Order Assist - Speech to Text

Rishabh Agrawal
Khoury college of Computer Science
Northeastern University
Boston, USA
agrawal.ris@husky.neu.edu

Rutu Nanavati
Khoury college of Computer Science
Northeastern University
Boston, USA
rutunanavati@ccs.neu.edu

Viral Patel
Khoury college of Computer Science
Northeastern University
Boston,USA
patel.vira@husky.neu.edu

***Abstract*—The key focus of this project is to speed up the process flow of ordering at restaurants. This is especially useful in a drive-thru and quick-service restaurant in order to get through the queue quickly and reduce the overall error rate both in the ordering process. Currently, a lot of resources and manpower are wasted by a person taking manual orders. Eventually, this could replace the person taking orders and save time and money for both user and business end. Our product QSR assist enables businesses to offer a speedy AI-enabled experience while serving increased appetites of their patrons.**

I. Introduction

The market value from quick-service restaurants (QSRs), fast food chains, drive-thru establishments, and deliveries are estimated to be worth 273 billion. Technology plays a critical role in keeping this market thriving by connecting consumers with their preferred fast-casual, fast food and QSR restaurants. Voice-based search technology has already become a cornerstone of consumers' search activities, with some companies estimating that one billion voice searches are performed each month. It is predicted that nearly half of all searches will be voice-based by next year.

Consumers are not limiting their voice-based searches to just home-integrated systems like Amazon Echo or Google Home, either. According to PYMNTS Digital Drive Report, 53.3 percent of commuters use voice assistants on their smartphones while they drive. The report also found that an increasing share utilizes voice capabilities during their trips, with nearly 17 percent of commuters using vehicle-integrated voice assistants to connect to the internet while they drive. Ordering food and finding restaurants are vital parts of these commuters' routines, and 14 percent used voice assistants to order meals for delivery while they were driving.

Natural language processing (NLP) is a form of artificial intelligence that focuses on analyzing human languages to draw insights, create advertisements, aid you in texting and more. The most difficult part of NLP is understanding or providing meaning to the natural language that the computer receives.

First, the computer must take natural language (humans speaking English/Spanish, etc.) and convert it into artificial language. This is what is called speech recognition, or speech-to-text. Once the information is in text form, the Next step is Natural Language Understanding(NLU) which is where you try to understand the meaning of that text.

In recent years, we have witnessed a revolution in the ability of computers to understand and to convert natural speech, especially with the application of deep neural networks (e.g., Google Home Mini, Alexa, etc). But in particular, these automated speech systems are still struggling to recognize simple words and commands. They don't engage in a conversation flow and force the user to adjust to the system instead of the system adjusting to the user.

The Quick Service Restaurant (QSR) Assist aims to accomplish the above. It would start by hearing the order, transcribe it to text, map it to the items in the menu and finally create an invoice. From the converted text we would be able to classify the intent of that part of the speech such as adding, deleting, modifying and canceling an order. The extracted information is then used for keyword extraction and invoice generation.

II. Dataset

We have used the google dataset Taskmaster-1 [7]. This dataset consists of 13,215 task-based dialogues in English. We will be using the conversation from only 2 of the 6 domains (ordering pizza and ordering coffee drinks) which are relevant to Quick Service Restaurants. Dialog annotations are based on the API calls associated with each type of task-based dialog. Each conversation was annotated by crowd-sourced workers they were made to believe that they were communicating with assistants. Both annotations are included in this collection. The ontology.json can act as a reference to validate the invoice generated in the end.

III. Methods

The project workflow as shown in Fig: 1 starts by taking a voice input from user. This input file is then uploaded to google cloud and then converted to .flac file format. The audio file is processed by google cloud functions to convert that audio into text format using deep neural networks. From the generated speech, the next task of intent classification is done by wit.ai whereas Google NLP API takes care of object recognition and quantity generation. The processed information is then matched to an existing restaurant database for final invoice generation.

The key functionalities of QSR Assist is as outlined below:

* Credits to Quantiphi for the project idea and motivation to use Google Cloud Platform.

