CollegeBot: A Conversational AI Approach to Help Students Navigate College

Mohinish Daswani, Kavina Desai, Mili Patel, Reeya Vani, and Magdalini Eirinaki(✉)

Computer Engineering Department, San Jose State University, San Jose, CA 95192, USA
{mohinishmaheshkumar.daswani,kavina.desai,mili.patel,reeya.vani, magdalini.eirinaki}@sjsu.edu

Abstract. In an organization as big as a university that has many distinct departments and administrative bodies, it becomes almost impossible to easily obtain information online or by other means. Assistance over the phone or in-person is often limited to office hours and the information online is scattered through numerous (often nested) web pages, often independently administered and maintained by each sub-division. In this work, we present CollegeBot, a conversational AI agent that uses natural language processing and machine learning to assist visitors of a university's web site in easily locating information related to their queries. We discuss how we create the knowledge base by collecting and appropriately preprocessing information that is used to train the conversational agent for answering domain-specific questions. We have evaluated two different algorithms for training the conversational model for the chatbot, namely a semantic similarity model and a deep learning one leveraging Sequence-to-Sequence learning model. The proposed system is able to capture the user's intent and switch context appropriately. It also leverages the open source AIML chatbot ALICE to answer any generic (non domain-specific) questions. We present a proof-of-concept prototype for San Jose State University, to demonstrate how such an approach can be easily adopted by other academic institutions as well.

Keywords: Chatbot · Conversational AI · Natural language processing · Deep learning · Sequence-to-Sequence · AIML · Semantic sentence similarity.

1 Introduction

Conventional methods of communication with companies and other organizations like email, call centers, or automated voice response system, are often time-consuming, slow, and tedious. These communication methods might be restricted to certain working days or a certain number of hours per day. To address these

© Springer Nature Switzerland AG 2020
C. Stephanidis et al. (Eds.): HCII 2020, LNCS 12424, pp. 44–63, 2020.
https://doi.org/10.1007/978-3-030-60117-1_4

issues, companies and businesses have started to adopt automated systems like chatbots, voice assistants, and other conversational AI. Chatbots are software programs that can converse with humans and provide answers to their questions. About 67% of the clients utilized conversational AI for client assistance in the business a year ago [11]. The simplest form of chatbots that is employed by many companies are keyword-based and return predefined answers to a set of generic questions. Some more advanced ones include answers to domain-specific questions pertaining to particular tasks in the company. However, we rarely see such chatbots be used by universities and other academic institutions.

In an organization as big as a university that has many distinct departments and administrative bodies, it becomes almost impossible to easily obtain information online or by other means. Assistance over the phone or in person is often limited to office hours and the information online is scattered through numerous (often nested) web pages, independently administered and maintained by each sub-division. In this work, we present CollegeBot, a conversational AI agent that uses natural language processing (NLP) and machine learning (ML) to assist visitors of a university's web site in easily locating information related to their queries. Our approach caters to a necessity for domain engineering as it includes end-to-end implementation from data retrieval using a web cralwer, to data preprocessing and real-time response generation to the end user's queries.

We considered and evaluated two different algorithms for training the conversational model for the chatbot. The RNN-based Sequence-to-Sequence (seq2seq) model is one of the state-of-the-art models used in the literature for AI chatbot systems [15]. The second model we evaluated is a semantic similarity model that follows a more traditional NLP approach. We evaluate the two approaches based on the computational efficiency in terms of time, and the correctness of the answers produced by them using the *BLEU* Score method. The proposed system is also able to capture the user's intent and switch context appropriately in a fast and efficient manner. In addition to the trained model used to respond to domain-specific questions, our prototype also uses ALICE bot, an open-sourced implementation of pre-coded AIML files to carry out informal conversation, usually used in the beginning of a chat.

In the rest of this paper, we discuss the technical design, preliminary experimental evaluation, and proof-of-concept prototype implementation of CollegeBot. We provide a brief review of related work in Sect. 2 and discuss the system architecture in Sect. 3. We then explain how we created the system's knowledge base in Sect. 4 and discuss in detail the design and evaluation of the training algorithms in Sect. 5. Finally, we present some technical details of our prototype in Sect. 6 and conclude in Sect. 7.

2 Related Work

We can distinguish between two kinds of chatbots, one which is based on a set of rules and principles to be followed for a particular query, and another that is based on artificial intelligence (AI) systems that iteratively get smarter

and more efficient. Two of the most popular open-sourced chatbots are ELIZA and ALICE. ALICE (Artificial Linguistic Internet Computer Entity) is one of the most popular and award-winning chatbots that primarily focuses on natural language understanding and pattern matching [2]. The famous pattern matching algorithm of ALICE is a simple depth-first search technique used to find the longest matching sequence and then generate an appropriate response based on the pattern matched. The underlying data structure that ALICE is built upon is AIML. AIML stands for Artificial Intelligence Markup Language and is an extension of the traditional XML (Extensible Markup Language). AIML was designed by the Alicebot open source community to empower individuals to include pattern information into chatbots [1]. In ALICE the core design of the chatbot engine and language knowledge model [2] are isolated. This offers us the chance to effortlessly introduce a newly designed language model on the top of the base system. On the contrary, ELIZA is built using more complicated rules and directions, that requires coming up with input transformation functions, output transformation functions, and complex keyword patterns that show input and output [2].

There exist several domain-specific chatbots. For instance, MamaBot is a conversational AI system built to help pregnant women, mother, children, and their families [13]. The authors outline three main aspects of building a chatbot architecture: intent identification and classification, entity recognition, and response generation. They use the open-source Microsoft Bot Framework which internally uses LUIS (Language Understanding Intelligent Service) as its cognitive service. Another example is the bank chatbot introduced in [4]. The authors discuss the process of applying NLP on the user input, using ML classification algorithm to identify the class it belongs to, and then finally applying cosine similarity between the input query and the questions of that same identified class.

