Entity Referring Classifier for Virtual Assistants

Lingxi Li[1], Li Xu[2], Zachary Polak[3], Tudor-Cristian Foca[4]

Abstract

There are a lot of smart assistants available in the market, but they all require wake phrases like "hey Siri" or "hey Alexa" to activate them. In daily life, humans are communicating with each other just by using names. However, when humans are talking to a bot, they are using redundant wake phrases. So, with the goal of delivering a better user experience, we want to eliminate the use of wake phrases to call a virtual assistant. Our goal was to make a classifier that can detect whether the name of an entity is being called in a sentence or if the entity is just being mentioned. To achieve that, we created a custom dataset, annotated the dataset, and trained a binary classification model based on our finalized dataset. In this project, we used a pre-trained BERT model used for tokenizing, and another BERT transformer used for the classification task.

[1] lingxili@umass.edu, *University of Massachusetts Amherst*
[2] lixu@umass.edu, *University of Massachusetts Amherst*
[3] zpolak@umass.edu, *University of Massachusetts Amherst*
[4] tfoca@umass.edu, *University of Massachusetts Amherst*

Contents

Introduction

Have you ever used a voice assistant like Siri or Alexa? All of the smart assistants available in the market require wake phrases like "hey Siri" or "hey Alexa" to proceed with a command. It works, but it is awkward to communicate like that. We want to improve the communication between two objects so they can communicate like humans, the objects could be voice assistants and humans or robots who have a name. The key point is: when should the virtual assistant respond, in other words, how do the virtual assistants know we are talking to them? They should not simply detect if the name exists in a sentence because we may say the name in daily conversations but do not mean to wake them. Therefore, our goal is to find a way that could distinguish the "calling" and the "mentioning".

The definitions of these two terms come from scenarios that a virtual assistant should have the ability to distinguish them. The term "calling" means that the sentence contains the target entity's name, and the speaker is talking to the target entity, so the target entity should respond to the conversation. In contrast, the term "mentioning" means that the sentence contains the target entity's name, but the speaker is actually talking to another person and just mentioning the target entity, in this case the target entity should not respond to the conversation. With this approach, we can talk to the voice assistant as we normally talk to a human. It would improve the user experience for any voice assistants related product.

We used dependency parsing and neural networks for this project to train our model. We want to know how accurate the output result is, and how we can improve further.

1. Related Works

After performing many different searches, we could not find any papers directly related to our task at hand. This may be the case because people are not actively working on a task like this. It may also be the case that work is being done on this task but the research is private. However, we did find some relevant papers that we were able to refer to while building our classifier.

1. **A Context-aware Natural Language Generator for Dialogue Systems**

   https://aclanthology.org/W16-3622.pdf

   This paper focuses on a "context-aware" dialogue system. Our task also has to do with a "context-aware" dialogue system. Although in the paper they are focused on tailoring machine output to user input. In our system we are more focused on our machine correctly understanding user input.

2. **Natural Language Generation in Dialogue Systems**

   https://aclanthology.org/H01-1055.pdf

   This paper is similar to the first one in relation to being "context-aware". But again, they focus more on tailoring machine output to user input, and less about classifying user input. Despite this difference the paper is still somewhat related to our task.

2. Data

2.1 Collection and Annotation

We did not use an external dataset for this project. We collectively decided to build our own dataset. This was because we were unable to find an external dataset that met our specific needs. The data that we were looking for were sentences that contained names which were either 'calling' or 'mentioning' an entity. We tried to have an even balance between the two types of sentences that we generated. All together we generated a total of about 1,000 sentences for our model to use. All of which we thought of ourselves.

2.1.1 Data Collection

In order to successfully build this dataset ourselves we needed an organized system to do so. Therefore, we decided to complete our data collection on a Discord server. We utilized a specific channel on this server called 'Data Collection'. The only messages sent in this channel were sentences that we wanted to include in our final dataset. Each sentence followed the same type of structure: (sentence with entity name) - (entity name). It was necessary to label the entity name so that it was very clear during annotation who the subject of the sentence was. A screenshot of our data collection system is included below at **Figure 1**.