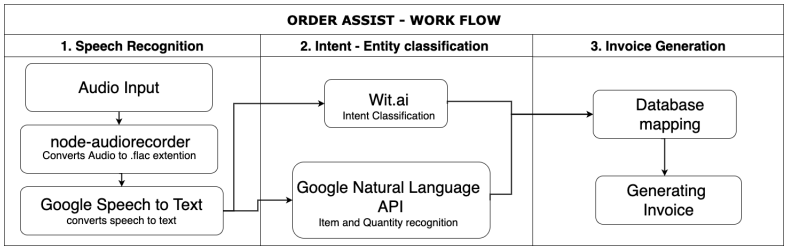

Fig. 1. Work Flow Order Assist

A. Speech Recognition

The first step is to receive the audio input (16000 Hz) and store it in (.flac) file format.As the audio file is uploaded in the cloud it triggers Google Cloud Functions which is a serverless execution environment for building and connecting to cloud services such as Google Cloud Speech API which converts the audio to text by applying deep neural network models. The API recognizes 120 languages and variants that can support ones global user base. Speech-to-Text can transcribe text from streaming audio or prerecorded audio in real-time.

Google cloud function is a compute solution for writing functions that are automatically triggered by certain cloud events. Functions are written in Node.js, and the type of event that will trigger each function is specified. To get our audio transcription and timestamp data we have added a Cloud Function to transcribe audio which will be triggered whenever a flac file is added to our audio bucket. For this function we've instantiated a speech client with google-cloud Node and then built our transcription function on top of that. This function will request Cloud Speech API and our client will kick off a long running speech operation and returns the final JSON transcription results when it finishes.
As described in Table Appendix 1, Google Speech to text has three types of API requests based on audio content. Here we are using Synchronous Recognition (REST and gRPC) that sends audio data to the Speech-to-Text API, performs recognition on that data, and returns results after all audio has been processed. Synchronous recognition requests are generally limited to audio data of 1 minute or less in duration. Google Cloud Speech-to-Text can return well-recognized text from audio stored in a file. It's capable of analyzing both short-form and long-form audio. Cloud Speech-to-Text is tailored to work well with real-life speech and can accurately transcribe proper nouns and appropriately format language (such as, dates, phones numbers). It handles noisy audio from many environments without requiring additional noise cancellation. It can also accurately punctuate transcriptions (e.g., commas, question marks, and periods) using machine learning.

B. Intent-Entity Classification

Intent-Entity Classification part of the pipeline is for understanding the intentions of humans and extract any relevant information to take some action.

It uses the text transcribed to recognize the entity(object) and intent(action). eg: "Please add olives to the pizza". Entities: Pizza. Sub-Entities: Olives. Intent: Customize

1) Intent Classification: Intents and entities are key concepts of any answering system. Intents create links between conversation and action of the application. We have tested the performance of Wit.ai as well as dialogflow for intent classification. We observed that Wit.ai could perform significantly better to identify the intent.

When Dialogflow receives a user request, it first matches the known intents or else it will give a default intent. It is like a pre-defined flow on giving mandatory and optional categories and it makes sure that the user fills all the mandatory categories. This can be useful as there should not be any missing values. However, disadvantages for this method are is that it cannot control entity matching with intent if the entity already exists. This method is only useful when building a chatbot.

Since our system is not a two-way communication system, Wit.ai works better as it works by creating stories. It treats stories as a graph of user intents. As the story concept works well with our use case, it allows us use any predefined entities as well as to create our entities. [10] We used wit.ai [6] API to classify the intent (add, modify or update) of the conversation. Refer to the results from wit.ai [6] API from Table I: Intent Classification.

2) Entity and Quantity Recognition: The object entity recognition is done to identify and classify items. This can be done by tokenizing and classifying the words. Item and Quantity recognition is achieved by using natural-language [5] API by Google cloud. We compared results for wit.ai and google cloud natural-language API for entity and Quantity recognition. However, google outperformed Wit.ai and the results for the same are discussed in the results section below (refer Table: IV).

As we can see the items are identified correctly from the statement along with the value. We are using entity analysis from the language module to perform this task. It identifies the value that is the entity and the type of entity. It can also recognize the quantity of the order. The google API achieves this by using part of speech tagging and dependency parsing and it also helps us to map quantity with the object. The words are then lemmatized to map to the order menu. We can observe the working example in the appendix Fig Appendix 2

C. Invoice Generation

For invoice generation, we are currently using a local database that stores the list of items associated with the price. The database can serve as a menu for the items that a restaurant has to offer. Entries from this database can be added, deleted or modified.

The items recognized through the API from the conversation are queried in the local database for creating the order and generating the invoice. At this point, the order is mapped to the items in the menu by the unique identification code which would fetch prices and generate invoices in the end. The fetched prices are further passed through a python function that would calculate item-wise prices as well as the total price of the order. This can be given out in any format. Refer to Table V, for checking results of final order invoice.

