Myles Lefkovitz
gth836x / 901929700
CS7637 KBAI – Alternate Project 3 Reflection

The goal of this reflection is to describe how I designed an AI agent to return answers to natural language FAQ questions about this class (CS7637 Knowledge Based Artificial Intelligence).

My agent design follows the examples from two papers:

1. "The Learning Chatbot" by Bonnie Chantarotwong (Chantarotwong, 2006)
2. "DLS@CU: Sentence Similarity from Word Alignment and Semantic Vector Composition" by Sultan, Bethard, and Sumner (Sultan, 2015)

Much of this reflection is based on my previous project reflection (Lefkovitz, 2017 in the references section). A new learning requirement has been added to this project, detailed in the "learning" section.

Agent design based on "The Learning Chatbot"

My original design utilized the approach in (Chantarotwong, 2006). The approach appealed to me because it didn't rely at all on understanding, but simply on case based reasoning. This approach counts the number of unique words matched between the input sentence and each FAQ question in the corpus. Each word in the input sentence is scored based on the proportion of the FAQ question the word represents and the total frequency of the word in the corpus. With this methodology, common words are given less weight when matched between an input sentence and FAQ question.

Consider the example in Figures 1, 2, and 3 below. There are 3 questions in the FAQ corpus (Figure 1). Under each word is the word's weight (the word's proprotion of that question). Questions 1 and 2 have 5 words, so each word is given a weight of 1/5. Question 3 has 7 words, so each word is given a weight of 1/7.

1	What	is	the	FAQ	version		
	1/5	1/5	1/5	1/5	1/5		
2	What	are	the	course	components		
	1/5	1/5	1/5	1/5	1/5		
3	What	is	the	structrue	of	this	course
	1/7	1/7	1/7	1/7	1/7	1/7	1/7

Figure 1. Example FAQ corpus consisting of 3 questions

Word	Total Word Weight	
What	1/5 + 1/5 + 1/7 =	0.543
is	1/5 + 1/7 =	0.343
the	1/5 + 1/5 + 1/7 =	0.543
FAQ	1/5 =	0.2
version	1/5 =	0.2
are	1/5 =	0.2
course	1/5 + 1/7 =	0.343
components	1/5 =	0.2
structure	1/7 =	0.143
of	1/7 =	0.143
this	1/7 =	0.143

Figure 2. Total Weight by Word

Figure 2 shows how each unique word in the corpus is given a total weight. The word "What" appears in all 3 example questions, so its total weight is 1/5 + 1/5 + 1/7 = 0.543. This weight will be used to minimize the impact of words that appear frequently in the corpus and increase the impact of words that appear infrequently.

Figure 3 shows the results of an example input sentence: "What FAQ version number is this?". The algorithm described identifies a score for each word, for each FAQ question in the corpus based on

whether the word matches and the word's total weight. The FAQ question with the highest score across all of the input words is selected by the agent as a match. In this case, the word "What" in the input sentence appeared in all 3 FAQ questions, but the word "is" appeared only in 2 of the FAQ questions. You can see that the score for the word "What" at 0.37, is less than the score for the word "is" at 0.58 in question 1. In this simple example, you can see how the algoirthm applies weight to promote the impact of uncommon word matches and lessen the impact of common word matches.

Words:	**1**		**2**		**3**	
What	1/5 / 0.542 =	0.37	1/5 / 0.542 =	0.37	1/7 / 0.542 =	0.26
FAQ	1/5 / 0.2 =	1.00	0	0.00	0	0.00
version	1/5 / 0.2 =	1.00	0	0.00	0	0.00
number	0	0.00	0	0.00	0	0.00
is	1/5 / 0.342 =	0.58	0	0.00	1/7 / 0.342 =	0.42
this	0	0.00	0	0.00	1/7 / 0.143 =	1.00
SUM		**2.95**		**0.37**		**1.68**

Figure 3. Score by word

My chatbot agent implements this algorithm by initializing with methods to:

- Read the FAQ file into lists of questions and answers,
- Strip the questions of unnecessarily common words (stop words), or words that the algorithm has trouble parsing ("the", "'s" – the possessive article is often considered its own word)
- Identify the unique words in each FAQ question and assign a weight (1/the number of words in that question),
- Create a unique list of words, and
- Calculate the total weight of each unique word

This is very similar to the example in figures 1 & 2 above, but with far more FAQ questions and words.

