

# CSE 535 - Question Answering Information Retrieval System

## Ping-Her

## Project Report

ARAVIND MAHADEVAN SUBRAMANIAN 50207700

EMAIL ID : as386@buffalo.edu

MOHIT ARVIND KHAKHARIA 50207819

EMAIL ID: mohitarv@buffalo.edu

SUDARSHAN VENKATARAMAN 50208037

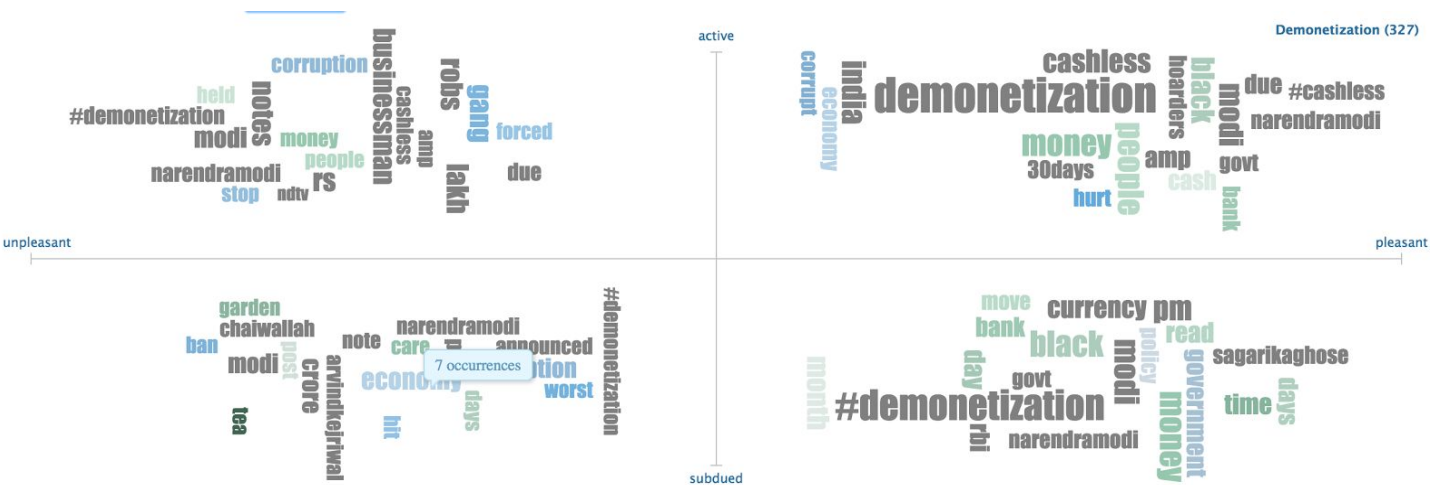
EMAIL ID: sv79@buffalo.edu

SUGOSH NAGAVARA RAVINDRA 50207357

EMAIL ID: sugoshna@buffalo.edu

<http://pingher.com>

### Related #Hashtags for which the content was collected



## Team Members & Roles

### Aravind Mahadevan Subramanian

- Natural Language Pre-processing of collected tweets in JSON using nltk (python), such as :
  - Sentiment analysis of tweets
  - Named entity tagging of proper nouns in tweets
  - Part of speech tagging of words in tweets
- Participated in research on APIs to use for question processing/chatbot
- Tested out AlchemyAPI on efficiency and limitations on NLP
- Participated in overall flow design of the QA processing

### Mohit Arvind Khakharia

- Developing User Interface (Material Design)
- Implementing Google Voice Recognition
- Implementing Google Synthetic Voice Generation
- Tweet Collection
- Creating a web app using Django(A python web framework) and deploying the same.
- Registering the domain (<http://pingher.com>) and DNS configuration to link to the AWS instance.

### Sudarshan

- Tweet Collection.
- Proof of concept on existing AI Api's.(API.AI)
- Deduplication of twitter data before indexing in solr.
- Pre-processing of twitter feed to extract appropriate fields.
- Participated in User-Stories and entities creation in WIT.
- Validation of Queries queried by users.

