**“Levi” a ‘semi-intelligent’ AI**

**Project Description, Implementation, Analysis**

Luke Hudspeth CS 072

1. **Proposal:**

Before I began implementing Levi, I proposed the creation of a simple interactive voice to text processor that could handle a few small tasks. My original intuition was that accurately processing the speech to text would prove the most difficult, so I anticipated my project’s success hinging on that functionality. Finally, under the assumption I would reach the point at which Levi should respond to user instructions, I expected semantically interpreting user input to also prove challenging. With such technology already available and in use (Siri, Alexa, Google Home), I knew that the project was feasible, however, I did not know if I could achieve reasonable results through an implementation using our course concepts.

1. **Implementation:**

The first component of Levi I implemented was the functionality of “speech-to-text.” I approached STT initially with the hope of getting into the level of analog audio and trying to parse out the sounds behind different letters, and also using the timing to fragment an audio sequence into words. Such a conversion immediately proved vastly too challenging and time consuming for the scope of this project. Therefore, I opted instead to utilize google cloud’s API for speech to text interpretation. The google API client functionality is quite impressive and helps handle problems with background noise levels, as well as having high accuracy for the translation of STT. The Google API also has the added benefit for future expansion into different languages and translation functionality.

After STT, I programmed the reverse: text to speech. My project implements TTS in two ways, if a user is on a mac, then I have the TTS merely operate via the Mac’s dictation functionality (Using the voice of “Lee” and “Juan” (for Spanish) which one might need to download). Otherwise, there is another google API for TTS that Levi can utilize. Both methods function with the same accuracy and this portion is more a matter of user preference to customize Levi’s voice. Both methods are also capable of being converted to another language, if so desired.

After integrating the google STT API into my python code, I just check to see if the user says “Levi” at some point in a spoken sequence, and if yes, I then process the user input. When figuring out this problem, I thought of several approaches. The easiest would be to just check hard coded strings and if a **key word** that maps to a function is present in the user’s spoken phrase, then just call that function. Such an implementation would no doubt work reasonably well for limited functionality, but when handling more complex tasks Levi needs to be *smarter.* In order to make Levi better suited to actually understand the user’s messages, I expanded upon HW5 and used vector space models to compare similarity of a user’s question to a potential task Levi can perform. I created “documents” corresponding to each of Levi’s function and in those documents I just wrote words that a user might speak to relate to that task. Vector implementation makes Levi more adaptable to varying user input. I.e. A user can say “Levi Google Harry Potter”, “Levi open Harry Potter”, or “Levi search Harry Potter”, and all of those commands will open a google chrome tab searching Harry Potter. Furthermore, this implementation makes it easy to expand and add more functionality or ability for Levi to interpret the user as desired. In order to make Levi answer questions accurately, I modified the word counts within the documents so that Levi will return the most likely response. For example, if the user says “Hi Levi can you tell me the weather?” We want Levi to tell the weather, and not return a greeting, so I incremented the occurrences of the word weather in the weather document, so that the question as a vector will be most similar to the weather vector. Like Homework 5, I implemented stemming so that Levi has an even higher accuracy and so that even more variations of user’s speech might be interpreted. I toyed around with Levi handling a few small conversations, this centered around interpreting a user telling Levi his or her name.

1. **Functionality**

**See videos from Levi.zip file for example functionality.**

1. Greeting: Levi will respond to any combination of “[hello, hi, hey] Levi” and ask the user for his or her name. After which the user responds, Levi will greet the user back with the proper name.
2. Date/Time: If the user asks Levi for the “[day, time, date]” Levi will look up the current date and time and reply to the user.
3. Weather: If the user asks about [weather, rain, cloud, sun] then Levi will read the current forecast for Hanover, NH.
4. Spanish: If the user asks about [language, Spanish, speak] or variants of the such, then Levi will respond in Spanish.
5. Music: If the user asks a question with [music, play, song] or variants on those words, Levi will play a song for you.
6. Google: if the user asks [google, search, open] Levi will try to extract the search from that sequence of audio and open a google search for the user.

Through the use of vector space modeling, I am able to improve Levi’s understanding of user input. Coupled with stemming, Levi can actually handle many potential variants of a question or user input to yield the correct response. The functionality I implemented is easily modifiable, and with more time I could likely sub-parse the functions to handle even more complicated sequences. I.e. “What will the weather be next week in New York?” is a potential improvement.

1. **Results/Analysis/Evaluation**

The google STT API was essential for this project’s success. Had I stuck to my original plan of trying to parse analog audio into text, I would have very likely failed. To have succeeded in that task would have required a much more sophisticated understanding of sound waves as well as the vast data to output words from that sound correctly. With the google API, Levi has high accuracy of correctly translating user speech into text. I have included several examples of Levi failing to translate accurately, but on the whole I was pleased with this component. I would say Levi can translate on par with Siri or Alexa, which again I demonstrate in the videos. So for the evaluation of speech-to-text processing, Levi is a success. As far as the functionality of Levi to actually understand the user, that is limited to whatever can be implemented via a vector space model. Levi is not an actual AI, in that he does not learn from user interaction, he can only understand questions that fall within the vector space model’s parameters. However, even with that limitation I am pleased with Levi’s ability to understand questions and even variants of words in those questions (due to stemming). \*see videos, “longexample.mov”

The flaws of Levi are obviously that sometimes he will not pick up on a user’s question because of “Levi” not getting translated correctly (sometimes goes to “Levi’s”). Perhaps a better name would actually function better, and I did play around with this and saw varying degrees of success. Furthermore, Levi is sometimes quite slow to respond (usually a delay between 2.5-5 seconds), and this response time also varies (which I believe is due to internet speeds and background noise).

1. **Future Work**

Were I to expand Levi more, I would ultimately try to speed up the response time, add more functionality, and implement a neural network to handle the interpretation of questions. To improve the response time, I would either explore more API’s for speech-to-text processing, or I would try to optimize my code so that the Google API can function at it’s optimal performance. I have noticed that internet speed seems to have some effect on Levi’s response interval as well as background noise, mic quality, and even the name “Levi.” Interestingly, when I make the *name* “Alexa” I actually see that the program responds better, so clearly words with certain consonants play an effect. Adding more functionality is actually one of the easier components of this project; I can simply expand my vector space model of vector words, add a new file to hold the words related to that functionality, and implement the function in code. More functionality is only limited to creativity and the capabilities of python.

Finally, to take Levi to a much more sophisticated iteration would be to implement some sort of neural network. A neural network would adapt to a user’s preferences and ultimately be far superior to any vector space model as the neural network is constantly growing and making new associations with use. Obviously this would be a highly involved task and require a huge amount of data to train such a network. For now, Levi is limited to his current skillset.

To conclude Levi’s advancement, I would also consider implementing different language abilities. I show a small example of a Spanish response, but this could fairly easily be converted and with an option, Levi could do everything he already can in a variety of different languages.