FINDTHATQUOTE: A QUESTION-ANSWERING WEB-BASED SYSTEM TO LOCATE QUOTES USING DEEP LEARNING AND NATURAL-LANGUAGE PROCESSING

Nathan Ji[1] and Yu Sun[2]

[1]Portola High School, Irvine, CA, 92618
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

The digital age gives us access to a multitude of both information and mediums in which we can interpret information. A majority of the time, many people find interpreting such information difficult as the medium may not be as user friendly as possible. This project has examined the inquiry of how one can identify specific information in a given text based on a question. This inquiry is intended to streamline one's ability to determine the relevance of a given text relative to his objective. The project has an overall 80% success rate given 10 articles with three questions asked per article. This success rate indicates that this project is likely applicable to those who are asking for content level questions within an article.

KEYWORDS

Deep learning, question-answer engine, natural-language processing.

1. INTRODUCTION

The main topic concerning our project is Natural Language Processing (NLP) [1] [15]. NLP is defined as the relationship between computers and human language, but more specifically, in our case, NLP is how a computer interprets human language in textual form. We utilize NLP within the context of the answer engine by allowing the program to interpret the sentence in the text and compare that information to the information within the question.

Natural Language Processing is an extremely important field of computer science because in order for us to be able to make any progression in AI development, we must first be able to understand language. An AI's understanding of linguistics is an important requirement in being able to fully automate the workforce and will most likely be one of the final steps necessary to create fully functional and lifelike AI [2].

Natural Language Processing as well as Question Answer Engines [3] are the staple of future AI and deep learning programs [4] as they are the most vital algorithms that are going to be required for successful integration of AI. If an AI is unable to interpret natural linguistics, then any hope of being able to act more autonomously based on a user's command would be impossible. The entire premise of AI is to make the code more condense and make the actions more broad, which can only be done if the AI can successfully interpret what needs to be done

David C. Wyld et al. (Eds): ITCSE, ICDIPV, NC, CBIoT, CAIML, CRYPIS, ICAIT, NLCA - 2021
pp. 97-104, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110909

without the information being hard coded within its software. Question Answer algorithms are vital tests for deep learning and AI programs to test retention as well as the ability to process linguistics. Our ability to determine the successfulness of software without the need to deploy it is also equally important, and thus it is important that Question Answer Engines be developed to test the effectiveness of strategies that will be incorporated in deep learning and AI programs.

Currently, there are a few approaches for combining NLP and question and answer algorithms. So far, the most successful algorithm is a deep learning [5] or AI approach. A good example of how successful this algorithm is the IBM Watson [6] [14], an advanced deep learning question and answer engine that also uses NLP to interpret its inputs. IBM Watson is the modern approach to this problem; it uses a deep learning software that is more accurate, memory efficient, and resistant to time complexity issues. This methodology is most likely the ideal approach to question and answering problems that require NLP, as it is much more efficient and also less naive.

The other approaches used are built upon hardcoding the data derived from deep learning algorithms and the storing of such information in a library--a method that therefore suffers from even greater time and memory issues.

There are multiple libraries, each with their own strengths and weaknesses. Word2Vec [7], the vectors developed by Google, are objectively the most reliable vector set. By combining this with a language comparison model like Gensim [8], its combination of NLP and question and answer can effectively return correct answers a majority of the time.

Another option used is Spacy [9], which was developed by Stanford. The approach using Spacy is to match questions with possible answers using its NLP capability. A massive flaw this approach has is that the question and answer part of this approach is particularly weak due to Spacy's lack of applicable data that can be derived using its software. Compared to the method above, it is less effective and also less accurate. NLTK--Python's Natural Language Toolkit--is a library that is essentially a staple built-in NLP processor used by Python [10]. Much like with Spacy, there is a lack of robust question and answer capability. However, NLTK uses vectors instead of comparison, making it perhaps a little more accurate, but the effects are almost negligible.

My project uses a vector system provided by Google, which is then used by a package called Genism that has a library full of possible English structures. This approach is a very brute force approach, which requires us to hard code all the possibilities, meaning packages like these are extremely large and rather ineffective for commercial use. Although our project works, it doesn't have a decent time complexity nor is it memory.