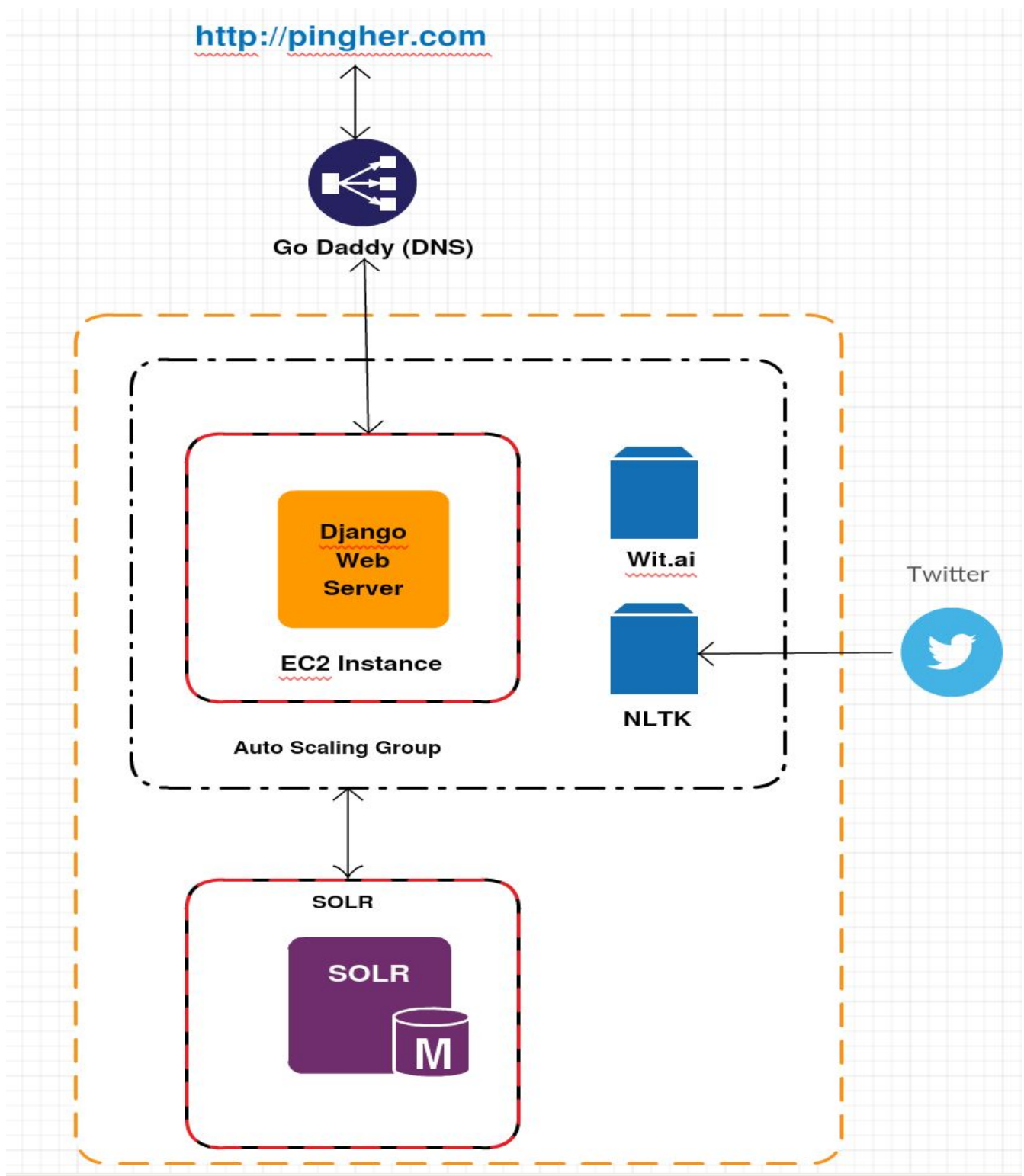
### Sugosh Nagavara Ravindra

- Design User stories (Questions) on wit.ai
- Design base case scenarios.
- Train the bot by validating question-entity relationship
- Entity specific handler functions
- Query formulation

## Project Development Fashion

- Agile, Using User Stories implemented on (<http://pingher.slack.com>)

## High Level Technical Architecture



## Preprocessing and Indexing in SOLR

After the tweets have been collected using HBC Twitter API , preprocessing of the tweets was mandatory as the tweets involved a lot of junk tweets (Retweeted tweets + Tweets conveying the same meaning). Retweeted tweets were removed by the usual method of only indexing tweets that didn't begin with "RT@". In order to remove texts conveying the same meaning , "DEDUPLICATION" was done. Deduplication is the process of indexing by making the **tweet\_text** as the **unique\_key** (instead of the usual tweet\_id).

Pre-processing the tweets also involved collecting the appropriate fields from the twitter field. The fields collected are namely the following : "tweet\_text" , "location" , "media\_url" , "screen\_name" , "created\_at" , "favourite\_count" , "retweeted\_count" and so on.

The purpose of saving those fields were to retrieve appropriate tweets for the queries fired by the user. Say for instance if the query was to "Show me an Image from <screen\_name> on topic" , then the the appropriate fields would be chosen to query and return the most relevant tweet for that query.