Among the few chatbots related to college inquiries is UNIBOT [7]. The authors introduce a new algorithm for finding the relevant answer from the knowledge database. The database consists of a row that has all the questions and more than one row is allocated for the answers, as it is presumed that one question might map to multiple answers. The user input is fetc.hed, stopwords are removed, the keywords are extracted, and then it searches in the database using either pattern matching or comparing SQL regex. In [3] the authors leverage the implementation of ALICE chatbot as a domain-specific chatbot that can act as an information system for undergraduate students. They present three chatbots that have different knowledge repositories: BaseBot, the most basic implementation containing the AIML files, UFaqBot(Dom_eng), a domain engineered system designed with knowledge repository, and UFaqBot(rep), a hybrid system which is an amalgamation of basic dialog conversation and domain knowledge. This setup is very similar to our proposed system, that leverages ALICE for generic questions, includes a domain-specific chatbot engine, and switches between the two as needed. However our approach in the domain-specific chatbot differs from our CollegeBot.

As we are dealing with a domain-specific conversational AI system, a user will be asking the chatbot specific questions about a particular topic. Thus, entity extraction and context identification NLP techniques are also very important. Additionally, as input comes in free-form text, there can be more than one ways to pose the same question. The chatbot system should recognize such similarities. Depending on the query representation method, this can be done employing vector similarity metrics such as cosine similarity, which is what we use in our semantic similarity engine, or ontology-based metrics like Wu-Palmer similarity and path similarity, used in [5].

Neural networks have been recently introduced for dialogue generation in conversational AI agents. As described in [14], the Sequence-to-Sequence (seq2seq) model depends on the recurrent neural network (RNN) that takes in the input sentence one token at a time and in a similar way predicts the output sequence one token at a time. At the time of training the model, the correct output sequence is fed into the model, so that backpropagation can be performed to make the learning more accurate. Long sequences can be problematic for deep neural networks on the grounds that they necessitate the dimensionality of the input and output to be fixed. In [12] the authors talk about the Long Short-Term Memory (LSTM) approach that can tackle the basic seq2seq problems. The approach is to utilize one layer of LSTM to read the input sequence, compute a fixed-size vector representation of that input sequence, and use the other LSTM to extricate the output sequence from that vector representation. The approach most similar to ours is discussed in [8]. This paper discusses the design of a web assistant where the ML module is based on the RNN wherein the seq2seq model is fed the input sequence as multiple tokens and it further gives a proper arrangement of tokens as the output sequence.

There is a lot of research and studies conducted on chatbots, however, various challenges have also been identified in developing one. We researched these challenges to find a way to address them while developing our applications. Challenges include limited amount of training data, context modeling and context switching, emotion recognition, natural language processing, language specialization, speaker-listener specific modeling, and many more. A general rule for ML systems is that the more the training data, the better the model is. Chatbots, especially those relying on deep learning, need lots of training data for the machine to learn the chatting pattern and to create a model to correctly maintain a conversation about a topic. Moreover, a common challenge in NLP is learning and understanding the syntax. There can be multiple ways a question or statement be made. For example, "What's the weather?" can be asked as "Could you check the weather?" [9]. We explore how different approaches in designing the chatbot engine address the aforementioned issues.

3 System Architecture

Figure 1 shows the system architecture of CollegeBot. Chatbots need a *knowledge base* to generate appropriate responses to the user's queries. To gather data

for our knowledge base we have used a web crawler. The data relating to the university is spread across several web pages and needs to be aggregated, filtered, grouped, and indexed. For our system prototype we collected most of the data from FAQs pages posted on various SJSU web pages for various topics such as admissions, international student and scholar services, courses offered, majors, and so on. The process is described in Sect. 4. However this selection could be expanded to cover more pages within a University.

The data is next fed to the *Preprocessor* module. In this module, we employ NLP techniques to bring the data in appropriate format for input to the *Training* module. This module is employed both in the back-end but also in the front-end part of our architecture, used to preprocess the questions of the end user prior to submitting it to the chatbot engine. The *Preprocessor* module, along with the *Training* and the *Engine* modules consist the core part of CollegeBot. They rely on NLP and ML to be able to decode a user's question and retrieve the most appropriate answers. We considered and evaluated two different algorithms for training the conversational model for the chatbot, namely a semantic similarity model and a seq2seq-based model, which we discuss in detail in Sect. 5.

In the front-end, the system checks whether the question is domain-specific or not. The system sends the domain-specific questions to the *Preprocessor* and subsequently the chatbot *Engine* module. The *Engine* then checks whether there exists a context for the sentence. If a context is found then it identifies the most similar question to the user query and returns the corresponding answer as the response. If the context is not found then it will check for the new context, append it, find the most similar question and return the corresponding answer as the response. The back-end then takes the response from the *Engine* and sends it to the front-end where it is displayed to the user, as shown in Fig. 2. Non domain-specific questions are handled by the *ALICE AIML* server. We discuss the technical details of the core modules of the CollegeBot prototype in more detail in Sect. 6.

