Alexa Discussion Board Skill

Matthew Baker

Abstract

Amazon Alexa offers users an innovative way to verbally interact with a device capable of accessing massive amounts of information. Students may find this to be a preferable means of obtaining information concerning their class instead of browsing through online discussion boards. This project utilizes multiple services offered by Amazon to implement an Alexa skill that maps student questions to data extracted from a class discussion board.

Introduction

Students often have questions about course work and procedures that they are unable to have answered during their lecture time. Options to allow students to have their questions answered outside of class time include instructor office hours, textbooks, and online discussion boards. Office hours, however, are often inconvenient for college students with busy schedules, especially for those who live far away from campus. Textbooks are expensive and often difficult to understand, as the students' questions could be about the textbook content itself. Discussion boards are the most convenient of these options, as they can be accessed remotely and at any time of the day. However, searching through an active discussion board to find an answer to a question can be time consuming. Many students do not even bother with this process and simply post questions that have already been answered, further cluttering the forum. There needs to be a solution to the problem of student questions that strikes a balance between the accessibility of a discussion board and the speed of information retrieval that instructors possess.

The purpose of this project is to achieve this balance by creating an Alexa skill that allows students to ask their Alexa-enabled devices questions about their courses and receive reasonably accurate answers drawn from their class discussion board. This project is implemented as an Alexa skill because of Alexa's increasing popularity and relevance. It would not make sense to create what is meant to be a tool of convenience on a device or platform that is inaccessible to most of its intended users. Alexa-enabled devices, however, are very common, as Amazon figures have claimed that as of January 2019, over 100 million of these devices have been sold [1]. ASU students have a unique access to Alexa, as Amazon and ASU have recently engaged in a partnership wherein residents of Tooker House will be offered Amazon Echo Dots to "[give] them touch-free access to information and services tailored to campus living" [5]. These cases demonstrate that implementing this project as an Alexa skill, meaning a custom-built capability, will make it accessible to many of its intended users.

Research

I began my research with the concept of Question Answering, where software systems accept user questions as input, process them, and return what they believe to be the answer to that question. Question answering is divided into two major paradigms: information-retrieval, which "relies on the vast quantities of textual information on the web or in collections like PubMed", and knowledge-based systems, which create "a semantic representation of the query" before attempting to answer [3]. Due to the source of the skill's data being quantities of textual information in the form of discussion board posts, I chose to focus my investigation on information retrieval.

However, once my research turned towards the implementation details of information retrieval systems, concerns began to rise. One of the first tasks that these systems must do is

identify what type of information the question is asking for. This is commonly accomplished with question classifiers, and although Li and Roth managed to create a classifier that achieved up to 98.8% accuracy, they did so by using rather complex machine learning techniques such as the SNoW learning architecture [4]. Implementation of this system would require computing resources that would test the limits of Lambda, the Amazon Web Service used to implement the back-end of Alexa Skills and Amazon APIs. Lambda has a memory restriction of 50 MB for zipped deployment packages, and as the later implementation of cosine similarity would show, python libraries that are frequently used for classification and linear algebra take up a large amount of space.

Space and complexity concerns encouraged me to scale back my research to the domain of simple document retrieval. My goal was to find a technique that would return the discussion board post that most likely aligned to a given question. This led me to consider sentence similarity as an approach that achieved a balance between accuracy and feasibility, and the most prominent sentence similarity technique that stood out in my research was that of cosine similarity. The principle behind cosine similarity is to represent two documents as vectors and compute the angle θ between the two vectors as follows:

$$\theta = \cos^{-1} \frac{\vec{a} \cdot \vec{b}}{\|a\| \cdot \|b\|}$$

The first task in cosine similarity is to take a document string and divide it into individual words, known as tokens. Once this is done, “stop words” which are words that are extremely common and not very useful in distinguishing documents from one another, will be removed and the remaining words will be lemmatized, meaning that the inflectional endings will be removed, and they will be converted into their basal dictionary forms [2]. Figure 1 shows a sentence string

undergoing tokenization (b), stop word removal (c), and lemmatization (d). Next, all the unique document tokens will be assembled into a single indexed object called the "lexicon". Each document will be a vector whose length will be equal to the length of the lexicon, and the elements of this vector will be the number of occurrences of the token at that index within the document. All these documents together will form a matrix known as the term-frequency or tf-matrix. Words that appear in relatively few documents are given higher importance by multiplying every tf term by its respective idf value. Given a term t, n documents, and x(t) representing the number of documents containing t, idf is calculated as follows:

$$idf = \log\frac{n+1}{x(t)+1} + 1$$

Once every tf value is multiplied by its idf value then we have the tf-idf matrix, and the vectors of the tf-idf matrix can be used in the cosine similarity formula given before to compute the similarity between two documents. Its relative simplicity but undeniable logic due to the tf-idf adjustments led me to choose cosine similarity as the method by which I will compare a user's question to discussion board questions.