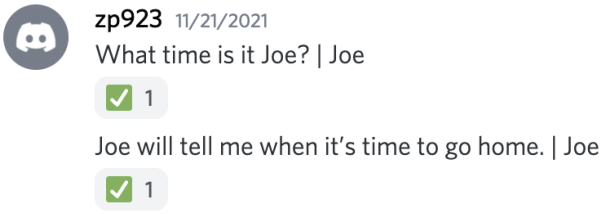

Figure 1. Data Collection System

2.1.2 Data Annotation

After the data collection procedure came the annotation process. Once the sentences were entered into the 'Data Collection' channel, the system automatically assigned each of the sentences to every member of our group to annotate. Annotations were completed in our Discord server by interacting with a bot. To start annotating we would send a message to the bot saying "annotation" and the bot would reply with one sentence to analyze, followed by three different reaction icons. One for the label 'calling', one for the label 'mentioning' and one last label 'snooze'. If we were unsure about how to label a sentence we could hit 'snooze' and come back to the sentence at a later time. We could also track our individual annotation progress by messaging "progress" to the bot. The bot would then reply with how many sentences were annotated, how many sentences there were in total, and give us our progress in the form of a percentage. A screenshot of this annotation system is included below at **Figure 2**. Utilizing these two systems for data collection and annotation greatly improved the efficiency of this process. It would have been much less efficient had we used some other system such as a spreadsheet for the data collection and annotation.

2.1.3 Data Exportation

Lastly, in order to work with the dataset that we had built, we had created a specific channel in our Discord server called 'Data Export'. This channel utilized export commands to export different data. For example, there was a command to export just the sentences in the dataset. There was also another command to export the sentences with their annotations. Having different

Figure 2. Our Annotation System

export commands allowed us to be able to work with only the specific parts of the dataset that we needed at that time. A screenshot of this data export system is included below at **Figure 3**.

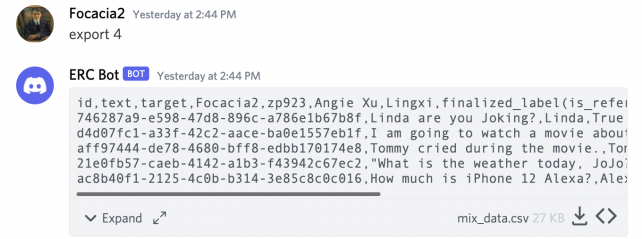

Figure 3. Exporting data from system

2.2 Final Dataset

At the end, we have collected 1,000 sentences for our dataset. That data was in the format of *(text - entity - label)*, where text is the sentence and label is the named entity in the sentence. In other words, label is subject of the sentence. The sample dataset is shown in **Figure 4**.

Overall our dataset is diverse. This is due to the fact that these sentences all come from examples used in daily life. With all group members contributing to the dataset, we were able to include a very wide variety of sentences.

3. Methods

3.1 Dependence Parsing Attempts

Since we are trying to detect if the named entity in a sentence is being called or is being mentioned, we tried to use Dependence parsing from spaCy to analyze this feature.

After parsing dependencies for all of the data entries with spaCy dependency parsing, the results were as showing in **Figure 5** and **Figure 6** for the dependencies of the targeted entity:

	text	target	finalized_label(is_referring)
0	Linda are you Joking?	Linda	True
1	I am going to watch a movie about Einstein.	Einstein	False
2	Tommy cried during the movie.	Tommy	False
3	What is the weather today, JoJo?	JoJo	True
4	How much is iPhone 12 Alexa?	Alexa	True
5	Set 20 minutes timer Bill.	Bill	True
6	Who is Bill?	Bill	False
7	Is this your new Alexa?	Alexa	False
8	Move Linda's bed to the other room.	Linda	False
9	Do you like JoJo?	JoJo	False
10	How is Logan's morning?	Logan	False
11	wow okay danny, you are right	danny	True
12	what do you like about Trump?	Trump	False
13	Who do you think will be the next President af...	Biden	False
14	How many votes did Biden get, Tommy?	Tommy	True