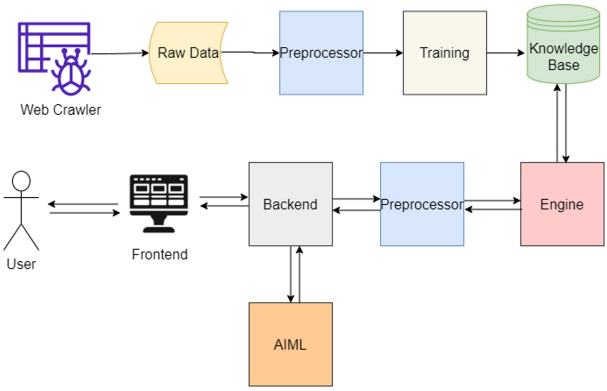

Fig. 1. CollegeBot system architecture

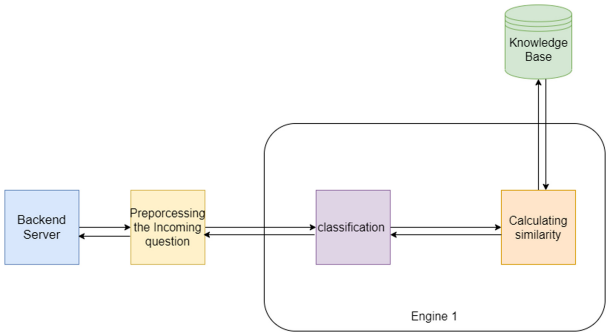

Fig. 2. Semantic sentence similarity Engine

4 Knowledge Base Creation

4.1 Data Collection

For creating the knowledge base for our proof-of-concept prototype we used the San Jose State University (SJSU) website[1]. We collected most of the data from the FAQs posted on the SJSU website for various topics such as admissions, international student and scholar services, courses offered, majors, and so on using a web crawler. We designed the web crawler to start from a particular page and collect all the links present on that page. It then filters the list to return the links which point to the SJSU website. The crawler then visits those pages, parses each page, and creates a list of python dictionaries containing questions, answers, relevant URLs, and the context/category of the question. It identifies all the questions and answers using regex for pattern matching. It also identifies any hyperlinks present in the answers and also adds them to the python dictionary. As shown in Fig. 3 the answer contains the hyperlink "Registrar" which points to the URL for the SJSU office of the registrar. The web crawler identifies this URL in the answer and adds it to the dictionary of FAQs. The crawler also captures the context for which the FAQs are posted and also adds it to the python dictionary. When the web crawler starts parsing a new page for FAQs it takes the title of the page as the context for all the FAQs present in that page. In the example of Fig. 3 the context is "GAPE" (Graduate Admissions and Program Evaluations) which was taken from the title of that page. An example of the python dictionary created for each FAQs is also shown in Fig. 3. This dictionary is then added to the python list which is then stored in a comma-separated file once the entire page is parsed. This process is repeated again for the next link.

During the process, the web crawler can encounter some pages where it is not able to parse the data. We made a list of all those links and changed the regex for identifying FAQs for those links. Additionally, some of the topics such

[1] http://www.sjsu.edu.

as faculty information, or research conducted by different faculty members and departments did not have FAQs. For such topics, we visited the SJSU website and collected the data in a question-answer format.

In all, we gathered 923 questions from SJSU's website. We had shortlisted some topics and focused on only those topics as part of this study. We covered topics like health center, recreation center, library, international students cell, admissions, courses and professor information.

For testing our system, we required a smaller dataset. We created this dataset by randomly selecting the questions from the training set and created variations of a few other randomly selected questions. This dataset only has questions and expected answers as columns.

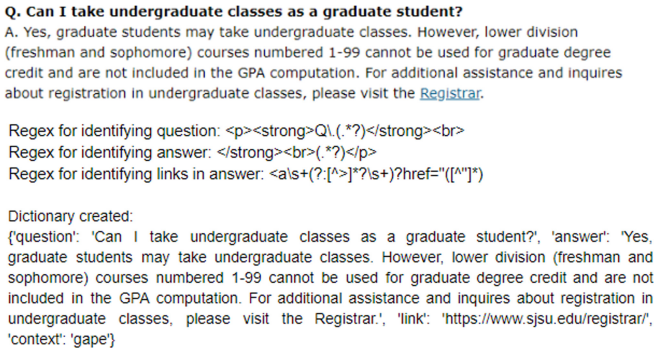

Fig. 3. Example for converting text to python dictionary using regex for pattern matching

5 Training Algorithms

For training our model, we considered two options, a purely NLP-based one, and a deep learning one. We discuss both, as well as the preprocessing steps applicable to each one in detail here.

5.1 Semantic Similarity Model

If we break down the logic of a chatbot in simple terms, it comes down to recognizing if a similar kind of question exists in our data repository or not. However, there can be more than one ways of posing a question as shown in Fig. 4. While humans can easily understand which sentences mean the same thing and which sentences are totally different, this is not as straightforward for a conversational agent like a chatbot. However, the system should recognize such similar input queries and generate the appropriate response for that. As shown in Fig. 4, the two queries "What is the last date to enroll?" and "When is the

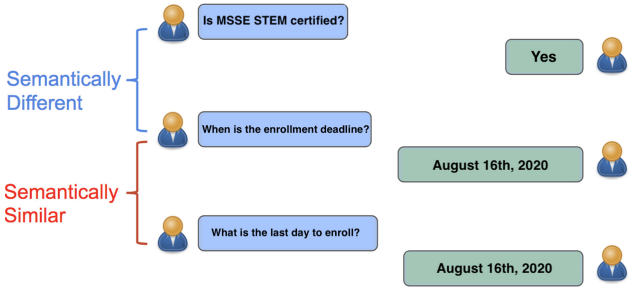

Fig. 4. Semantic similarity example

enrollment deadline?" have the same meaning (and there might be even more ways of posing the same question).

When employing this model, the system finds semantic sentence similarity. To do this, all questions in the knowledge base need to be first preprocessed, through stemming and lemmatization. We also created a dictionary for storing all the synonyms and we replaced all the similar words with the standard abbreviations across the whole dataset (e.g. words like "United States", "United States of America", "America", "USA" will be replaced by a common synonym "US" everywhere in the sentence). As we are dealing with a domain-specific conversational AI system, a user will be asking specific questions about a particular topic. Thus, entity extraction and context identification NLP techniques are also important.