(a) "Are notes allowed on the exam"
(b) ["are", "notes", "allowed", "on", "the", "exam"]
(c) ["notes", "allowed", "exam"]
(d) ["note", "allow", "exam"]

Figure 1 - A sentence undergoing tokenization, stop word removal, and lemmatization

Overview

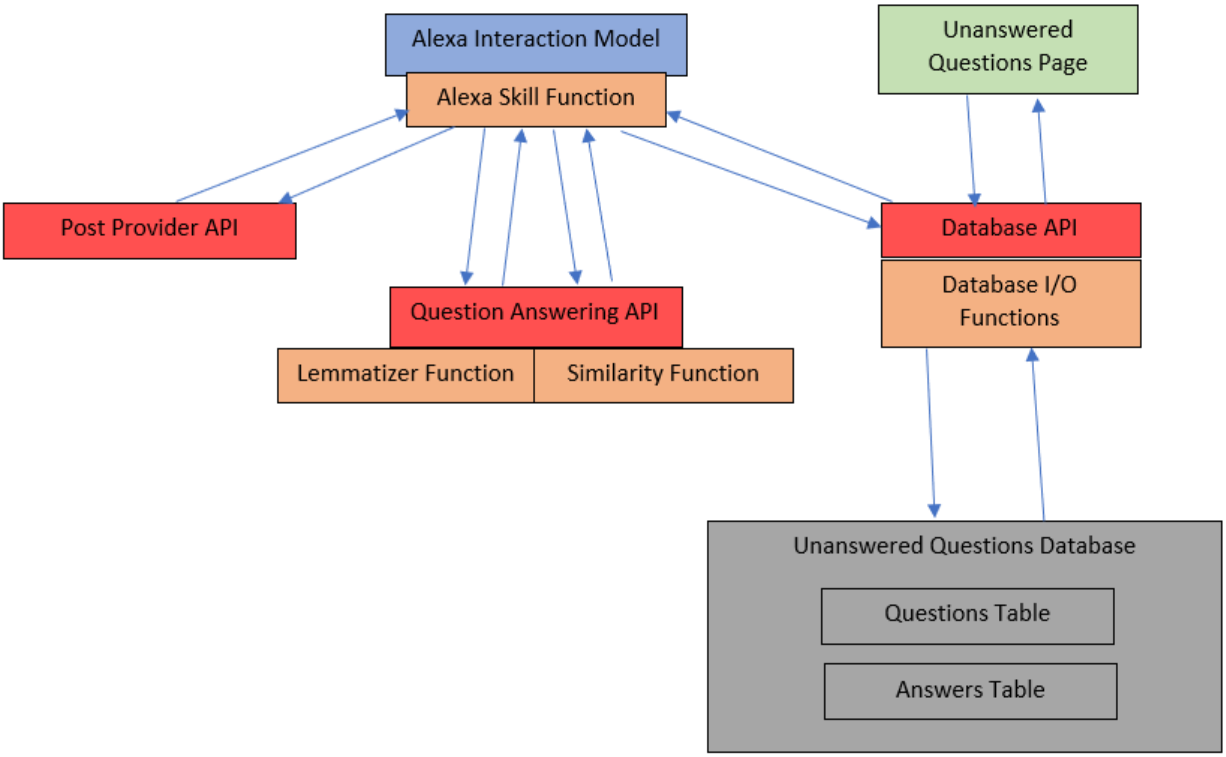

Figure 2 - The Overall Architecture of the System

This project utilizes numerous services offered by Amazon to accept, process, and answer questions. Figure 2 shows the major components of the system, with their color representing the AWS service used. Blue represents the Alexa interaction model, orange represents lambda functions, red represents APIs built using API Gateway, gray represents the DynamoDB tables, and green represents the website hosted by S3.