Figure 4. Sample Dataset

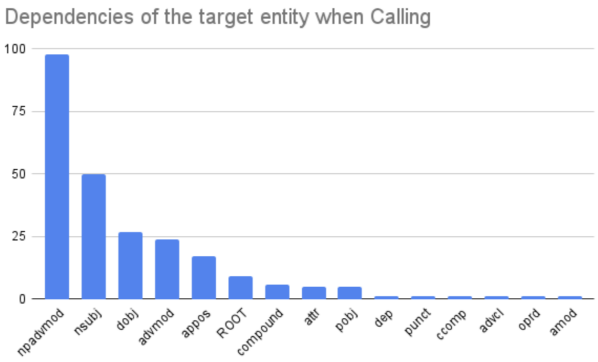

Figure 5. Dependency Parsing Calling

As of the limitation of the dependency parsing and the complication of grammar, we did not expect a promising result.

And since we did not find a proper way to combine the Dependency Parsing and the PyTorch model (or BERT transformer), we decided to not follow this path at the end.

3.2 Pre-trained BERT and BERT transformer

In general, we used a pre-trained BERT model for tokenizing the sentences and a BERT transformer (trained with our dataset) for the classification task.

First, we pre-processed the data and only select the finalized data to train because the size of our dataset was increasing continuously. We split the data in the following setup and shuffle them every time before training:

- Training — 85%
- Validating — 5%
- Testing — 10%

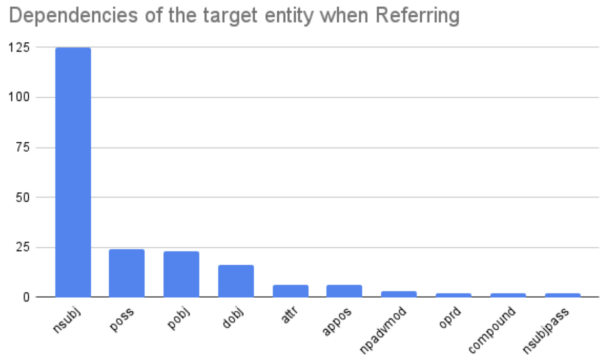

Figure 6. Dependency Parsing Mentioning

Our choice of BERT tokenizer was *bert-base-uncased*, so we were not caring about the case of text. The pre-processing procedure included:

- Lowercase.
- Mask the given target entity name.

Because we had our criteria on making the dataset, so we did not need to normalize our data before using. Then, we replaced the target entity name with the mask token of BERT tokenizer and we used it to tokenize the sentence and filling up the rest of space with offset token.

The training is based on the pre-trained *BERTforSequenceClassification* transformer. In every training iteration, we put our data into this model and move forward. Because we did not have foundations on Neural Network, we were not able to optimize the training process. We follow the document of HuggingFace transformer and PyTorch library to make adjustments and fine-tune.

4. Results

4.1 Previous ERC (PyTorch RNN Model)

Before using the BERT transformer for classification task, we tried training a PyTorch RNN model from basis and reported that in our progress report. But it did not work well in the end, so we decided to pick BERT transformer as our final solution because it turned out to be very good after evaluating.

4.2 Final ERC (BERT Transformer)

The BERT transformer *BERTforSequenceClassification* was pre-trained, which gave us a better basis on clustering similar sentences and structures even if our dataset was not large enough. Based on that, we trained it with our dataset, which included the label of each sentence, so the transformer could learn from our dataset and then predict the result well.