Initially, this project used Spacy in a naive approach to first interpret the question and the text and then compare the two results in another naive approach to question and answer. This yielded around a 60% accuracy rate, which, compared to the current 80, is much less. This can mostly be attributed to the amount of corner cases Spacy could not accurately interpret and compare, making the more broad option of using vectors and Gensim far better than Spacy. NLTK also would derive similar results to Spacy, making Gensim and Word2vec the best non-AI approach.

Additionally, it should be noted that a deep learning approach that doesn't need to derive its information from vast libraries would easily trump the success of all three naïve approaches. Comparatively, an AI algorithm makes the most sense in this scenario given the inherent complexity involved.

In testing our algorithms, there were ten articles with three separate questions, each meant to derive a specific part of text within the article. Each time, we would give the algorithm a question and an article, and it would then return a sentence. If the sentence matched the sentence that was interpreted to be correct, the algorithm would have been successful. The algorithm was successful in doing this around 80% of the time.

2. CHALLENGES

In order to create an application that can streamline one's ability to determine the relevance of a given text relative to his objective, a few challenges have been identified as follows.

2.1. Challenge 1:

When I first approached this problem, I initially used SVOs to match the sentences. However, this approach returned a multitude of errors when I encountered several special cases. Some sentences are structured in OVS, SVVO, and SVOO formats, which make them difficult to read. Furthermore, some sentences had multiple SVOs in them in convoluted patterns like SSVOO or SVVOO. There were too many deviations from the original SVO that the overall accuracy was rather low. Additionally, I was ignoring the prepositional phrases, which further decreased the accuracy. Ultimately, I switched to using Gensim, whereby I changed the approach, and rather than highlighting key words, I used all of the words to find the best match.

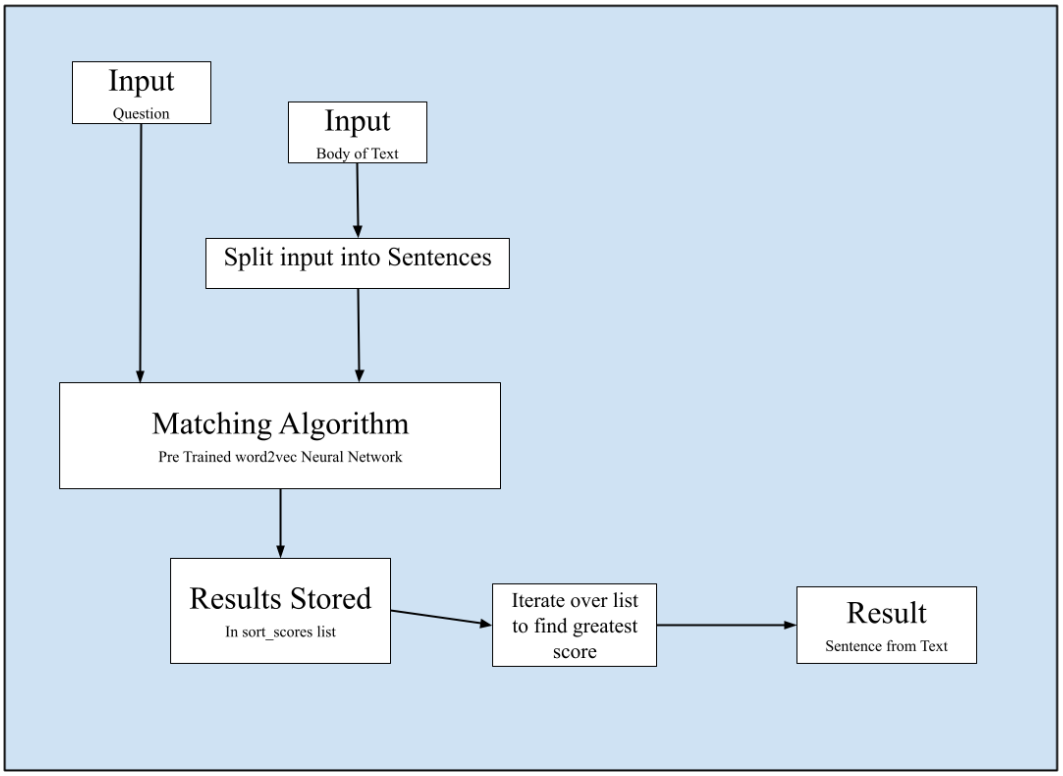
2.2. Challenge 2:

After I figured that a model for matching is the best way to go about answering my goal questions, I needed to use either a vector model or a NLP model to match the words. However, each matching algorithm was extremely different, and all were equally convoluted. Matching singular words didn’t work because the context that each model each word was in was different, which ultimately made it impossible for me to only match one word at a time, meaning that it was more difficult to find a model that didn’t define each word individually. Eventually, I found Gensim through trial and error and ended up using that.