A general summary of how the components interact in a typical use case is as follows: a user asks Alexa a question, which the Alexa interaction model interprets and sends to its supporting lambda function. This lambda function extracts the question text, lemmatizes it, and calls the Post Provider API to obtain discussion board posts to select an answer from. After lemmatizing the subject and body portions of the candidate posts, they and the lemmatized question are sent to the similarity function in order to obtain a vector of similarity values. The

Alexa skill function then recites the replies obtained from the candidate post with the highest similarity value. If the user is not satisfied with the recited answer, or if no candidate post with a nonzero candidate score can be found, then the user is given the option of sending the post to the Unanswered Questions Database. Professors and other students can view the members of the Unanswered Questions Database from the Unanswered Questions Page and answer them. The database is not accessed directly by the skill function or the Questions Page, but through API calls in order to maintain loose coupling.

Implementation

Alexa Interaction Model

Development of this project began with the interaction model, which defines the terms by which the user will interact with the Alexa skill. First, I defined the invocation, which is the phrase that users say in order to start interacting with the skill. The invocation name of this project is simply “discussion board”.

Intents define the actions that the skill will take at the user’s direction. Amazon has many default intents, such as YesIntent, NoIntent, and StopIntent. However, I had to define a custom intent in order to have the user prompt Alexa to begin answering the question, which is passed through the intent as a slot. The only slot I defined is called “QuestionText” and figuring out how to incorporate it in the sample utterances proved difficult. Sample utterances are the words the user speaks to activate the intent and must contain a clearly defined phrase, called the “carrier phrase” in addition to the slot. Sample utterances cannot be composed of only the slot value. However, I want QuestionText to contain the full text of the question because if any words were omitted and given to the carrier phrase (such as the question words “who”, “what”, “when”, etc.) then if the user wanted to store the question in the Unanswered Questions Database it would be as a truncated form. Therefore, my sample utterances contain simple carrier phrases such as “my

question is" and "I want to know" immediately preceding the question text. This makes interacting with the skill somewhat inconvenient, as users cannot simply ask their questions but must preface them with the carrier phrase, but this inconvenience is necessary for preserving the original question text.

Alexa Skill Function

The Alexa Skill Function is the most complex component because it not only makes the service calls to obtain candidate posts, lemmatize them and the question text, and compute the similarity scores, but it also must decide the course of the conversation by defining the response to every intent. This is difficult because the desired action after an intent may differ according the status of the conversation. If the user answers "yes" to the question of whether they were satisfied by the recited answer, then we expect a different behavior from the skill than were they to say "yes" to sending the question to the Unanswered Question Database. However, both events register as YesIntents and map to the same header. Figure 3 is an activity diagram showing what actions to take in response to certain events, assuming the user does not wish to ask another question.

In order to achieve these conversational transitions, I chose to utilize Alexa's session attributes to define the state of the interaction. The skill can be in five different states: "Initial", "Answering", "Rephrasing", "SendAsking", and "Asking". Users can only ask a question when they are in the "Initial" state, and once that question is asked then they enter the "Answering" state. If the skill is in the "Answering" state and no answer is found from the discussion board data, then they will automatically transition to the "SendAsking" state rather than asking the user whether they were satisfied, which clearly would not be the case. If an answer is found, then the QuestionIntent handler would ask that quation, and if the user says "yes", triggering a YesIntent, then Alexa will enter the "Asking" state and ask them if they would like to ask another question.

In the “Asking” state, replying “yes” will return them to the “Initial” state while answering “no” will end the session. If the user says “no” to the question of whether they were satisfied, then they transition to the Rephrasing state, in which they will be asked if they would like to ask the question again in a different way. If they respond “yes” then they will return to the “Initial” state and try to ask the question again. If they respond “no”, then they will enter the SendAsking state and be asked if they want to send the question to the Unanswered Page. Both responses to this question causes a transition to the Asking state, the only difference is that saying “yes” to this question triggers an asynchronous call to the Database API to save the user’s question to the database.