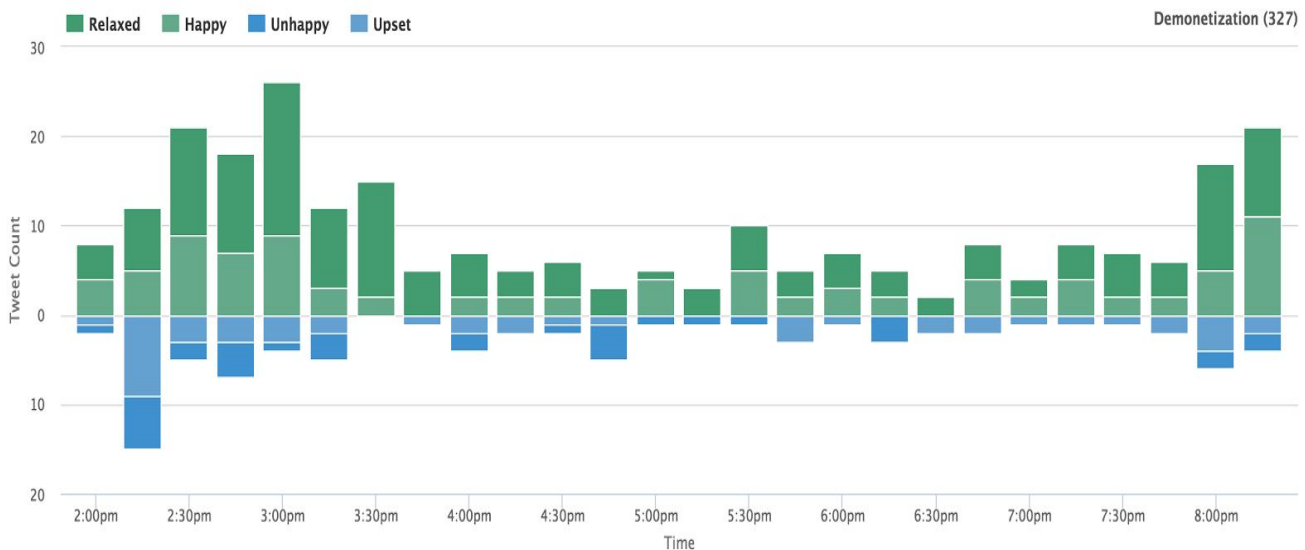
There is one other important field , which is the sentimental analysis of that tweet , the sentiment of a tweet represents the overall emotion of the tweet , i.e if it is positive/negative/neutral. This field is used to answer queries that involve emotions like happy/positive/negative/sad and so on. By maintaining the sentiment of every tweet , we get to know the overall sentiment of the corpus indexed. This helps us to answer queries like "Overall opinion of the public on any topic".

Thus , based on the intents returned from the "withandler" , we query for the appropriate intents from solr and retrieve tweets that are most relevant for that query and returns it back to withandler which in turn returns it to the UI.

## Natural Language Processing using NLTK

Natural Language Processing (NLP) is the process of trying to make sense of a body of text, in our case a tweet. There are several concepts that we looked into for implementation in our system such as, Sentiment analysis, Named Entity tagging (NE tagging) and Part of Speech tagging (POS tagging). A POS tagger tags every word in a sentence with their respective part of speech such as noun, verb, adjective, etc. Using this, we can split up tweets into Noun-Verb-Noun phrases that gives information about the subject Noun. An NE tagger tags every proper noun in a sentence with one of “Person”, “Organization” or “Location”. This can help us in answering who or where questions and the likes. Sentiment analysis is the process of analysing a body of text to determine its sentiment. We chose NLTK to implement the aforementioned concepts in our system.

Using NLTK in python, we can derive a sentiment score for every tweet. This score will either be a positive value, a negative value or neutral (0.0). By storing this score as a separate field in our SOLR index, we can answer sentiment related queries. For example, we can answer a question like “Show me a positive tweet from \*screen\_name\* on \*topic\*”. We retrieve all the tweets from “screen\_name” and filter only the positive tweets from these. Now we can query for the topic on these queries and return the top scoring tweets. Further improving on this, we can support queries on the general opinion of the public on a particular topic, for which we’ll query on the topic and count up the number of positive, negative and neutral tweets. Using these counts we can calculate and display the percentage breakdown of emotions of tweets.



Furthermore, we've also implemented Named Entity tagging in our system. A POS tagged sentence is passed into the NE tagger which tags the proper nouns in the sentence as either a Person, Location or Organization. By parsing these tagged strings, we've stored all the Person and Organization tokens in the Person field of our SOLR index and all the Location tokens into the location field of our index. Using this, we can try to answer Who and Where questions with further processing. For example, to answer a who question, we consider the subject of the question and search for this subject in all the tweets whose "Person" field isn't empty. Now, we count the occurrences of all tokens in the person field using a facet query and return the token which has the maximum occurrence as the answer. Similarly, we can also answer 'Where' questions using the same technique. This process of answering becomes more accurate, more the data we have indexed. At the time of writing, the question processing part of this query flow hasn't been trained yet in wit.ai.