2.3. Challenge 3:

The 3rd challenge I faced was a natural consequence of challenge 2: the inability to match words when using the model. After determining the best model, there was a lot of difficulty using that model to actually successfully match the words. The inputs for the words were difficult because splitting sentences was extremely difficult. Using Gensim solved this problem because we were able to match entire sentences with vectors in order to establish context. Overall, Gensim proved the best option, as it matched entire sentences.

3. SOLUTION



The main goal of this solution is to match the question given with each sentence within the test to find the answer to the question. When working to achieve this goal, the first step is splitting the sentences within the test using Spacy's sentence splitter. This is important because I couldn't split sentences based on periods because of instances with a title like "Mr." or an abbreviation. Spacy does this by reading through the sentence and establishing the context in which a period is found. If the period is found within a title, which involves a proper noun, it would skip over it. Spacy does this for a multitude of cases, which ultimately divides the sentences correctly. Step two is to input each sentence with the question to my spacyMatching file, which would naturally lead to execution of step three. Step three involves first initializing vectors so that Gensim has all the necessary parameters. We do this by downloading the vectors from the IDE, which then gives us a list of vectors, which we can't interpret but Gensim can. These vectors are a specific list of a series of numbers that define a word based on context, which helps Gensim piece together the meaning of the sentence. These numbers are different for different vector lists, but Gensim can read Google's list accurately. Step four is then to pass each sentence, question, and vector to Gensim, which then returns a number based on how closely these match one another. The greater the number, the greater the similarity. This occurs by comparing the various vectors in the sentence as well as the position of those vectors to achieve a number from 0 to 1, which we can then interpret as similarity. Step five is to rank these numbers with the relevant sentence and then return the requested number of top sentences in the order of similarity.

I used the following imports in my code: Math, Gensim, En_core_web_sm (spacy model), Scipy, numpy,

Flask, flask_cors, JSON, and Spacy.

Step 1 is splitting the sentences within the text. This is achieved by a list comprehension done by SPACY. Essentially, we run a for loop for each phrase that SPACY deems a sentence.

```
doc = self.nlp(text)
#splits text into setences
sentences = [sent.string.strip() for sent in doc.sents if len(sent) > 2]
# print(sentences)
```

Step 2 is sending the input sentences to spacyMatching. First, I initialized scores for later use as I needed to rank the sentences in step 5. This also helped me store the return values of step 4, which is the similarity ratings for the sentences. What we pass into the function of spacyMatching is our question, sentences, and a variable called wv, which stands for word vectors [11]. This is all the parameters needed for steps 3 and 4.

```
scores = {}
for sentence in sentences:
    scores[sentence] = spacyMatching.matching(self.nlp(question), self.nlp(sentence), self.wv)
#sort scores
sort_scores = sorted(scores.items(), key=lambda x: x[1], reverse=True)
#check for no match
# if sort_scores[0][1] <= 0.5 or math.isnan(sort_scores[0][1]):
if math.isnan(sort_scores[0][1]):
    return scores, ['there was no match']
else:
    result_list = []
    for i in range(numofanswers):
        # sentence_index = sentences.index(sort_scores[i][0])
        result_list.append(sort_scores[i][0])
```

Step 3 is initializing vectors for the matching algorithm. We do this simply by first interpreting the parameter that was imported in the file within step 2 and step 1, which we then use for the next two lines. The next two lines is where step 4 comes in and matches the two numbers. The next line gives us the actual similarity from the two that we use to match the sentences as the best way to quantify similarity is using an imported cosine function that helps derive the average difference between the two numbers.

```
def matching(a, b, word2vec):
    index2word_set = set(word2vec.wv.index2word)
    s1_afv = avg_feature_vector(a, model=word2vec, num_features=300, index2word_set=index2word_set)
    s2_afv = avg_feature_vector(b, model=word2vec, num_features=300, index2word_set=index2word_set)
    sim = 1 - spatial.distance.cosine(s1_afv, s2_afv)
    return sim
```