In order to identify relevant questions in the knowledge base, even when not phrased identically by the end user, we calculate a similarity score between the (preprocessed) user query and the queries stored in the knowledge base. The similarity is calculated using word vectors with the help of the word2vec algorithm. This algorithm employs multi-layered neural network structure, which is modelled to form linguistic contexts of words. The algorithm takes as input a text corpus and gives a vector space as the output. There is a unique vector assigned in the space for each different word in the corpus. Thus the comparison is essentially done between these word vectors by calculating their cosine similarity and then a score between 0 to 1 is returned as the similarity score for comparison between two sentences.

Therefore, for each incoming query, the system sorts all questions in the knowledge base in descending order of similarity. A threshold for the similarity score is preset and if the highest similarity score from the matched question is greater than the decided threshold, then the corresponding answer to that question is returned to the end user. An example of the top-10 relevant questions to a user query is shown in Fig. 5. It can be observed that the incoming question was identified as almost 90% similar to the first question in the list. On the other hand, if there is no proper match between the incoming question and any of the questions from the dataset, we set a fallback mechanism and a generic response

such as “I am not sure about that” or “Sorry, I do not know that” is returned as the response.

```
****************************** Incoming question ******************************

 I had enrolled in an upper division course at other CSU, however the course did not turn out that well, so what form
should I use

****************************** Similarity ******************************

                                                Questions  Similarity score
0  I took an upper division GE course at another CSU, but...          0.895677
1  Can I take a course from my technology major (MIS, CMP...          0.864332
2  I have already published a master's thesis, a PhD diss...          0.856760
3        As a ChAD student, how can I apply for graduation?           0.853763
4  I completed all of my lower division GE and I have pas...          0.830573
5  I took an American Institutions (US 1, US 2, US3) cour...          0.824764
6  I tried to enroll in a class, but the enrollment (regi...          0.821910
7  I tried to enroll in a class, but the enrollment (regi...          0.821910
8  I tried to enroll in a class, but the enrollment (regi...          0.821910
9   Do I need to sign up for utilities, or internet service?          0.819263
```

Fig. 5. Implementation of semantic sentence similarity

Although this approach yields high accuracy, it is inefficient in terms of time taken to generate the response, as the system needs to preprocess and calculate, in real-time, the similarity between the user query and all questions in the system’s knowledge base. Additionally, such approach does not take into consideration the context, i.e. previous questions the user might have asked. Humans can interpret the context easily and the conversations never break. However, sometimes when a user is interacting with an AI chatbot, they may not explicitly mention the context in every question, and thus the bot may not give an appropriate, context-specific response. This results in either a wrong answer or the bot saying that it doesn’t know the answer, even though the bot was trained to have the answer for that specific question as well. See for example the questions in Fig. 6. The user first asks about Open University and the response by the bot is accurate. The user follows up asking about the application deadline (implying this applies to Open University admissions), but the bot cannot interpret that. So, it returns the answer for which it found the maximum similarity for the question resulting in a contextually wrong answer.

To overcome this problem, we categorized the knowledge base into a few specific categories or contexts, such as Open University, GAPE (Graduate Admissions & Program Evaluations), MSSE (Master of Science in Software Engineering), ISSS (International Student & Scholar Services), and so on. Whenever a

Question (User)	**Response (*CollegeBot*)**
What is an Open University?	"Open University," or "Continuing Education" at SJSU allows individuals that are not registered in a degree program to take university courses. There are a number of restrictions.
What is the application deadline?	University Graduate admission deadlines are listed at http://www.sjsu.edu/gape/prospective_students/deadlines/

Fig. 6. Sequence of questions and responses without context

question is asked, the query is preprocessed reducing every noun in the question to a specific word from the dictionary and saving it in the user's browser environment. When a question follows without a context, the bot assumes that the user is referring to the previous context and the saved context is appended to the query. Once the context is identified, the similarity score is calculated only for the questions pertaining to this context, thus reducing the time to return the accurate response as well. This results in context-specific answers, as shown in Fig. 7. We observe that the response returned by the bot for the same question about the application deadline is now pertaining to Open University.

Question (User)	**Response (*CollegeBot*)**
What is the Open University?	"Open University," or "Continuing Education" at SJSU allows individuals that are not registered in a degree program to take university courses. There are a number of restrictions.
What is the application deadline?	Please see the registration website for Open University: https://www.sjsu.edu/openuniversity/registration/

Fig. 7. Sequence of questions and responses with context

One shortcoming of this methodology is that it plainly depends on the similarity score calculated between the sentences. This algorithm used word vectors to find the similarity between sentences but it would not be able to link different synonyms. For example "What is the deadline" and "What is the last date" includes two different ways implying the word deadline, however these words would have different representations in the form of word vectors. Thus the similarity score between these kind of sentences would not be very high.

5.2 Seq2seq Model

In this model, we leverage deep neural networks (DNNs) by employing the seq2seq algorithm. DNNs [12] are strong models which can achieve incredible performance on learning difficult and complex tasks such as speech recognition or object detection. However such networks can be only be applied to use cases or problems with input parameters and target variables encoded in a fixed dimensionality. However this does not apply in cases when the text sequence length is unknown at the beginning. Answering free-form user questions falls under this category, as the questions (and answers) are modelled as a sequence of words of variable length. Our approach incorporates a multi-layered LSTM (Long Short-Term Memory), a modified version of Recurrent Neural Network (RNN), that maps the given input sequence to a vector or matrix with a fixed dimension. The next step of the LSTM decodes back the input sequence from the vector or matrix and produces the output.

As a part of preparing the data file to be fed to the algorithm, we manually append a delimiter between the question and the answer so that the algorithm

can distinguish between those two and store them as a vector pair. Also, the algorithm takes care of removing unnecessary special characters and symbols. We first pre-process the data by tokenizing the sentences and converting the words into numbers that uniquely identify them. These tokens are stored into a vocabulary so that we can convert them back to words. After processing the data, the algorithm converts the question and answer pair into word vectors and then performs the encoding and decoding on those vectors. An example is shown in Fig. 8.