When an input question is asked of the agent, the agent:

- Extracts unnecessary words (similar to the words extracted from the FAQ questions),
- Finds all words in all questions that match any of the words in the input question,
- Groups all words from the same question and sums their scores,
- Sorts the results and returns the best result based on a couple of conditions:
    - The highest score is higher than a manually defined threshold
    - The highest and second highest scoring questions have different scores

This is similar to the example in figure 3, but with some additional steps to ensure that the matched question meets some minimum relevance.

The paper described only exact matches. I expanded this to include:

- Case insensitivity ("faq" == "FAQ")
- Spelling correction on the input sentence ("verzion" -> "version")
- Singularization of words ("versions" -> "version")

The agent does not include synonym matching.

Agent design based on "Sentence Similarity from Word Alignment and Semantic Vector Composition"

After successfully implementing a chatbot agent as described in the "The Learning Chatbot" (Chantarotwong, 2006) and seeing positive results, I wanted to try implementing another approach. I found a promising and straightforward approach in the "Sentence Similarity from Word Alignment and Semantic Vector Composition" (Sultan, 2015) paper.

This approach is similar in that it scores the similarity between two sentences based on the number of matched words, but the calculation differs from the above method. Instead of weighting words across the entire corpus, word similarity is simply calculated as the number of similar words between two sentences divided by the number of words in each sentence. The simplicity of this method appealed to me, plus I would have an opportunity to implement more complex word alignment, including synonyms identification. The calculation is defined in figure 4. Where $n_c(S^{(i)})$ is the number of content words in sentence $S^{(i)}$ and $n_c^a(S^{(i)})$ is the number of aligned content words in sentence $S^{(i)}$ (Sultan, 2015).

$$sts(S^{(1)}, S^{(2)}) = \frac{n_c^a(S^{(1)}) + n_c^a(S^{(2)})}{n_c(S^{(1)}) + n_c(S^{(2)})}$$

Figure 4. Sentence Similarity (Sultan, 2015)

My agent implements this algorithm by:

- Initializing the chatbot with a method to read the FAQ into lists of questions and answers
- Reading the message and removing unnecessary words ("the", "'s")
- Scoring the input message against each question in the question list (2 * number of aligned words / (number of words in input message + number of words in FAQ question)
- Sorts the results and returns the best result based on a couple of conditions:
    - The highest score is higher than a manually defined threshold
    - The highest and second highest scoring questions have different scores

The paper describes simply counting the number of aligned words. In testing, I found that my word similarity method wasn't very reliable. I decided to decrease the weight of similar words, so that I could keep them in the method, but prevent them from causing the agent to select bad questions. I chose to weigh similar words at 0.4 aligned words. Exact matches, corrected spelling matches, and singularized matches would all maintain 1.0 aligned word weight. This simple change improved my method dramatically, as discussed in the performance section.

Ensemble Agent Design

After submitting project 2 and performing peer review, I noticed a classmate, Miller Pepper, had successfully implemented an ensemble method to allow multiple agents to work together to produce the best response (Pepper, 2017). Following this example, I designed an ensemble agent.

In order to compare results, this agent needs confidence scores from the "Learning Chatbot" and "Sentence Similarity" agents. While the "Sentence Similarity" agent described above returns a ratio between 0 and 1 (that we could call 'agent confidence'), the "Learning Chatbot" method returns a number greater than or equal to 0, as seen in the example used in Figure 3. I needed a way to scale this score.

According to Wikipedia: “Sigmoid functions have domain of all real numbers, with return value monotonically increasing most often from 0 to 1” (Sigmoid Function, 2017). See examples of sigmoid functions in figure 5 below. The sigmoid function seemed like a perfect use-case. After trying a few variations, I settled on the function in figure 6.

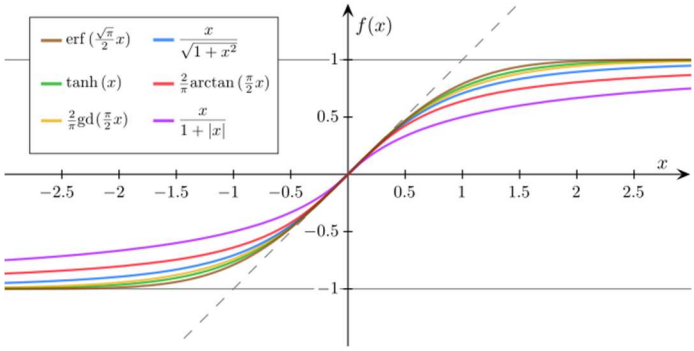

Figure 5. Sigmoid Function Examples (Sigmoid, 2107)

$$\frac{x}{1+|x|}$$

Figure 6. Sigmoid Function (Sigmoid, 2017)

With both agents producing an aligned score between 0 and 1 that I called ‘agent confidence’, I developed an ensemble agent that, when asked a question via the chatbot autograder, calls both agents (“The Learning Chatbot” and “Sentence Similarity”).