# Wit.ai

## Overview


Wit.ai is an API that turns natural language (speech or messages) into actionable data. Wit.ai makes it easy for developers to build applications and devices that you can talk or text to. Wit.ai learns human language from every interaction, and leverages the community: what's learned is shared across developers. Wit has three main tabs; User stories, Understanding, Inbox.

## User Stories


User stories represents questions (In this case, about demonetization) which are to be answered. After we define a question, we need to specify entities within the question which are relevant to the question and use these terms to query in SOLR. Training a Wit intent with a dozen examples works well, and it's easy to leverage the community to get more examples. We train the entities which are absent in the example dataset ourselves by defining examples to the user stories. The bot is intelligent enough to learn by itself after couple of iterations by studying the proximity terms and entities.

 Inbox  Understanding  <sup>beta</sup> Stories  Actions


Read out tweets from Nare...

 "Read out tweets from Narendra Modi" tweets\_from


Show me a positive tweet ...

 "Show me a positive tweet on demonetization" demonetization sentiment question\_type:desc\_q

Show me the image of curr...

 "Show me the image of currency notes." image\_subject question\_type:desc\_q query\_subject

Show me the most favourit...

 "Show me the most favourite tweet about demonetisation." " superlative query\_subject demonetization question\_type:desc\_q



## Understanding

The understanding tab contains the list of entities present in the user stories which we defined. We can add/delete necessary key terms from the entities under this tab. We can also add synonyms to the existing key terms to map to the same term. Another important feature in understanding tab is for us the programmers to verify the entities defined in the stories and checking if it holds true for all cases. If it fails we fix and validate the story. This process is called training. Once we train it few times, the story will be generalized to a right format.

## Inbox

The inbox tab has all the queries entered by the user. This tab can be very useful to understand the user's expectation and requirement. We can leverage this by fixing our bot to conform to the user queries. We did this during the demo. We launched a video and asked our friends to try out the website, once they did we were able to get a clear idea of the questions that are being asked and we took this opportunity to further validate our stories.

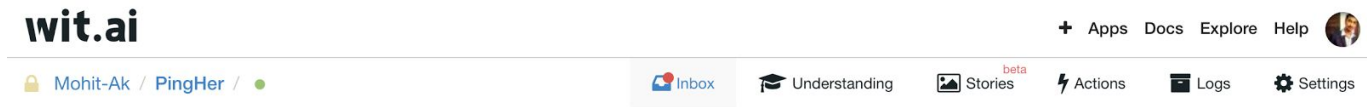
## Implementation

Wit uses OAuth form of authentication to communicate. Once the user has made their query, we make a HTTP request to wit.ai and it returns the entities corresponding to that particular query. We handle the entities returned from wit by passing it to their corresponding handler functions. What it means is, for example, if the entity include *tweets from* we make sure that we query the *screen\_name* field with the following entity value. Another example is if we have *image* as an entity we query *media\_url* field. We don't have to write handlers for all entities as we can generalize most of them by forming a context structure. For example if we have a query; *Show me the most retweeted tweet on Demonetization.* and *Show me the most retweeted tweet from DNA on Demonetization.* For both the queries we can make use of same functions with an addition of *screen\_name* field for the second query.

## Issues that exist

### Relevance Feedback

- There is no automated method for training the bot that is available right now. Unlike the search engines like Google & Bing where they use user clicks as measure of relevance, Wit.ai has a inbox where every query has to be manually validated for relevance.



### Live injection of data

- Currently, the data from twitter is preprocessed for Sentiments, Nouns, Verbs and indexed in solr every 24 hours. So the live analysis of people's sentiment can't be obtained.

### Chrome Synthetic Speech Generator

-The Chrome Synthetic Speech Generator is still in development stages and doesn't support "text to speech" for more than 200 characters.

## Future Improvements

- Answering questions like **Who** and **Where** specifically, by using NLTKs outputs, Named Entity Tagging. E.g. "Who is the Prime minister of India ?"
- Voice activation, still not supported in chrome. Can be done in Native apps or dedicated hardware devices.
- Decision Making using neural networks using knowledge graph. Would enable answering questions that involve prediction and judgements. E.g - Is it a good investment to buy gold today?
- Continuing conversation, Wit.ai supports context passing between dialogues in conversation. That enables us to have a human like conversation. E.g -  
User : When is the ...  
AI Bot : When is the, what?  
User : The last date ...  
AI Bot : The last date for what ?  
User : Exchanging the old notes...  
AI Bot : December 20, 2016