}{\sigma}$$

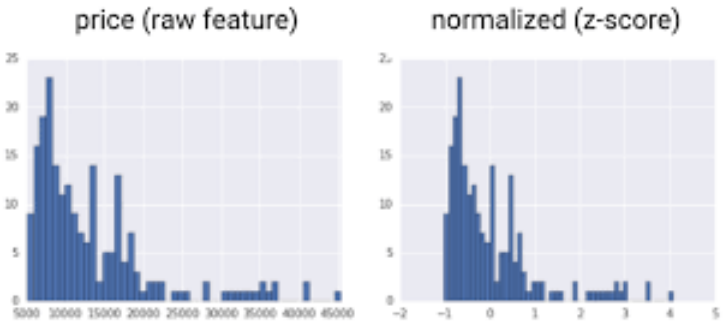

Figure 3: Before-and-after of normalization using z-score [fD22b]

Feature Clipping simply involves confining the range of a data set, removing any extreme outliers that lie beyond this confined range. This normalization process can be used before or after other normalization techniques.

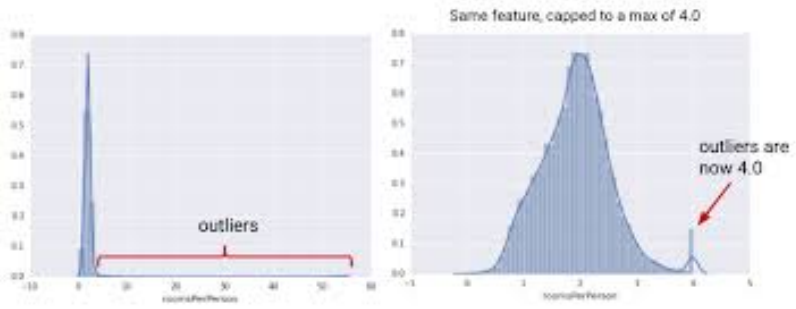

Figure 4: Before-and-after of normalization using clipping [fD22b]

4 Vector Semantics

Natural language is often transformed into mathematical vectors after a preprocessing phase, a process known as *embedding*. The term "embedding" comes from the aspect of embedding words into a vector space. Relationships between words, phrases, sentences, and other units of natural language are made and understood by computers using the relationships between their corresponding vector embeddings. The following information on vector semantics also references the textbook *Speech and Language Processing* by Dan Jurafsky and James H. Martin of Stanford University [DJ22].

4.1 Cosine Similarity

Word Similarity is a measure of semantic similarity between a given set of words through their vector embeddings. Words (vectors) within a vector space that are similar in vector composition tend to have similar semantic meaning. With the following example. it's intuitive to think that words (a) and (c), embedded as vectors, are the most similar in meaning, even without knowing what the words are in natural language.

$$a)\ [18, 27, 41]$$
$$b)\ [-4, -1, 30]$$
$$c)\ [16, 25, 45]$$

Cosine Similarity is one common method of computing semantic similarity between words embedded as vectors. Recall that for vectors x and y, the dot product can be computed using the magnitude of both vectors alongside the cosine of their adjacent angle. Additionally, the dot product of x and y can be viewed as the sum of the products of each individual unit within both vectors. Both formulas for the dot product can be used to derive the cosine similarity formula.

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} (x_i)(y_i) = |x||y| \cdot cos(\theta)$$

$$cos(\mathbf{x}, \mathbf{y}) = cos(\theta) = \frac{\sum_{i=1}^{n}(x_i)(y_i)}{|x||y|} = \frac{\mathbf{x}}{|x|} \cdot \frac{\mathbf{y}}{|y|}$$

The adjacency of words embedded as vectors within a vector space can be viewed by the cosine of their adjacent angle. Any two words $\mathbf{x}, \mathbf{y}$ are more similar as $cos(\mathbf{x}, \mathbf{y})$ approaches 1, implying the two words are within close proximity in the vector space.

The following example demonstrates how cosine similarity can be used to find semantic similarity between two words, given their surrounding context.