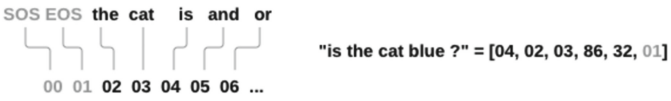

Fig. 8. Tokenizing a sentence [10]

The tokenized number form of the sentences is sent as a batch to the encoders. The batch is of a fixed size and the length of the sentences in each batch is also fixed. To maintain the size of sentences, padding is added. The sentences are also padded with a token at the beginning to identify the start of the sentence and a token is appended at the end of the sentence to identify the end of the token. An example of this is shown in Fig. 9.

The matrix or tensor thus formed is transposed to be sent to the RNN. The encoder takes the input from the tensor till an end of the sentence is encountered and a vector representation called context vector is the output. The tokens which are semantically closer are grouped together in the n-dimensional point which are passed into the next layer, bi-directional Gated Recurrent Unit (GRU). GRU is essentially a modified and updated version of a traditional LSTM approach which proves to be computationally less costly than LSTM. The GRU then generates a new tensor by reading the input tokens in forward order in the first layer of GRU and in reverse order in the second layer.

The information is passed from the encoder to the decoder in the form of a context vector. Certain words in the input sequence are of greater importance than the other words and thus more attention needs to be given to them. For

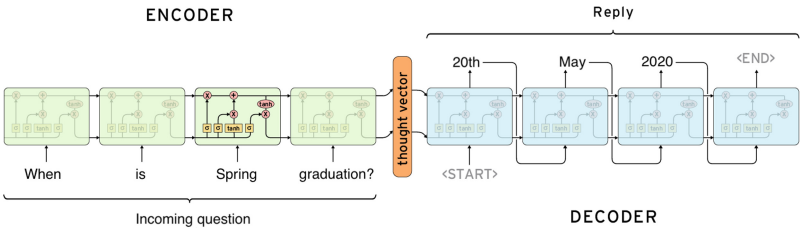

Fig. 9. Example of the encoder decoder architecture

that purpose, we use the Luong Attention layer mechanism. A weighted sum of all the encoder outputs is calculated and then that computed context vector is used in calculating the hidden state of the decoder. Thus, the output of the decoder depends on three things: decoder output from previous step, the hidden vector of the decoder, and the context vector computed from the attention layer.

The next step is to train the model. This step consists of setting the right value of the hyper-parameters such as number of iterations, teacher forcing rate, learning rate and so on. The training time depends on the size of the data and the number of epochs. After the training process is completed, the model reads the user input, evaluates it and finds the relevant answer with the help of the model trained earlier. Thus, this approach gives a faster reply on the fly, but it takes a very long time for the model to train.

Due to limited input data, the model does not converge quickly during training. One way to ensure better convergence is to provide the expected answer to the decoder instead of using the decoder's predicted value. This mechanism is called "Teacher forcing". Another shortcoming we encountered during the implementation of this model is that it fails when a word outside the vocabulary is given. When a question contains an unrecognized word, it does not generate an answer. The model should ignore the unrecognized word and try to find an answer. This shortcoming is due to the limited size of the data and can be overcome with a rich training dataset.

5.3 Evaluation

We evaluated the two training models in terms of their response time and accuracy. For testing response time, we provided the same queries to both the models to calculate the time it takes for them to return an output. Half of the test queries were identical to some included in the training data (i.e. the knowledge base) and the other half were rephrased questions. We also wanted to evaluate the accuracy of the model as a function of the number of the correct answers predicted for the input queries. We selected a subset of queries to evaluate the models' accuracy. Half of the queries were rephrased questions, while the other half included only a few keywords. We discuss our results and findings in what follows.

We used *BLEU* (Bilingual Evaluation Understudy) score to evaluate the accuracy of responses for both the models. *BLEU* score is a metric to compare the similarity between a generated statement to a reference statement. The score metric ranges from 0.0 to 1.0 where 0 means no similarity and 1 means perfect match. It is defined as follows [6]:

$$BLEU = BP * exp(\sum_{n=1}^{N} w_n \log p_n)$$

$$BP = \begin{cases} 1 & \text{if c} > \text{r} \quad (1) \\ e^{(1-r/c)} & \text{if c} \leq \text{r} \quad (2) \end{cases}$$

where c denotes the candidate sentence length and r the reference length. In essence, the score takes a weighted geometric mean of the logs of n-gram precisions up to some length (usually 4), adding a penalty for too-short predictions.

We used "sentence_bleu" from Python's NLTK library[2] to measure the correctness of the answer generated by both the models. We wrote a script to measure the *BLEU* score of the test data with various threshold values to determine the correctness of the model. *BLEU* Score gives us results between 0 and 1. Additionally we are measuring the time taken by both the models to respond to a user query. Thus, the average time taken by the semantic similarity model was 2.03 s and that for the seq2seq model was 0.41 s. However, the training time for the seq2seq model was considerably higher (and, in some setups with a larger input dataset, the system would not converge even after thousands of iterations).

Semantic Similarity Model. We measured the accuracy of the semantic similarity model by using the true positive values and the size of the dataset. Our preliminary findings after implementing the Semantic Similarity approach was that the loss of context in the conversations breaks the dialogue or renders the responses meaningless. Moreover without having a category to search for, the model has to look for answers in the entire dataset, which increased the complexity of the solution and time to return an answer. Also, we noticed that nouns in a question can have multiple synonyms which the users can use. And if these are not mapped to the dataset then the responses would be inaccurate.