Step 4 is the actual matching, which was called in step 3. This step first splits the sentence into words and for each of those words, it first checks if it is our set of vectors (index2word_set). Then, if it is, we pass it through the model and add it to our feature vector variable and if not, we get rid of it. We then return the feature vector back to step 3.

```
def avg_feature_vector(sentence, model, num_features, index2word_set):
    stop_words = ['a', 'an', 'the']
    # words = sentence.split()
    feature_vec = np.zeros((num_features,), dtype='float32')
    n_words = 0
    words = [token.text for token in sentence if not token.is_stop]
    for word in words:
        # if word in index2word_set and not word in stop_words:
        if word in index2word_set:
            n_words += 1
            feature_vec = np.add(feature_vec, model[word])
    if (n_words > 0):
        feature_vec = np.divide(feature_vec, n_words)
    return feature_vec
```

Step 5 is ranking the sentences. This is done by simply first sorting our list of scores using lambda and then making a return list to make it easier to return in a readable form.

```
scores = {}
for sentence in sentences:
    scores[sentence] = spacyMatching.matching(self.nlp(question), self.nlp(sentence), self.wv)
#sort scores
sort_scores = sorted(scores.items(), key=lambda x: x[1], reverse=True)
#check for no match
# if sort_scores[0][1] <= 0.5 or math.isnan(sort_scores[0][1]):
if math.isnan(sort_scores[0][1]):
    return scores, ['there was no match']
else:
    result_list = []
    for i in range(numofanswers):
        # sentence_index = sentences.index(sort_scores[i][0])
        result_list.append(sort_scores[i][0])
```

4. EXPERIMENT

My solution uses a pre-trained neural network that identifies similarity between two different sentences, in order to answer the question given within a piece of text. The first few approaches all utilized subject verb object pairs, only reaching up to a 60% success rate with fewer test cases, while the newer approach managed to reach an 80% success rate with more varied and difficult test cases. This newer version is better because we used a pre-trained neural network that was far more efficient as well as accurate due to the heightened abilities of AI.

The results of my algorithm isolated a certain body of text given a question, hence increasing the speed at which someone can analyze the text by allowing them to ignore a large majority of an article, which they didn't need. Although the success rate was only 80%, many of those 20% of cases were extreme corner cases, which we wouldn’t see people normally asking.

Summary: 30 Test Cases yielded in an 80% success rate.

5. RELATED WORK

Chapter from Capturing Intelligence, "From search engines to question answering systems—the problems of world knowledge, relevance, deduction and precisiation” by Lotfi A. Zadeh.

The first very clear instance of question answering engines in the world was the application of search engines, the most prominent one being Google. However, Google uses much more advanced algorithms to do this, not only using text relevance, but also systematical deduction, and other logical theories. Although the bases of both projects are similar, Google instead matches searches based on a vast network of data rather than a sole question. This allows Google's results to be far more precise and accurate. Google has perfected its advanced Question Answer design, however one flaw is the vast amount of data this engine would require, obtainable by no more than two or three extremely large corporations [12].

"Improving chronological ordering of sentences extracted from multiple newspaper articles" by Naoaki Okazaki, et al.

This research paper explores a different approach to Question Answering Engines using Information retrieval models, which are an alternative to NLP and matching. Since this is an entirely different approach, while it has the same input and output as my algorithm, it uses chronological ordering and probability to derive its results. It is likely that this algorithm is just as effective, or even more effective, than NLP. Indeed, according to the abstract, "ABRIR uses both a word index and a phrase index formed from combinations of two adjacent noun words. The effectiveness of these two methods was confirmed according to the NTCIR-4 Web test collection," demonstrating that they most likely achieved a rather high accuracy result rate [13].

"Extracting Radiotherapy Treatment Details Using Neural Network-Based Natural Language Processing" by D.S. Bitterman, et al.