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$cos(cherry, information) = 0.017$$
$$\text{cos(digital, information)} = 0.996$$

5 Information Retrieval

Information Retrieval is one of the many use cases of Natural Language Processing. The following information on Information Retrieval also references the textbook *Speech and Language Processing* by Dan Jurafsky and James H. Martin of Stanford University [DJ22].

To preface, here are key terms that are important in understanding Information Retrieval. A *document* denotes any unit of text that a system indexes and retrieves. Documents could be words, sentences, paragraphs, articles, or any other unit of text. A *corpus* or *collection* is a set of documents. A *term* is a word within a collection. Finally, a *query* is the user's information need expressed as a set of terms. In other words, user's query involves characteristics the user hopes to find within a set of documents. Nevertheless, Information Retrieval is the process of scoring various documents in terms of relevancy to a user's query. An information retrieval algorithm wishes to rank all available documents within a collection based on the user's query from most relevant to least relevant. This procedure is commonly used in the development of search engines.

5.1 Term Frequency - Inverse Document Frequency

One important concept used in elementary information retrieval algorithms is Term Frequency-Inverse Document Frequency (TF-IDF). This concept is a statistical measure used to evaluate the importance of a word in a document, which is part of a corpus [DJ22].

Term Frequency simply measures how frequent a word appears in a document. With this measurement, all terms are given the same weight or importance.

$$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$$

Inverse Document Frequency (IDF) measures how important a term is within a set of documents. Terms that are common across multiple documents are given a lower IDF score.

$$IDF(t) = log(\frac{\text{Total number of documents}}{\text{Number of documents with term t}})$$

Altogether, TF-IDF is found by finding the product of the Term Frequency and Inverse Document Frequency of a term t within a corpus.

$$\text{TF-IDF}(t) = TF(t) \cdot IDF(t)$$

5.2 Ad-hoc Information Retrieval

Ad-hoc is a formal name used to reference the task of returning information resources related to a user query formulated in natural language [DJ22]. Here is one general approach to the ad-hoc information retrieval algorithm:

- Use a vector-space model to embed queries and documents as vectors
- Find cosine similarity between the embedded query vector and documents within a corpus
- Rank the documents within the corpus based on the cosine similarity measurements

Consider the following example of the ad-hoc algorithm, which uses TF-IDF as the embedding algorithm. Suppose we have the following information:

Query: sweet love
Document 1: Sweet sweet nurse! Love?
Document 2: Sweet sorrow

First, preprocessing is used to prepare both the query and the documents for vector embedding. For this example, we tokenize the query and documents, perform stemming and lemmatization on all tokens, and create the following set of all tokens:

['sweet', 'nurse', 'love', 'how', 'sorrow', 'is'].

Next, the TF-IDF algorithm is used to embed the query and documents into vectors. Afterwards, the cosine similarity between the query and documents are found. Lastly. the documents are ordered in descending order based on cosine similarity to the query, The following chart from Stanford's *Speech and Language Processing* textbook demonstrates the intermediary steps along with the result of this process.

	Query					
word	**cnt**	**tf**	**df**	**idf**	**tf-idf**	**n'lized** = tf-idf/$\|q\|$
sweet	1	1	3	0.125	0.125	0.383
nurse	0	0	2	0.301	0	0
love	1	1	2	0.301	0.301	0.924
how	0	0	1	0.602	0	0
sorrow	0	0	1	0.602	0	0
is	0	0	1	0.602	0	0
$\|q\| = \sqrt{.125^2 + .301^2} = .326$						

	Document 1					Document 2				
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**× q**	**cnt**	**tf**	**tf-idf**	**n'lized**	**×q**
sweet	2	1.301	0.163	0.357	**0.137**	1	1.000	0.125	0.203	**0.0779**
nurse	1	1.000	0.301	0.661	0	0	0	0	0	0
love	1	1.000	0.301	0.661	**0.610**	0	0	0	0	**0**
how	0	0	0	0	0	0	0	0	0	0
sorrow	0	0	0	0	0	1	1.000	0.602	0.979	0
is	0	0	0	0	0	0	0	0	0	0
$\|d_1\| = \sqrt{.163^2 + .301^2 + .301^2} = .456$						$\|d_2\| = \sqrt{.125^2 + .602^2} = .615$				
Cosine: $\sum$ of column: **0.747**						Cosine: $\sum$ of column: **0.0779**				