To overcome all these limitations, we devised a few steps of preprocessing. We created a dictionary for the most common nouns in the datasets and added all the synonyms the users might use. As part of the preprocessing step, when the user query comes the model, the preprocessor will replace all the nouns in the question with dedicated nouns in the dataset as mapped by the dictionaries. We added the categories for each question in the dataset and similarly saved the context of the incoming query in the user's environment to keep the model context-aware in the conversation. The bot would now search for the responses in that specific category the user is referring to. This drastically reduced the computation time for the model. So, we concluded that more the categories we divide the dataset into, the faster the processing becomes. As shown in Fig. 10, the accuracy of the model was persistently over 90% for various sizes of input query size.

Seq2seq-Based Model. The answer generated by the algorithm is termed as actual answer and the correct answer is termed as expected answer. We measure the *BLEU* score of these two and if the score is above the threshold value, we increase the number of true positive answers. We measure the score for each pair of the expected and actual answers over the test data set. We finally calculate the number of correct answers based on our threshold and find the accuracy percentage using the total test data set size and the true positive answer values.

[2] https://www.nltk.org/_modules/nltk/translate/bleu_score.html.

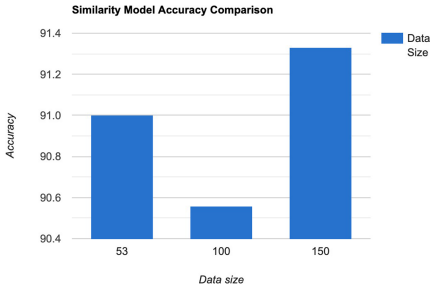

Fig. 10. Semantic similarity: performance in various sized data sets

To find the balance between the number of iterations and the data size where the model would converge, we trained various models of same data size and varying number of iterations. We gathered results for the test data for threshold values of 0.7, 0.75 and 0.8 for the seq2seq model, for various numbers of queries. We also measured the decrease in loss using "maskedNLLLoss" while increasing the number of iterations to see how the dataset is modeled to work with the algorithm. The results, for various test data sizes and number of iterations are shown in Figs. 11, 12, 13, 14, 15 and 16. We observe that the optimal hyperparameters vary depending on the dataset size, but we can achieve 80% or higher *BLEU* score accuracy for the optimal parameters for each dataset.

We observed that, since the seq2seq model generates its vocabulary from the training dataset, it is unable to recognize the word that is outside of its vocabulary. So, one limitation of seq2seq model is that it will not provide an answer when a word outside of its vocabulary is used to ask a question. If we are limiting the vocabulary by limiting the dataset, then the chances of the model not recognizing a new word increases by default due to the nature of the algorithm. Thus, to extract the best possible result of seq2seq model, we need to train the model for appropriate number of epochs and feed in with a large enough dataset such that the vocabulary of the model can be formed properly. On the other hand, a larger corpus of training questions requires a larger number of iterations to converge and increases the computational complexity of the model.

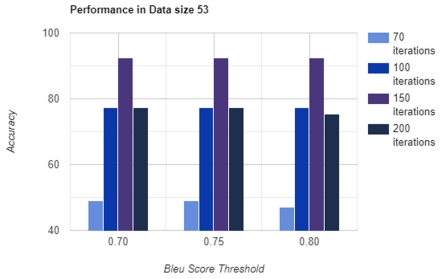

Fig. 11. Seq2Seq: performance in dataset of size 53

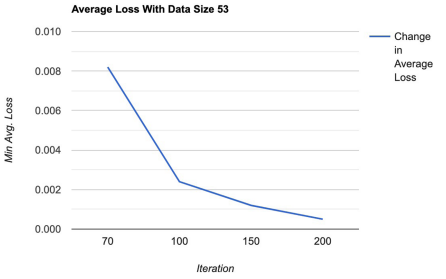

Fig. 12. Seq2Seq: average Loss with Data size 53

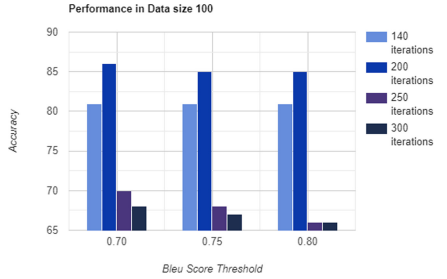

Fig. 13. Seq2Seq: performance in dataset of size 100

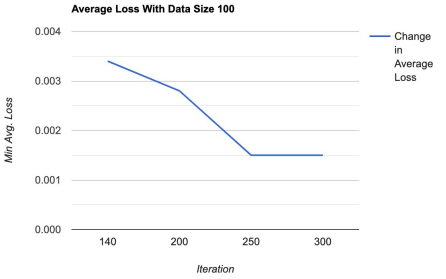

Fig. 14. Seq2Seq: average Loss with data size 100

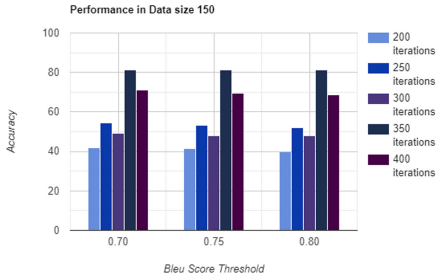

Fig. 15. Seq2Seq: performance in dataset of size 150

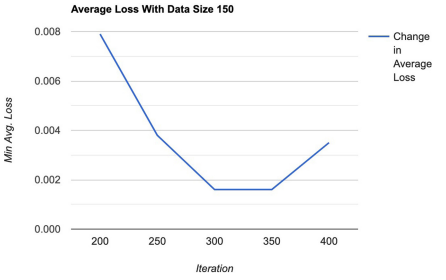

Fig. 16. Seq2Seq: average Loss with data size 150

6 CollegeBot Prototype

Figure 1 shows the system architecture of the CollegeBot prototype. After experimenting with both training models, we decided to implement the semantic similarity model in our proof-of-concept prototype. We have created a distributed architecture using React.js as frontend server, Java server for AIML, and Python Flask servers in the backend. The Python Flask servers in the backend are responsible only for minor tasks, thus creating a micro-service architecture.