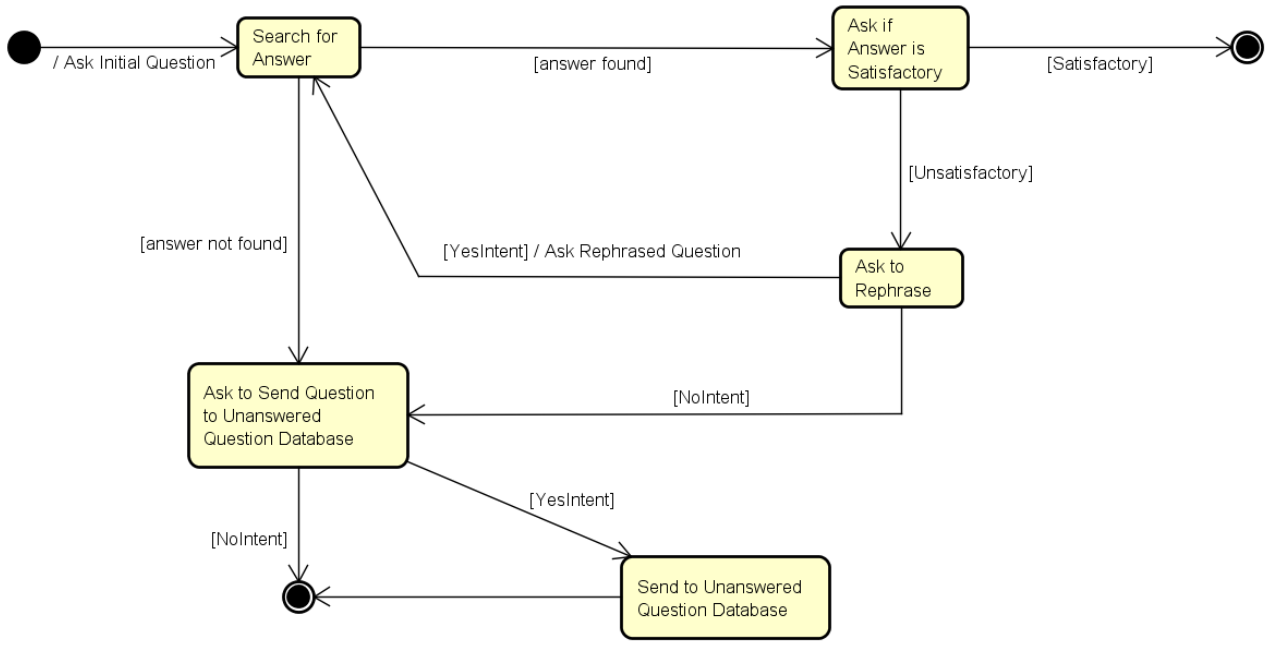

Figure 3 - Activity Diagram for the Alexa Skill

State	Description	QuestionIntent	YesIntent	NoIntent
Initial	Waiting for a Question	Answering	n/a	n/a
Answering	Answering a question and asking for feedback	n/a	Asking	Rephrasing
Rephrasing	Asking if they would like to rephrase the question	n/a	Initial	SendAsking
SendAsking	Asking if they would like to send the question to the database.	n/a	Asking	Asking
Asking	Asking if they would like to ask another question.	n/a	Initial	End Session

Figure 4 - Table of states of the Alexa Skill and how state changes on reception of certain intents

Post Provider API

The Post Provider API was created by one of my peers, Erica Anderson, to provide the Alexa Skill function with data scraped from her Fall 2018 CSE 445 section's discussion board. Because her code is separate from mine, and the only point of contact is an API call made during the answering process, there is no need to display the lambda functions or databases that go into the functionality of the service in the Figure 2 diagram.

To use this service, the client, in this case the Alexa skill function, sends a list of keywords in an array. The service then returns all discussion board posts, represented as JavaScript objects, containing one or more occurrences of any one of the keywords in its post body. Alexa obtains candidate posts by passing the lemmatized form of the user's question as the

keywords. Appendix A contains the format of a discussion board post object that would be returned from this service.

Question Answering API

This API is composed of two methods, lemmatize and similarity. Both methods are implemented by lambda functions that are coded in Python and both utilize third-party libraries. Finding a way to utilize these third-party libraries was the most difficult part of this project because the libraries are very large, and even when the deployment package is zipped for direct upload its size often exceeds 50 MB when the libraries that assist lemmatization and cosine similarity are contained in a single lambda function. This is what motivated me to divide the service into two methods, one for lemmatizing and one for similarity, that are both called in the skill function instead of having one service method that does the entire cosine similarity process.

The lemmatize function takes a string as an input and returns an array of lemmatized tokens. Implementing this function was relatively straightforward due to the use of NLTK, which automatically supports tokenizing, and WordNet, which automatically supports lemmatizing. Unfortunately, the large size of the NLTK library and the complexity of accessing Python libraries in an AWS environment made developing this function quite difficult, although not impossible.