Sentence	Confidence Score	Intent
Can i order five Burger and a fries	0.7741	Add
I would like to order one large pizza and one soda	0.989	Add
Please remove the burger from my order	0.995	Remove
Can you swap the burger for a cheese pizza	0.987	Update
I would like to get a coffee too	0.9744	Add

TABLE I
INTENT CLASSIFICATION RESULTS

	Add	Delete
Precision	0.80345233	0.20322
Recall	0.86352332	0.052345
F1-Score	0.83240546	0.083247

TABLE II
DIALOG FLOW API PERFORMANCE RESULTS

	Add	Delete
Precision	0.86910995	0.5
Recall	0.94318182	0.09090909
F1-Score	0.90463215	0.15384615

TABLE III
WIT.AI API PERFORMANCE RESULTS

IV. RESULTS

The model was generated by iteratively running various APIs. First we unit tested the model and the results of those are as explained below: We alternatively used Wit.ai and dialog flow for intent classification. To test the intent classification results we manually labeled 2000 lines from the taskmaster conversations with Add and Remove labels as the dataset didn't have labeled intents. These intents were classified using APIs (Wit.ai and Dialogflow) on the same labeled conversations.

By referring to Table I, we can observe that the trained intents have been classified with a minimum of 0.70 confidence score but most of the intents have been classified over 0.95 confidence score using wit.ai [6]. The test results for the taskmaster conversation for dialogflow are given by Table: II and for Wit.ai are given by Table: III.

Similarly, item and quantity have been recognized using natural-language (Google cloud platform) API and Wit.ai. The accuracy for both the methods is shown in Table: IV. The accuracy for GCP (Google Cloud Platform) is much higher than that of Wit.ai

	Wit.ai	GCP
Accuracy	0.9087	0.9956

TABLE IV
ENTITY RECOGNITION RESULTS

The final results are shown as key-value pair of item and quantity in each iteration also the intent in each iteration is given in Table: VI. It can be observed that for each sentence the intent is recognized and then the corresponding item is either added or deleted from the dictionary. The first sentence is an example of an "Add" intent where Fries and Burger are added to the order. However, the 3rd sentence is an example of a "Delete" intent where a single quantity of burger item is removed from the order.

The results from the final iteration are returned which is used for preparing the order and generating an invoice. In Table V, we observe the final list of items with their quantity and price associated with it. This table is generated by mapping it with the price information from the database.

Item	Price	Quantity	Amount
burger	10.0	4	40.0
fries	4.5	6	27.0
pizza	10.0	1	10.0
soda	3.0	1	3.0
coffee	5.0	1	5.0
coke	3.0	2	6.0
sandwich	5.0	7	35.0
	Total	22	126.0

TABLE V
FINAL INVOICE

V. DISCUSSION

It was observed that the Google Speech to Text API works better with chunks of voice data rather than a whole long audio file. So it would be better to process each chunk with the content of only a single sentence in it. As the intent recognition API can only identify single intent in a given sentence. Also, processing long audio with multiple sentences worth of data would create problems in getting one clean intent from a sentence as the conversation will be back and forth.

As Wit.ai Table III has slightly higher values than Dialog Flow Table II for both the precision and recall of both "Add" and "Delete" intents. So it is preferable to go with Wit.ai for this particular dataset. It turned out that in most cases

Speech To Text Sentences	*Intent*	*Item Mapping*
Can i order five Burger and a fries	Add	'burger': 5, 'fries': 1
I would like to order one large pizza and one soda	Add	'burger': 5, 'fries': 1, 'pizza': 1, 'soda': 1
Please remove the burger from my order	Remove	'burger': 4, 'fries': 1, 'pizza': 1, 'soda': 1
I would like to get a coffee too	Add	'burger': 4, 'fries': 1, 'pizza': 1, 'soda': 1, 'coffee': 1
Hi I would like to order one cheeseburger and five fries and three Coke and 7 Sandwich	Add	'burger': 4, 'fries': 6, 'pizza': 1, 'soda': 1, 'coffee': 1, 'coke': 3, 'sandwich': 7
Remove one coke too	Remove	'burger': 4, 'fries': 6, 'pizza': 1, 'soda': 1, 'coffee': 1, 'coke': 2, 'sandwich': 7

TABLE VI
STEP BY STEP : ITERATION OF THE MODEL

"Update" intent is just a combination of add and delete so no intent corresponding to update was added in the final model. As seen the recall and precision values of delete is quite low for the test because of less training data. Regardless of that during our live testing the model identified delete instances quite well. The final model only uses the intent Add and delete as shown in Table VI.