Figure 17 explains the flow of this system. The frontend takes the user input and sends it to the Python Flask *server1* where it checks whether the question is a domain-specific question. If it is a domain-specific question it sends the question to Python *server2* for query processing. This server preprocesses the input question with techniques such as lemmatization and stemming and sends the response back to *server1*. The *server1* then sends the received string to the semantic similarity engine on *server3*. The engine then checks whether there exists a context for the sentence. If a context is found then it finds the most similar question to the user input and returns the corresponding answer as the response. If the context is not found then it will check for the new context, append it, find the most similar question and return the corresponding answer as the response. The backend then takes the response from the engine and sends it to the frontend where it is displayed to the users. If the question was not domain-specific then the server sends the question to the AIML server that sends a response back to *server1*, that forwards the response to the frontend. A snapshot of the UI of the CollegeBot prototype is shown in Fig. 18.

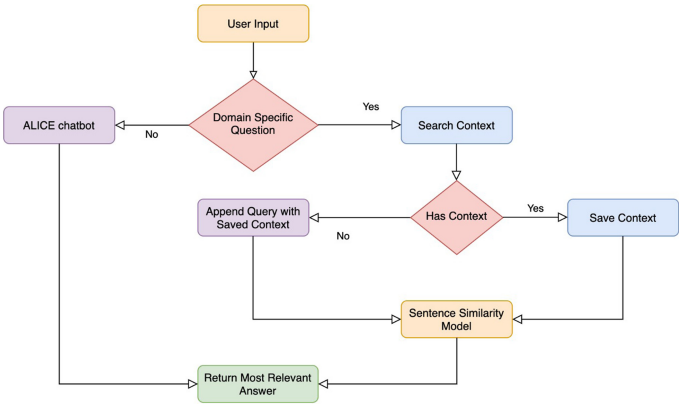

Fig. 17. CollegeBot Q&A process flow

6.1 AIML Engine

We have integrated the open-sourced ALICE chatbot [1] as part of our system prototype, to handle generic, non-domain-specific questions. ALICE has in-built coded AIML files which have generic patterns and templates to serve basic user questions such as "Hi", "How are you", "What time is it", and so on. ALICE maintains the data in the AIML (Artificial Intelligence Markup Language) format which is an extension of the traditional XML format. This format has been implemented by many other chatbots as well because it very convenient to maintain the data in such kind of templates and patterns. AIML is built of

data objects known as the AIML objects which portrays the behavior of intelligent systems. It comprises of several units/tags which are termed as topics and categories in the AIML terminology. A category can be understood as an event which comprises of patterns, that are specific user inputs, and templates, which are the corresponding answers to those patterns.

Program AB is a readily available implementation of the Alicebot. We have used the basic Java implementation of Program AB.[3] After building Program AB as a Maven project, we can chat with the system. There are multiple AIML files already present in the program which lets the system answer all the generic questions. Whenever a new input query comes, a pattern matching algorithm is applied across all the AIML files present and the pattern that has the highest match is identified and its corresponding answer is send out as the response.

We should note at this point that we also ran some tests after developing our own AIML files. We used another open-source platform for testing the manually written AIML files. A lot of wildcard characters can be used to yield better results. Also, AIML tags such as srai help in writing the same question in different ways and we can point all those questions to one answer. This way we can capture different types of ways a user can pose a question. This open source platform provides a convenient way to test the AIML files immediately after writing and it gives a nice experience as if we were chatting with an actual bot.

However, one of the major shortcomings of this approach is that it is a very basic way of retrieving answers and there is no scope of performing any preprocesing on the stored data. Thus, the performance of this approach can be very limited. The use of different kind of wildcards and AIML tags are not sufficient for building a proper intelligent chatbot system. Also, the text written in pattern tag should exactly match with the incoming query. So, the incoming query would not match if there is any spelling mistake made by the user or if the user has used completely different set of words to pose that question. AIML expects the ordering of the text in a sentence to be exactly similar to the text written in the pattern tag. Thus, this method is quite restrictive about how it expects the text to be. Moreover, it does not leave any scope for proper preprocessing step before passing the query as an input. For all these reasons, we decided to go with the semantic similarity engine to handle domain-specific questions.

6.2 Semantic Similarity Engine

The incoming query is passed as an input to the system to perform general preprocessing on it, as discussed in detail in Sect. 5.1. Using the custom-made dictionary of synonyms, the system checks and replaces all query words with their corresponding synonym abbreviations. The sentence is then stemmed using a Porter Stemmer[4] before being sent to the semantic similarity chatbot engine.

[3] https://code.google.com/archive/p/program-ab/downloads.
[4] https://www.nltk.org/api/nltk.stem.html#module-nltk.stem.porter.

The engine, depicted in Fig. 2, is the part where we are finding the semantic sentence similarity between the sentences. We used Python's spacy library[5] which uses word2vec to create word vectors and calculate the similarity score (using cosine similarity). We compute a similarity score between the incoming query and all the questions present in the data repository. As mentioned in Sect. 5.1, as soon as the query hits the similarity model, the algorithm tries to identify a context. If it does not already have a context, then it finds a context and associates it with that question. If there already is a context associated, then the model proceeds to retrieve the most similar question classified under that context from the knowledge base, and returns the respective answer. If the model was not able to find a question from the database having the similarity score above the threshold, then that existing context is disassociated from the incoming query. Further, the incoming query, now having no context, is compared for similarity for all the questions in the database and the most generic answer is written. For subsequent questions, the model already knows what context the user is referring to so the search is narrowed down and the performance is improved.