The similarity function takes a list of token vectors and computes their cosine similarity. Difficulties with getting sci-kit Learn, a Python library that automates much of the tf-idf process, to work properly on Lambda, forced me to implement many of the steps of the cosine similarity process, such as constructing the tf-matrix, in a more manual way than I would have hoped, although I still used a Python library, in this case NumPy instead of scikit-learn

Unanswered Questions Database

The unanswered questions database stores questions that the Alexa skill was unable to give a satisfactory answer for into a table called "Questions", while answers for these questions submitted by users through the Unanswered Questions Page are stored in a table called "Answers". The "Questions" table stores the text of the question into "QuestionText" and the time the question was asked into "QuestionTime". The "Answers" table stores the "QuestionText" and "QuestionTime" of the question it is answering as well as columns for the text of the answer, time of the answer, and answerer. The schema for the two tables is as follows:

Questions(QuestionText, QuestionTime)

Answers(AnswerText, AnswerTime, Answerer, QuestionText, QuestionTime)

All these column values are of type "string". DynamoDB is the most convenient and well-documented database solution for AWS, but unfortunately it is not a relational database, and this use case calls for a one-to-many relationship between Questions and Answers.

Unanswered Questions Database API

As mentioned before, the Database is not accessed directly by the skill function or the questions page, but instead through an API. This API offers a useful layer of abstraction. If, for some reason, it was decided to move from DynamoDB to some other database, then none of the clients' code needs to change so long as the API models remain the same. API methods exist for:

- Adding answers to the Answers table, which is used for the answer functionality of the unanswered questions page.
- Selecting all questions from the Questions table, which is used for displaying the questions on the unanswered questions page.

- Adding questions to the Questions table, which is used for adding unanswered questions from the Alexa skill function.
- Selecting all answers for a given question from the Answers table, which is useful for displaying questions in the unanswered questions page.

Unanswered Questions Page

The unanswered questions page is a publicly accessible webpage implemented with React and hosted with S3. On rendering the page, each member of the Questions table is given an HTML collapsible component as shown in Figure 5. When these components are expanded, the answers to the question, if there are any, are loaded and displayed as shown in Figure 6. Users can submit answers to these questions using the textarea and button. The submitted answers are then stored into the Answers Table.