Figure 5: Computation of tf-idf and cosine score between the query and nano-documents [DJ22]

5.3 Evaluation of Information Retrieval

Two common metrics for measuring performance of an information retrieval algorithm include *Precision* and *Recall*, explained by Google's Developer Platform [fD22a]. Precision measures what portion of positive identifications are indeed correct. In the context of Information Retrieval, precision measures whether the documents retrieved from the algorithm are truly relevant to the user's query. Recall measures what portion of actual positives were correctly identified. In other words, Recall finds the percent of all relevant documents to the user's query that are returned.

Consider the following chart.

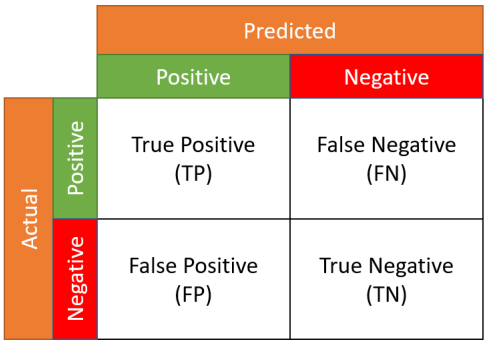

Figure 6: Graph representing the relationship in performance for the predicted and actual performance of a machine learning model

For Information Retrieval, TP (True Positive) represents all documents that were correctly returned from a user's query. TN (True Negative) represents all documents that were cor-

rectly ignored from a user's query. FP (False Positive) represents all irrelevant documents that were returned from a user's query. FN (False Negative) represents all irrelevant documents that were ignored from a user's query. Generally, a machine learning algorithm hopes to match it's predictions data with actual data by having True Positives and True Negatives as often as possible.

Unfortunately, precision and recall often times can oppose each other in performance in the sense that improving precision typically reduces recall and vice versa [fD22a]. For example, an algorithm that fails to return all relevant documents to a query but does correctly identify relevant and irrelevant documents within a subset can bear high precision. Meanwhile, an algorithm that returns a large portion of relevant *and* irrelevant documents can bear high recall.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

5.4 Contextualized Embeddings

While the TF-IDF algorithm does serve as a straightforward way of embedding natural language to vector spaces, it fails to convey some important aspects of semantic meaning. Normal embeddings such as those generated by the TF-IDF algorithm are *static*, meaning that they do not take the true semantic meaning of tokens into account. Consider the following sentences:

"I went to the bank to deposit some money"
"The player made a bank shot in the last quarter"
"They met me at the river bank"

Here, the word bank is used in three separate contexts and has three separate meanings. Thus, this example highlights a crucial problem with static embeddings: they fail to contextualize words. The TF-IDF algorithm would therefore an ineffective embedding algorithm for the given example. A general solution is to use an embedding algorithm that takes context of words into consideration during the embedding process. These are known as *contextualized embeddings*.

One well-known approach for contextualized embedding is the Word2Vec algorithm [MCCD13]. Word2Vec can use two approaches: the Continuous Bag-of-Words model (CBOW) and the Continuous Skip-gram model. Within a sentence, document, or other unit of text, CBOW is used to predict the semantic meaning of a token based on the context of its environment. Skip-gram, on the other hand, predicts the context of a sentence based on a specific token. There are many other models commonly used to generate contextualized embeddings, including ELMo, ULMFiT, ChatGPT, and BERT.

6 Project Implementation: W&M Market Application

Here lies the main inspiration for this paper. The goal of this project is to create a mobile application where W&M students can sell and purchase items from other W&M students. Ideally, these items can range from books, supplies, technology, appliances, and other types of daily use. Many students have expressed a great need for a new medium for shopping for dorm goods while in school. The application is currently being pursued by a small group of William and Mary students, including the author of this paper, under the leadership of *Yera Park*, W&M class of 2025.