After adjusting variables and learning rates, we concluded that when learning rate was set to $2e^{-5}$ and epoch number was set to 5, we had the best performance of the training. Then, by saving the checkpoint and metrics during training, we got the loss plot of our trained model. It is shown in the **Figure 7**.

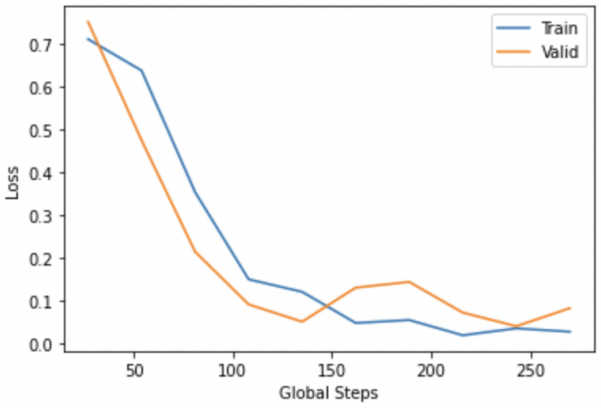

Figure 7. Loss plot of ERC Model

After evaluating the model we trained by using test dataset, we draw the confusion matrix by using *seaborn* library. The graph is shown in **Figure 8**.

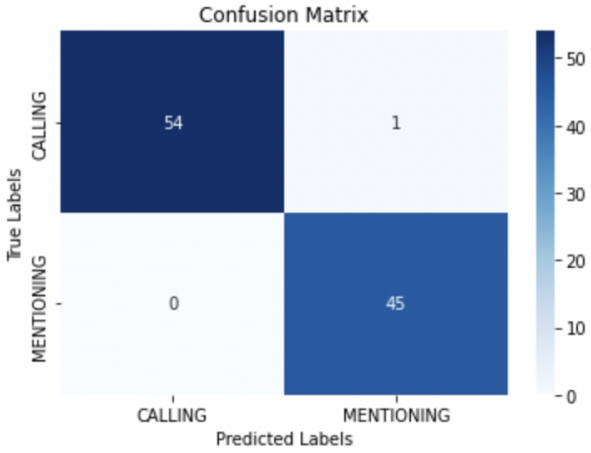

Figure 8. Confusion Matrix of ERC Model

By using *Sklearn.matrics* library to analyze the accuracy of our model from testing dataset, we got the following evaluation report shown in **Figure 9**.

Finally, we have delivered our model to the HuggingFace space for live demo, feel free to try it out: https://huggingface.co/spaces/erc/entity-referring-classifier

```
SK-Learn Evaluation Result:

              precision    recall  f1-score   support

           1     1.0000    0.9818    0.9908        55
           0     0.9783    1.0000    0.9890        45

    accuracy                         0.9900       100
   macro avg     0.9891    0.9909    0.9899       100
weighted avg     0.9902    0.9900    0.9900       100
```

Figure 9. Evaluation Result of ERC Model

5. Discussion and Future Works

Our project is unique in the fact that we do not believe that any work is being done on this particular topic already, or if there is work that has been done, the research is at least not being made public. It's an interesting idea, being able to use an assistant without the use of a wake word. For our project we implemented this on a chat bot, and proved to be successful. However, our work can most certainly be expanded upon.

For example, this could also be implemented on a voice assistant. A voice assistant with this feature may be much more useful than a chat bot. Our project provides a baseline to create more extensive projects like this. While we used a custom dataset made up of about 1,000 sentences, a significantly larger dataset can be used in order to successfully train a voice assistant capable of not using a wake word.

Additionally, Lingxi is going to expand the project into Simplified Chinese in the future, implement it into a Discord Bot, and deliver it into medium-sized channels to see if users feel better in communicating with bots in this way and if it works well in conversations in practice. If it works well, we may try to combine it with the speech-to-text to analyze the experience in the context of voice, which has a lot more complicated scenarios.

We are very curious to see if this is an idea that technology companies will implement in the future.