The agent then selects a response by:

- Comparing the confidence scores of the agents. If the “The Learning Chatbot” agent’s score is higher than the “Sentence Similarity” agent’s score plus a modifier (0.2), then return the response of the “Learning Chatbot” agent.
- Otherwise, return the response of the “Sentence Similarity” agent, even if it’s “I do not know”.

The modifier was calculated to reduce the impact of the more poorly performing “Learning Chatbot” agent. More discussion is provided in the agent performance section.

External libraries

My agent uses TextBlob to simply solve a variety of problems.

Most importantly for my chosen algorithms, TextBlob makes it easy to reliably break sentences into words. Since my algorithms make comparisons at the word-level, being able to define words with a built-in function was a time saver.

My comparison methods also utilized a few functions to standardize words before comparing them:

- lower() – this function concerts all letters in a word to lowercase
- correct() – this function corrects spelling on input words
- singularize() – this function removes any pluralization on words

Without these functions, easy matches could have been missed. Words at the beginning of a sentence would have been far less likely to find matches, given their difference in case. Chatbot users accustomed to making typographical errors or without perfect spelling would find their questions unanswered. Questions that differ only in scale would remain unmatched.

Additionally, my comparison method uses “synsets”, sets of synonyms for words, in order to identify whether two different words may have the same meaning. I tried a few different WordNet similarity methods on an online word similarity tool (http://ws4jdemo.appspot.com/) to compare scores with key words I felt were important to match across different similarity measures. In the end, I decided to use Wu & Palmer (wup_similiarity), as it produced the most consistent results for the terms I was interested in.

See pip freeze output below (also available in requirements.txt):

```
> pip freeze
certifi==2017.7.27.1
nltk==3.2.5
six==1.11.0
textblob==0.13.0
```

Agent’s KBAI techniques

My approach is based on two KBAI concepts: Case based reasoning and understanding.

Case based reasoning “is the process of solving new problems based on the solutions of similar past problems” (Case Based Reasoning, 2017).

In the class lectures, the professors explain that for Exemplar concepts (concepts like 'Beauty', 'Inspirational', 'Holiday', or 'Saltiness') that cannot be defined explicitly with necessary and specific conditions (like Axiomatic concepts) or by example (like Prototypical concepts) are best understood using methods like case based reasoning (Goel, 2015). “We may have experience with specific holidays, but we cannot abstract them out into a concept with prototypical conditions”. This specific quote comes from Lesson 11 video 14 in (Goel, 2015). The questions in the FAQ corpus represent a concept that is Exemplar. We are given specific examples of questions and their answers, but we cannot define the concept at a more formal level. Case based reasoning is ideal for concepts like FAQ questions.

My chatbot largely relies on Case Based Reasoning. In each agent design, the input question is compared to corpus questions, which act as the existing knowledge cases. The closest matching case is retrieved. This process follows the logic of learning by recording cases, a part of, or predecessor method to, case based reasoning. Since the chatbot is expected to return the exact answers stored in the FAQ corpus, adaptation is not necessary, which means that evaluation and storage are also unnecessary.

Understanding allows us to assign meaning to given words in a sentence. (Goel, 2015). My chatbot uses understanding when identifying whether two words are similar.

Agent’s relation to human cognition

Unlike my agent, humans don’t have a collection of all of the KBAI questions they’ve ever answered, and when asked a new question, search their recorded cases to retrieve the one that most closely matches the new question, based on the words used. In short, the agent does not resemble human cognition very closely.

Humans answer natural language questions by utilizing understanding and common-sense reasoning. When a human receives a new question, they may interpret it into a thematic role system, which aids in understanding. A human, with strong general intelligence and a lot of experience conversing, is well

suited to parse ambiguous words, identify constraints (grammatical, or topical) and resolve that ambiguity. In addition to being able to parse ambiguity, humans are able to understand implications and the desire of the person asking the question. If a question is worded in a way such that details are missing, a human may be able to use common-sense reasoning within the context of their environment to fill in missing blanks. As a simple example: The question "What's the professor's name?" might not be specific enough for an answer, but if a human knows that the question is asking about the KBAI professor, they may be able to answer the question.

Agent Performance

In this second phase of the project, I started with an agent that performed quite well, designed in the first phase. The agent's results are as follows:

Criteria	**Results**
False positive rate	0.26
True positive rate	0.95
Utility	0.69

This agent performed fairly well against the previous rubric. Answering nearly 70% of the questions correctly.