On average Wit.ai recognizes an intent with a confidence score of 0.92 which is quite high and accurate. For entity recognition, it turns out that Google NLP API does a lot better job at entity recognition than Wit.ai. From Table IV we can see that the accuracy rate of GCP is about 10% more than Wit.ai, which is quite high. Here dependency recognition is used for mapping entities with their quantity by GCP without any extensive training. So overall, GCP is used for all of entity, quantity and dependency recognition.

There are several ways or methods which could be explored to improve the accuracy of the model even further. If more labeled data with accurate intents can be obtained or manually labeled then the accuracy of the model might go even higher as the model will have sufficient instances of each intent to build the model. Another method that can be tried is to do multiple intent recognition. If a model can be trained which recognizes different intent for different parts of sentences then breaking the audio or taking the input in chunks can be avoided and the process can be made much more smoother so that the audio can be processed even in a continuous stream.

Moving Forward, another goal which could be worked on or improved upon would be to build a more robust sub-entity recognition model as it can help customize the order by user preferences.

STATEMENT OF CONTRIBUTIONS

Rutu Nanavati : Entity Analysis, Researched on APIs from Google and wit, Visualizations and flow diagram presentation for report, order aggregation presentation, proposal, Manual Data Labeling report.
Viral Patel : Preliminary Exploration, Market Analysis, Entity recognition, Visualizations, Manual Data Labeling. Flow visualization for project, Report, presentation, proposal, researched on rasa NLU Pipeline
Rishabh Agrawal : Created 'MakeMenu' class, Intent Classification, Order generation, Report, Proposal, Presentation, SQL mapping, preliminary analysis.

REFERENCES

[1] https://www.seelevelhx.com/2018-drive-thru-study-key-findings-full/
[2] https://github.com/voxy/django-audio-recorder/blob/master/README.md
[3] https://www.audacityteam.org
[4] https://ai.google/research/pubs/pub44631
[5] https://cloud.google.com/natural-language/
[6] https://wit.ai/
[7] https://ai.google/tools/datasets/taskmaster-1
[8] https://cloud.google.com/text-to-speech/
[9] https://www.topbots.com/ai-nlp-research-papers-acl2019/
[10] https://tryolabs.com/blog/2017/01/25/building-a-chatbot-analysis–limitations-of-modern-platforms/
[11] https://www.pymnts.com/

APPENDIX

Content limit	Audio Length
Synchronous Requests	$\sim$ 1 Minute
Asynchronous Requests	$\sim$ 480 Minutes
Streaming Requests	$\sim$ 1 Minute

TABLE Appendix 1
API REQUESTS

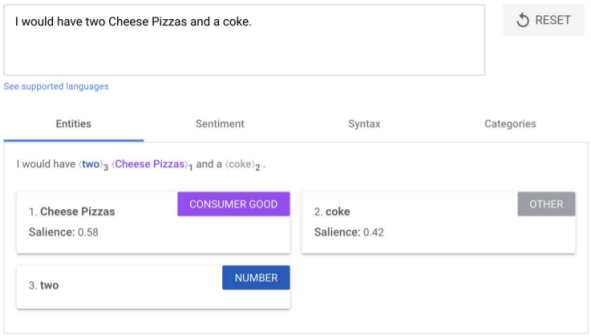

Fig. Appendix 1. Entity And Quantity Recognition Example

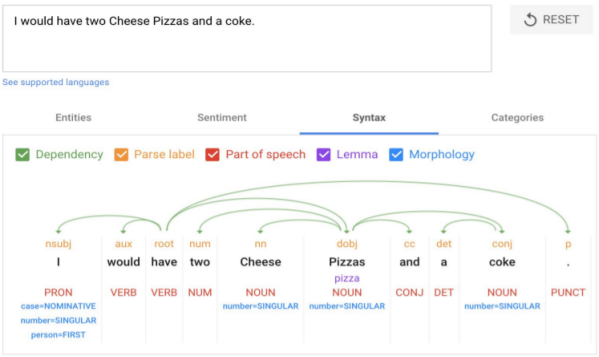

Fig. Appendix 2. Dependency Recognition Example