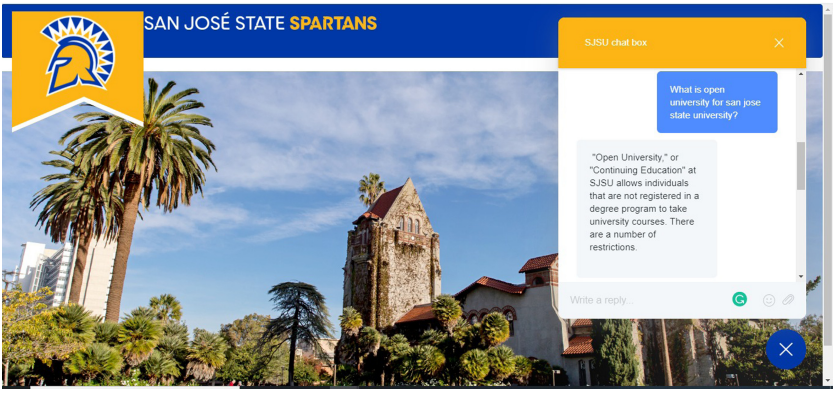

Fig. 18. SJSU CollegeBot UI

7 Conclusions

In this work we present CollegeBot, a conversational chatbot that employs information retrieval, natural language processing, and machine learning, to assist students in navigating the website of a university. We compare and evaluate two training models, and conclude for a small-sized dataset the semantic similarity model outperforms the seq2seq model in terms of accuracy. The seq2seq model is faster than the semantic similarity model, however it does not converge

[5] https://spacy.io/usage/vectors-similarity.

fast for a small-sized dataset and needs an appropriate number of iterations to train the model properly. In such cases, where the training dataset is relatively small, the semantic similarity model is a better choice. We also introduce the notion of context. CollegeBot can maintain context in a user session and retrieve the appropriate answers to subsequent, semantically-related questions. We also present our proof-of-concept prototype of CollegeBot for San Jose State University, demonstrating how such an approach could easily be implemented for other academic institutions and beyond. As part of our future work, we plan to explore leveraging libraries like Wordnet to automatically map nouns to their synonyms existing in our knowledge base. This will let the model map new incoming words to the questions in the datasets and return accurate results. Similarly, we plan to explore how additional similarity metrics, like Wu-Palmer, could be leveraged to improve the accuracy of the system.

References

1. Abushawar, B., Atwell, E.: ALICE Chatbot: Trials and Outputs: Computación y Sistemas **19**(4) (2005). https://doi.org/10.13053/cys-19-4-2326
2. Bani, B., Singh, A.: College enquiry chatbot using A.L.I.C.E. Int. J. New Technol. Res. (IJNTR) **3**(1), 64–65 (2017)
3. Ghose, S., Barua, J.: Toward the implementation of a topic specific dialogue based natural language chatbot as an undergraduate advisor. In: International Conference on Informatics. Electronics and Vision (ICIEV), pp. 1–5. IEEE, Dhaka (2013)
4. Kulkarni, C., Bhavsar, A., Pingale, S., Kumbhar, S.: BANK CHAT BOT - an intelligent assistant system using NLP and machine learning. Int. Res. J. Eng. Technol. **4**(5) (2017)
5. Lalwani, T., Bhalotia, S., Pal, A., Bisen, S., Rathod, V.: Implementation of a chat bot system using AI and NLP. Int. J. Innov. Res. Comput. Sci. Technol. (IJIRCST) **6**, 26–30 (2018). https://doi.org/10.21276/ijircst.2018.6.3.2
6. Papineni, K., Roukos, S., Ward, T., Zhu, W.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002), pp. 311–318 (2002). https://doi.org/10.3115/1073083.1073135
7. Patel, N., Parikh, D., Patel, D., Patel, R.:AI and web based human-like interactive university chatbot (UNIBOT). In: 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 148–150, India (2019)
8. Prajwal, S., Mamatha, G., Ravi, P., Manoj, D., Joisa, S.: Universal semantic web assistant based on sequence to sequence model and natural language understanding. In: 9th International Conference on Advances in Computing and Communication (ICACC), pp. 110–115 (2019). https://doi.org/10.1109/ICACC48162.2019.8986173
9. Rahman, A., Mamun, A., Islam, A.: Programming challenges of chatbot: current and future prospective. In: 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), pp. 75–78, Dhaka, December 2017
10. Sandeep, S.: End to end chatbot using sequence to sequence architecture (2019). https://medium.com/swlh/end-to-end-chatbot-using-sequence-to-sequence-architecture-e24d137f9c78. Accessed 10 June 2019

11. Shukairy, A.: Chatbots in customer service - statistics and trends [infographic] (n.d.). www.invespcro.com/blog/chatbots-customer-service/#:~:text=The%20use%20of%20chatbots%20in,a%20human%20agent%20by%202020. Accessed 12 June 2020
12. Sutskever, I., Vinyals, O., Le, Q.: Sequence to sequence learning with neural networks. In: NIPS, pp. 3104–3112. Curran Associates Inc. (2014)
13. Vaira, L., Bochicchio, M., Conte, M., Casaluci, F., Melpignano, A.: MamaBot: a system based on ML and NLP for supporting Women and Families during Pregnancy. In: Desai, B. (eds.) 22nd International Database Engineering Applications Symposium (IDEAS 2018), pp. 273–277. ACM (2018). https://doi.org/10.1145/3216122.3216173
14. Vinyals, O., Le, Q.: A neural conversational model. In: Proceedings of the 31st International Conference of Machine Learning, France (2015)
15. Zhang, Y., Xu, T., Dai, Y.: Research on chatbots for open domain: using BiLTSM and sequence to sequence. In: Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science, pp. 145–149 (2020). https://doi.org/10.1145/3349341.3349393