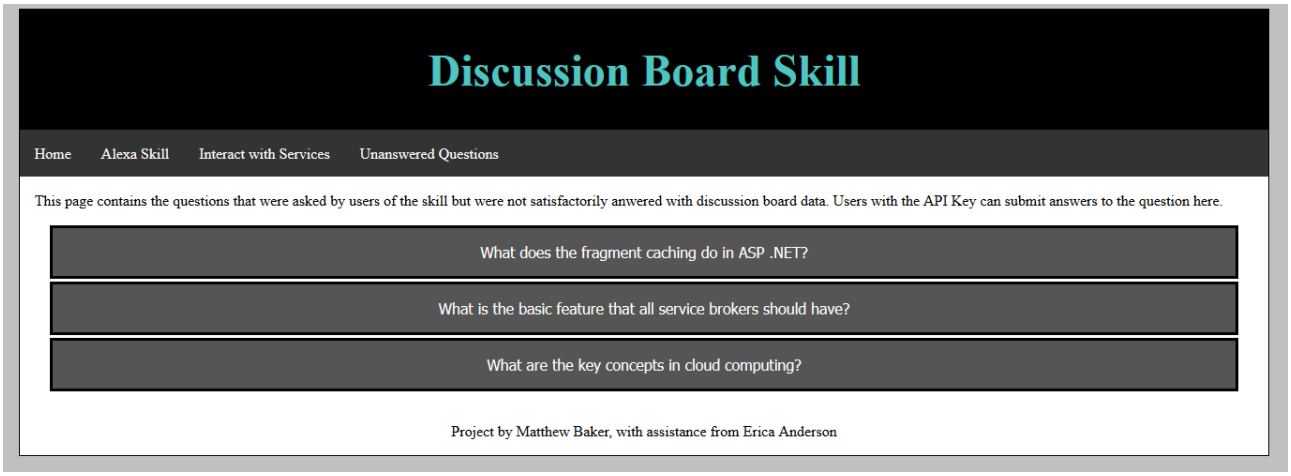

Figure 5

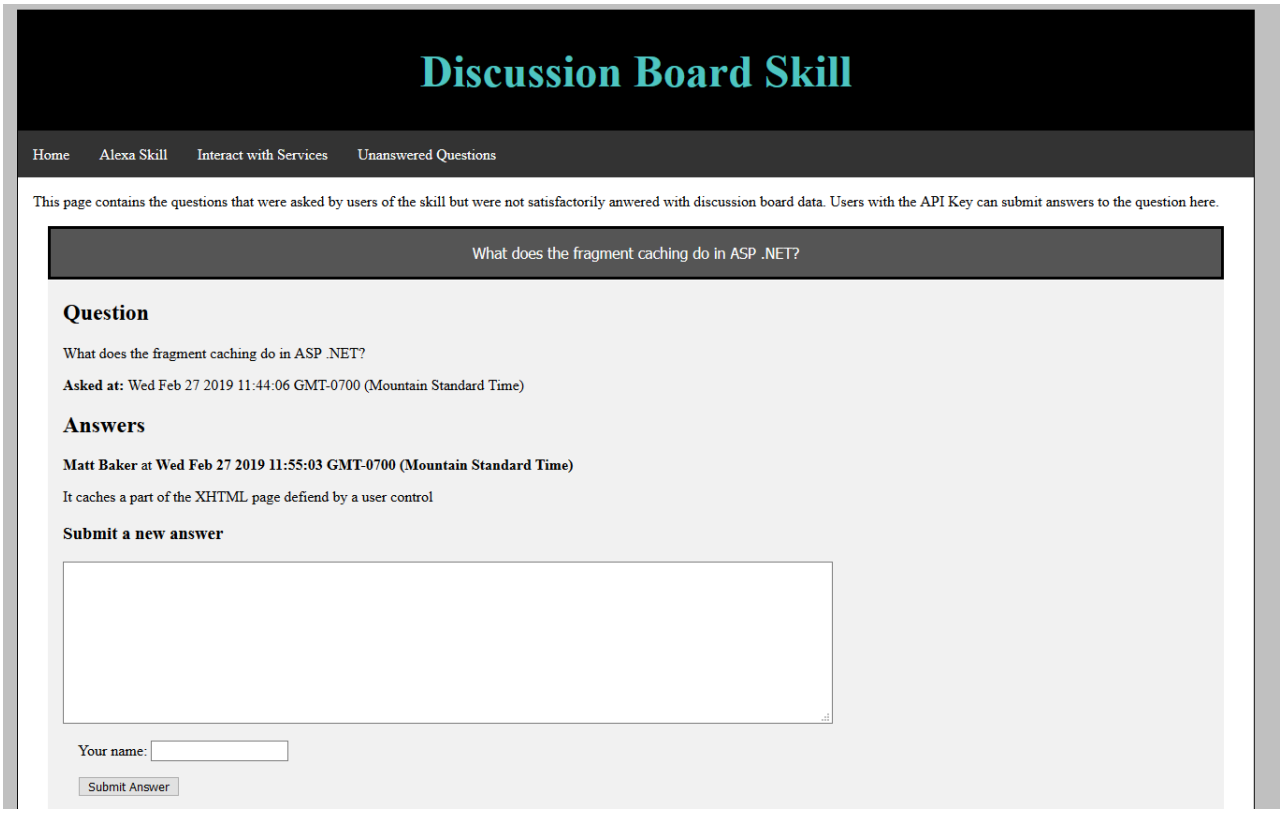

Figure 6

Results

The most immediately apparent trait of this skill is the long wait-time between asking the question and receiving an answer. This can be due to a myriad of factors, such as the size of the lemmatization and similarity deployment packages or the amount of discussion board data to evaluate. However, an interesting trend with the duration of the service calls is that the first calls after long gaps tend to be the longest. Figure 7 is the duration in milliseconds of the Alexa service lambda function, and it shows a consistent pattern of abnormally high values after periods without activity. Wait time becomes less of an issue the more the skill is used.