Our team wishes to create an application for both Android and iOS. Typically, mobile applications of this complexity are done using a full-stack development stack. The front-end of full-stack applications are responsible for user interaction. In other words, the front-end renders all of the texts, buttons, colors, and other aspects of the screen. Meanwhile, the back-end of full-stack applications are responsible for processing and storing important information that is used in the app. Some examples in the market application include a user's profile information (username, password, email, profile picture), messages, and post information.

Our current development stack includes *React Native* as the front-end mobile application framework and *Django* as the back-end framework. React Native, a JavaScript/TypeScript based front-end framework, allows developers to create a cross-platform mobile applications. Django, a Python-based backend framework, is used to create a platform of communication between the user from the front-end and the data stored in our database. This communication is specifically done using Django's REST API framework, where *REST API* frameworks are procedures that two computer systems use to distribute information between one another. For public use, the team plans to deploy the React Native front-end on both the App Store and the Google Play Store. Furthermore, the team plans to host the Django backend using Amazon Web Services (AWS), using features like Amazon EC2 for running the Django application, Amazon RDS to create and use a Postgres database, and Amazon ROute 53 to obtain a domain to reference Django's REST API.

Students can create and navigate *posts*, objects which describes items that students are selling, on the app's home page. Information shown in user posts include a product name, a product description, a display image, and the username of the user selling the product. One crucial component of the app is its searching feature in the home page. Here, students should be able to search through posts for their specific needs using keywords. For example, when a user has a search of "Tennis", the app should display posts that are relevant to tennis. Prior to using Natural Language Processing, the app only used tokenization and returned documents (posts) based on exact matches to the tokens within the query. This process, however, was insufficient for multiple reasons. Posts that did not contain any of the exact tokens in the tokenized query were not returned in the former search algorithm. Not only did this implementation ignore semantic similarity, but a post with the title "pencil" would not be returned with a query of "pencils". Thus, the performance of our previous implementation was considerably less than subpar.

The research presented within this paper helped the team align with a better approach to our search engine. To implement a more performative algorithm, the team uses ad hoc Information Retrieval, incorporating contextualized embeddings and cosine similarity to rank posts based on user queries. The application currently references a open source machine learning model on *HuggingFace*, a machine learning and data science platform and community that helps users build, deploy, and train machine learning models. Specifically, the Django backend uses the *Sentence Transformers* model [RG19].

6.1 Current Search Engine Performance

The following python code snippet embodies the ad hoc Information Retrieval implementation for our search engine.

```
api_url = [URL]
headers = [HEADERS]

def get_embeddings(documents):
    """
    Transforms the each document in the list into an embedded vector
    """
    response = requests.post(api_url, headers=headers,
    json={"inputs": documents, "options":{"wait_for_model":True}})
    return response.json()

def rank_similarity(input):
    """
    Ranks the similarity between the input vector and the document vectors
    using cosine similarity.  In other words, it ranks the posts based on
    the user's search input
    """
    posts = Post.objects.all()
    documents = [post.product for post in posts]
    documents.insert(0, input)
    documents_ids = [post.id for post in posts]

    embeddings = get_embeddings(documents)
    query = embeddings.pop(0)
    documents.pop(0)

    return similarity(query, embeddings, documents, documents_ids)

def similarity(query, embeddings, documents, documents_ids):
```

```
    """
    Function that calculates the cosine similarity between the query and
    document vectors.  Return a list of posts in descending order by
    their search ranking
    """
    data = []
    for vector in embeddings:
        dot_product = np.dot(query, vector)
        query_magnitude = np.linalg.norm(query)
        vector_magnitude = np.linalg.norm(vector)
        data.append((dot_product) / (query_magnitude * vector_magnitude))
    df = pd.DataFrame({
        "Post_id": documents_ids,
        "Name": documents,
        "Score": data
    })
    df = df.sort_values(by=['Score'], axis=0, ascending=False)
    return list(df['Post_id'])
```

Using a dataset of approximately 50 posts, the following data represents the results of performing various searches within the application.