This research paper explores the application of neural networks as well as natural language processing in extracting data from cancer treatments, specifically radiation therapy. Due to the lack of a robust NLP system that exists for this task, these researchers developed a neural network to extract data from reports and conglomerate them. This research displays that much

like my research, many subjects and documents can be greatly enhanced through the applications of NLP in conjunction with neural networks, where my design helps find quotations for fields like debate and general writing, while their research creates a concise way to analyze radiation therapy data. Comparatively, their research application goes more specific and in depth than mine as they noted that, "neural networks achieved reasonable performance on RT [Radiation Therapy] detail extraction despite the small dataset and the highly specialized language," whereas my research provides a general summary that will be unable to interpret more specialized datasets [16].

"Semantic Convolutional Neural Network model for Safe Business Investment by Using BERT" by Maryman Heidari, Setareh Rafatirad

This research paper explores the applications of neural networks and semantic analysis in the realm of finances in both a response to the 2008 financial crisis and future real estate investment. Researchers used a semantic convolutional neural network to predict rent to offer a safe real estate investment, ideally to better those attempting to find affordable housing. In this instance, these researchers used semantic analysis contrary to my natural language processing, which although greatly related, semantic analysis is more geared to analysis of financial markets with a binary. Additionally, their research pulls from a much larger data pool, as they used a new public data set with over 5 million houses, allowing for their results to be more accurate and precise[17].

6. CONCLUSION AND FUTURE WORK

Essentially, I have created an algorithm that matches sentences and a question by similarity in order to answer the questions provided by the user. This can be comfortably integrated into any person's day who seeks to quickly find the answer within a large article. The effectiveness of this algorithm is around 80%, which makes it usable, but not ideal. Overall, the algorithm works and helps people accurately identify sections of a passage that may contain the answers to their questions, which can help reduce time wasted on reading lengthy articles.

The main problem with the algorithm is its lack of accuracy. A user would expect a very high level of accuracy to consistently use this, which poses a problem for this specific algorithm. From a practical standpoint, the runtime is also fairly long and rather inconsistent. Given that there is no cap to the data that can be currently inputted, it is plausible that a long document would render the algorithm untenable due to its inefficient runtime. Further work could possibly include an even greater level of AI incorporation where I would train a neural network with just the text and sentence desired to increase accuracy.

REFERENCES

[1] Chowdhury, Gobinda G. "Natural language processing." Annual review of information science and technology 37.1 (2003): 51-89.
[2] Nilsson, Nils J. Principles of artificial intelligence. Morgan Kaufmann, 2014.
[3] Harabagiu, Sanda, et al. "Falcon: Boosting knowledge for answer engines." TREC. Vol. 9. 2000.
[4] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.
[5] Goodfellow, Ian, et al. Deep learning. Vol. 1. No. 2. Cambridge: MIT press, 2016.
[6] High, Rob. "The era of cognitive systems: An inside look at IBM Watson and how it works." IBM Corporation, Redbooks 1 (2012): 16.
[7] Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." arXiv preprint arXiv:1402.3722 (2014).
[8] Řehůřek, Radim, and Petr Sojka. "Gensim—statistical semantics in python." Retrieved from genism. org (2011).
[9] Srinivasa-Desikan, Bhargav. Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras. Packt Publishing Ltd, 2018.
[10] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.
[11] Lam, Maximilian. "Word2bits-quantized word vectors." arXiv preprint arXiv:1803.05651 (2018).
[12] Zadeh, Lotfi A. "From search engines to question answering systems—the problems of world knowledge, relevance, deduction and precisiation." Capturing Intelligence. Vol. 1. Elsevier, 2006. 163-210.
[13] Okazaki, Naoaki, Yutaka Matsuo, and Mitsuru Ishizuka. "Improving chronological ordering of sentences extracted from multiple newspaper articles." ACM Transactions on Asian Language Information Processing 4.3 (2005): 321–339.
[14] Lally, Adam, and Paul Fodor. "Natural language processing with prolog in the ibm watson system." The Association for Logic Programming (ALP) Newsletter 9 (2011).
[15] Liddy, Elizabeth D. "Natural language processing." (2001).
[16] Bitterman, D.S. et al, (2020, November 1). Extracting Radiotherapy Treatment Details Using Neural Network-Based Natural Language Processing.
[17] Heidari, M., & Rafatirad, S. (2020, December 14). Semantic Convolutional Neural Network model for Safe Business Investment by Using BERT. IEEE Xplore.