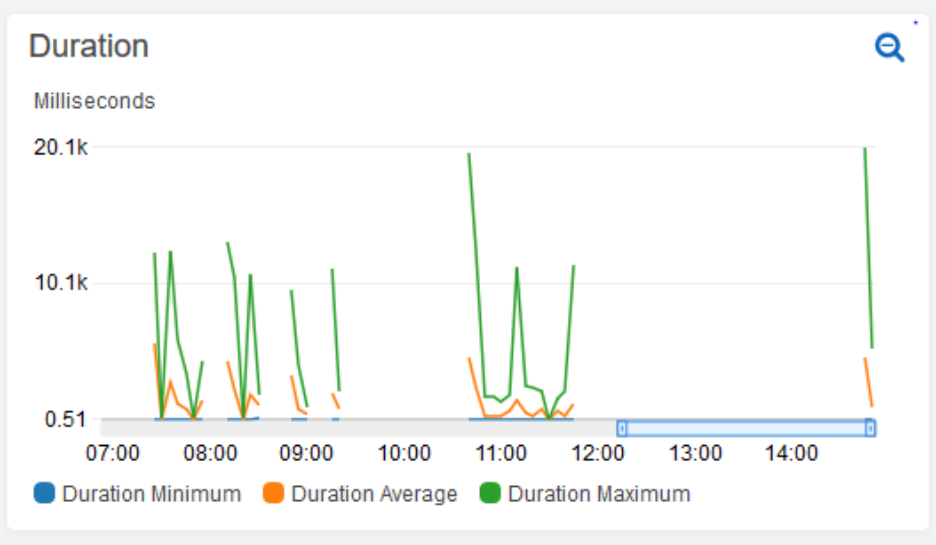

Figure 7 - Duration spikes after periods of inactivity

The skill performs well when the user asks a question that already exists on the discussion board. For example, a discussion board post exists in the dataset with the subject, "Style sheet not applying". If a user asks the skill the skill matches with that post with a cosine similarity score of .7933. However, this tool does not perform well when it comes to understanding the semantic meaning of the question being asked. For example, if we asked the skill, "Can we use a cheat sheet on the exam" the skill will respond with: "No proctor is needed. The exam will be open book and open computer. The exam will be tightly timed." However, if we ask Alexa the same question but use "test" instead of "exam" we'll get a different result, as pictured in Figure 8.

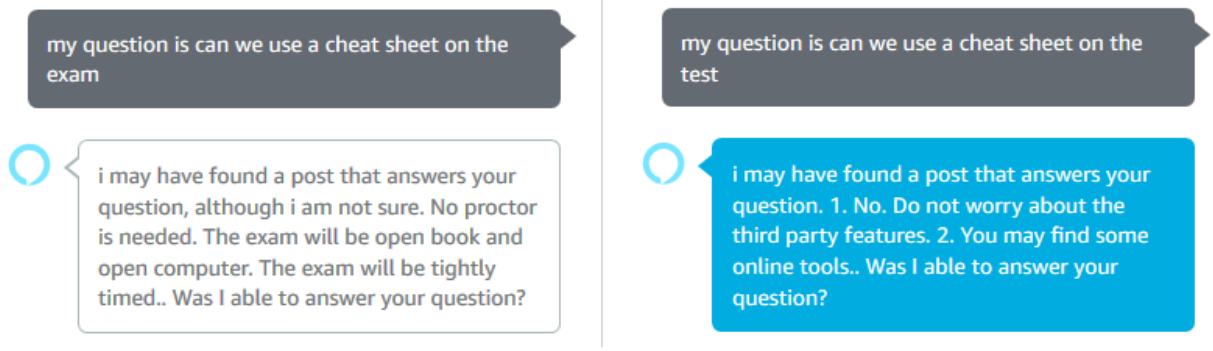

Figure 8 - Substituting one word with a synonym has a catastrophic result

This happens because cosine similarity is based on the overlap of *words*, not the overlap of *meaning*. The reason why the second wording of the question has a different answer is because the old answer originated from a post that featured the word "exam" but not the *exact* word "test". Meanwhile, the new answer originated from a post that featured the word "test" (or "testing", which is lemmatized as "test") but not as a synonym for "exam". Cosine similarity as used in this project does not take the words' context into account, nor is it capable of understanding the concept of synonyms. Figure 9 shows this flaw in the question processing.