With a new rubric penalizing incorrect answers more severely, I needed an approach that would decrease my agent's false positives, while maintaining its high true positive rate. In developing an ensemble method that rated the sentence similarity agent's results much more highly than the "learning chatbot" agent's results, I produced an agent that would return better results than either agent alone.

I tested my agent against the classmate test script posted on Piazza and discussed in the Slack channel. Out of 200 questions, only 8 represent the new 'learn from the response' types of questions.

Assorted test results:

Method	Score	# Correct	# Incorrect
Agent based on "The Learning Chatbot"	126.5	151	49
Agent based on "Sentence Similarity"	135.5	157	43
Ensemble agent	140	160	40

I'm concerned about how well any of these methods will generalize to the TAs test questions. I feel strongly, however, that the ensemble agent will perform better than either individual agent.

Agent Performance – alternate wording

My agents' method for handling alternate wording is very simplistic and doesn't perform very well. The only alternate wording my chatbot agents support is basic word similarity (without considering any sentence-level constraints, like the part of speech, or other words used in the sentence). Further, my agent identifies word similarity based on the first synset for each word in the pair, as opposed to examining all synsets related to identify the closest matching pair. This was a speed consideration, but it further decreased the utility of the word similarity method.

As discussed above, I used the classmate test script posted on Piazza. My agent is able to capture obviously similar words, like “instructor” = “professor”, and “course” = “class”, but any subtly prevents the agent from matching.

This conservative approach prevents the agent from providing incorrect answers, and with the large penalty of -0.5 points per incorrect answer, this may be a strategic approach.

Learning Agent

This chatbot agent uses learning and memory in two distinct ways.

First, the agent has the ability to save new information to the corpus. This is a new requirement for this project. When material is missing from the corpus in either project 2 or 3 the agent should respond “I don’t know”. In project 3, however, the new question and answer should be added to the corpus and the chatbot agent should be able to respond to similar questions correctly in the future.

Second, the agent keeps a running log of performance during a given run. After an initial warmup period (10 questions), if performance isn’t meeting a predefined standard, then a more conservative approach is instituted by increasing the threshold, decreasing the amount of moderately confident answers, while increasing the amount of “I don’t know”s returned. This approach is outlined in figures 9-11 below. While this approach may not

```
def UserFeedback(self,yesorno):
    self.count = self.count + 1
    if yesorno == "yes":
        self.yes = self.yes + 1
    return
```

Figure 9. User feedback counter

```
if thresholdLearner == 'basic':
    scoreThreshold = .5
else:
    scoreThreshold = .6
```

Figure 10. Adjuster threshold

```
# Learning portion of the agent
if self.count >= 10:
    proportionCorrect = self.yes / self.count
    if proportionCorrect >= 0.90:
        self.learningScoreThreshold = 'basic'
    else:
        self.learningScoreThreshold = 'increased'
```

Figure 11. Learner

Project Suggestions

I was really excited about the first phase of this project (Alt project 2). I think chatbots are a very exciting area of knowledge based AI, and I was planning to work on one as a side project following this semester. The teaching team in this class has a lot of very deep experience with chatbots (in the form of Watson-based TAs), that only increased my excitement. I really appreciate the opportunity to have participated in this alternate project.

I’m happy with the open-ended nature of this project.

I’m a little disappointed with the scope change between projects 2 and 3. If any student completed a successful project 2 they could easily augment that into a successful project 3 in a few short hours,

without implementing new material. I think the learning requirements of that chatbot could have been increased.

I think with a published autograder this project is ready to be assigned to the entire class.

References

Goel, A. Joyner, D. (2015). Knowledge-Based AI: Cognitive Systems. Retrieved from: https://classroom.udacity.com/courses/ud409.

Chantarotwong, B. (2006). The Learning Chatbot. https://pdfs.semanticscholar.org/4f6e/b7a11467c4e90936703142e5c4fdf9031d9d.pdf

Sultan, M. A., Bethard, S., Sumner, T. (2015). DLS@CU: Sentence Similarity from Word Alignment and Semantic Vector Composition. *Proceedings of the 9th International Workshop on Semantic Evaluation.* p. 148-153. http://www.aclweb.org/anthology/S15-2#page=190

Case Based Reasoning. Wikipedia. Retrieved 10/29/17 from https://en.wikipedia.org/wiki/Case-based_reasoning.

Sigmoid Function. Wikipedia. Retrieved 11/26/17 from https://en.wikipedia.org/wiki/Sigmoid_function.

Lefkovitz, M. (2017). KBAI Alternative Project 2 Reflection.

Pepper, M. (2017). Project 2 Reflection.