Query: "Bottle"

Post_id	Name	Score
23	Shaker Bottle	0.72448
35	Black water bottle	0.717623
45	Purple water bottle	0.63708
34	W&M water bottle	0.604581
1	Cups	0.457705
48	V8 Juice	0.413679
40	Helmet	0.380746
6	Bird	0.37847
15	Spicy Sauce	0.371038
37	TV	0.369128

Precision: $(5)/(5+1) \approx 0.833$
Recall: $(5)/(5+0) = 1$

Query: "Sports"

Post_id	Name	Score
37	TV	0.509026
12	Tennis Balls	0.465537
40	Helmet	0.456399
7	Purple Soccer Ball	0.447629
8	Soccer sweatshirt	0.414176
1	Cups	0.386867
6	Bird	0.355266
5	Dog	0.306227
38	Road Bike	0.277349
33	Boots	0.276298

Precision: $(5)/(5+0) = 1$
Recall: $(5)/(5+0) = 1$

Query: "Pencils"

Post_id	Name	Score
18	BIC Mechanical Pencils	0.687582
1	Cups	0.365652
22	Five Star Paper	0.357549
36	Printer	0.347954
19	Mini Stapler	0.345633
25	Coat	0.324235
12	Tennis Balls	0.302151
46	Coffee creamer	0.286429
37	TV	0.272269
33	Boots	0.271016

Precision: $(1)/(1+0) = 1$
Recall: $(1)/(1+1) = 0.5$

Query: "Instrument"

Post_id	Name	Score
13	Oboe	0.688479
50	Oboe Reed	0.646337
24	Handheld vacuum	0.337073
12	Tennis Balls	0.320299
37	TV	0.315224
40	Helmet	0.30879
19	Mini Stapler	0.308068
18	BIC Mechanical Pencils	0.305089
6	Bird	0.281814
5	Dog	0.263972

Precision: $(2)/(2+0) = 1$
Recall: $(2)/(2+0) = 1$

Query: "School"

Post_id	Name	Score
37	TV	0.481606
5	Dog	0.434241
6	Bird	0.353587
1	Cups	0.352532
40	Helmet	0.334864
36	Printer	0.3288
25	Coat	0.313415
46	Coffee creamer	0.283103
33	Boots	0.27802
38	Road Bike	0.270293

Precision: 0
Recall: 0.0833

In general, the current implementation of our searching algorithm is a solid stepping stone towards having an ideal search engine for the market application. That being said, there's definitely still more room for improvement in performance. There are now a few actions in consideration for the team to consider. First, the team would like to test the current search algorithm with a larger data set–one with thousands of posts to query from. Testing on a larger data set would provide a greater indication on the performance on the current implementation, as well as expose any problems with optimization and efficiency. Second, the team is considering developing our own contextualized embedding model. Because the team references an open source model for embeddings, the team has no power on adjusting parameters to improve the results of the query. The *Sentence Transformers* model currently seems to struggle to find similarities between queries and brand names, the latest releases

in technology (i.e. Airpods), and natural language used by the younger generation. In developing our own contextualized embedding model, our team can perform training that would produce better searching results.

7 Conclusions

This paper presents one of the many applications of Natural Language Processing, Machine Learning, and Artificial Intelligence. Development in these fields improve the user experience within technology. As technological advancements continue to grow, especially in Artificial Intelligence, many of the daily activities humans perform both with and without technology will likely improve in efficiency, reliability, and usability. Developing efficient search engines proves to be essential for many forms of media and with the internet, seeing as millions of users worldwide perform search requests for their own personal needs. It's interesting to note that such a complex and essential feature of the internet and social media fundamentally boils down to vector mathematics. Like many other subjects, there seems to be a natural and almost inevitable connection to mathematics.

References

[DJ22] James Martin Dan Jurafsky. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Stanford Universitu, 2022.

[fD22a] Google for Developers. Classification: Precision and recall. 2022.

[fD22b] Google for Developers. Normalization. 2022.

[Lab20] Lotus Labs. Clarifying ai, machine learning, deep learning, data science with venn diagrams, 2020.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[oIftPoMS] Dissemination of IT for the Promotion of Materials Science.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[W3S] W3Schools.