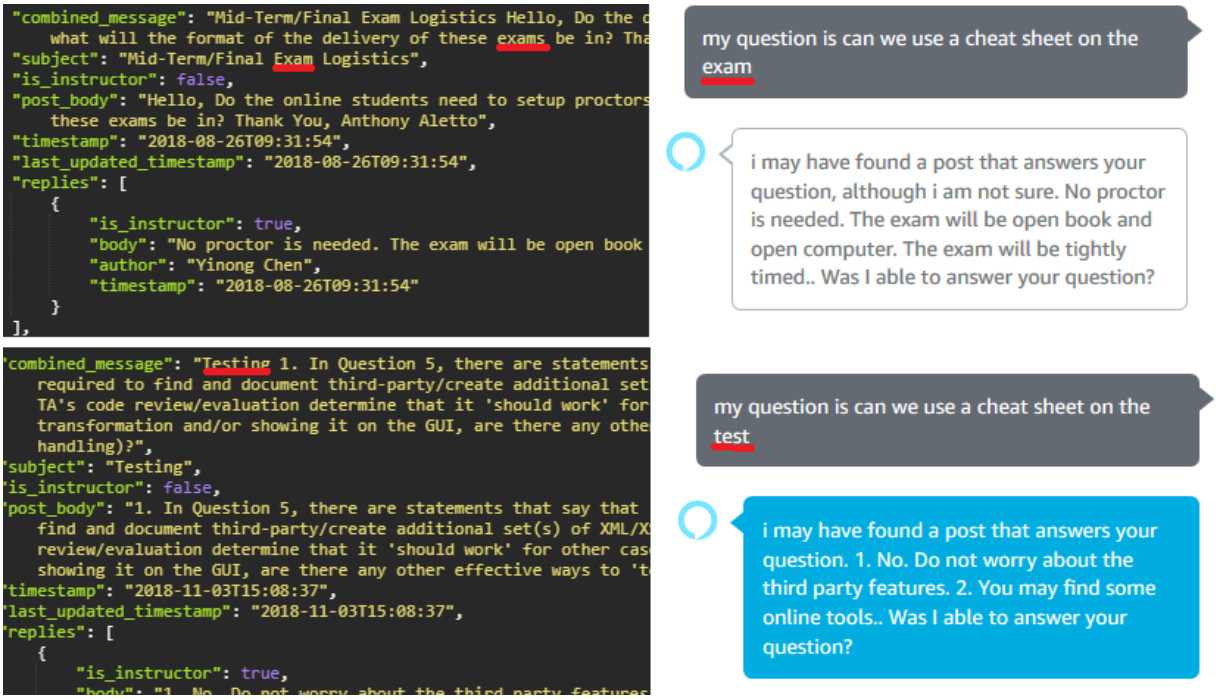

Figure 9 - Cosine similarity is about matching words, not meanings

The inability of the skill to find an answer for every single question posed is not a fundamental flaw of the program. The data to answer that question may not be present in the discussion board data or in the unanswered database. However, the inability of this skill to understand what is being asked for *is* a fundamental flaw and limits this skill's usefulness. If the goal of Alexa is to mimic human interaction, then it needs to more closely mimic human thought.

Conclusions

The goal of this project was to create an Alexa skill capable of answering students' questions using discussion board data. Due to the performance limitations of Lambda and the difficulties in incorporating advanced Python libraries into its environment, this project was designed with the cosine similarity sentence matching algorithm instead of a more sophisticated question answering system. The skill is capable of matching questions to posts that have nearly identical word use. However, its inability to grasp more nuanced concepts such as synonyms limits this application's effectiveness. However, if AWS continues to improve its infrastructure and if advances in AI can somehow make deep learning technologies more lightweight, then a robust question-answering Alexa skill can become a real possibility.

Acknowledgments

I would like to acknowledge my director, Yinong Chen, for helping with this project, providing resources for me to use, and recommending texts to research. I would also like to acknowledge ASU student Erica Anderson for her help with this project, particularly with scraping the discussion board data.

References

[1] Bohn, D. (2019). *Exclusive: Amazon says 100 million Alexa devices have been sold*. [online] The Verge. Available at: https://www.theverge.com/2019/1/4/18168565/amazon-alexa-devices-how-many-sold-number-100-million-dave-limp [Accessed 1 Feb. 2019].

[2] Huang, L. (2017). *Measuring Similarity Between Texts in Python*. [online] Digital Scholarship Center. Available at: https://sites.temple.edu/tudsc/2017/03/30/measuring-similarity-between-texts-in-python/ [Accessed 24 Sep. 2018].

[3] Jurafsky, D. and Martin, J. (2009). *Speech and language processing*. Upper Saddle River, N.J.: Pearson Prentice Hall.

[4] LI, X. and ROTH, D. (2005). Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(03), p.229.

[5] Seckel, S. (2019). *ASU, Amazon bring first-of-its-kind voice-technology program to campus*. [online] ASU Now: Access, Excellence, Impact. Available at: https://asunow.asu.edu/20170817-asu-news-asu-amazon-dots-tooker-house [Accessed 28 Jan. 2019].

Appendices

Appendix A – Scraped Discussion Board Post Format

```
{
        "combined message": "string which is the subject of the post combined with the body",
        "subject": "post subject here",
        "is_instructor": bool,
        "post_body": "body of original post here",
        "timestamp": "timestamp in iso format",
        "replies": [
                {
                        "is_instructor": bool,
                        "body": "reply body",
                        "author": "post author",
                        "timestamp": "timestamp in iso format"
                }, ...
        ],
        "url": "url of post",
        "discussion_board_name": "name of discussion board the thread belongs to",
        "author": "post author"
}
```

Appendix B – Links

GitHub Repository for Alexa Skill Implementation: https://github.com/mebaker7/alexaDiscussionBoardSkill

Unanswered Questions Website: http://alexadiscussionboardskill.s3-website-us-west-2.amazonaws.com/UnansweredQuestionsPage

Blog Detailing Project Development: http://amazonalexa499.blogspot.com/