MediBo - a contextually aware medical chatbot

Andrew C Satz[1]*, Janak A Jain[1]*, Jonathan Galsurkar[1]*, Minghong Zheng[1]*, Shengyang Zhang[1]*

Abstract
MediBo is an artificially intelligent chatbot service with a vision to guide users respond better to medically sensitive situations. Using [4]'s Programmable SMS service and a data science toolbox involving natural language processing and machine learning, our team has successfully created a chatbot that is currently able to identify new users and on-board them on to the service. Currently, we are working on tweaking our NLP and ML models and will soon be to able to present the final product to the sponsors. This report highlights details of our work post the mid-term presentation along with pictures, illustrations and snippets of code. We look foward to the valuable feedback from the readers of this report. We also welcome them check our GitHub repository here.

Keywords
chatbot, natural language processing, human computer interaction, machine learning, artificial intelligence

[1] *Data Science Institute, Columbia University in the City of New York, New York, USA*
*__Emails__: {acs2228, jaj2186, jfg2150, mz2597, sz2624}@columbia.edu

Contents

Introduction

With continued construction of a medical chatbot (MediBo) that verifies whether adverse drug reactions are occuring, an onboarding system for users, a chat system, a natural-language-to-symptom matching system, and an symptom-to-advice system were necessary to provide a usable minimum viable product. The following report details the process of construction of these systems of MediBo as well as the challenges and failures. In addition, due to the fact that MediBo is providing a type of limited medical advice, it was necessary to examine the risks and potential failures of MediBo, in order to limit unintended negative consequence to patients which could lead to serious injury or death. The examination of these risks, potential failures, and processes to limit these risks are also explained.

1. Elements of a Conversation

In order to understand the working and structure of the chatbot, it is important to understand some basic terms that define the experience and several components contained within this experience. These have collectively been termed as "elements" of the chatbot. We have provided a details walkthrough through these elements

below:

1. **User**: A user is any human who consumes the experience of the product MediBo. Specifically, a user can be a patient, a medical practitioner, or a pharmacist.

2. **Chatbot**: The artificially intelligent computer program that communicates with the users through SMS channel.

3. **Thread**: A conversation thread (or simply a 'thread') is composed of one or more interactions between the user and the chatbot in a contiguous unit of time. A thread is composed of several elements such as messages, stories etc.

4. **Message**: A message is delivered as a "bubble" on the screen of the user and essentially contains some text. It also contains metadata such as intent, targets, return_intent, return_targets, tags etc.
   Story: A story is a message or collection of messages that delivers a specific piece of dialogue. Example: A story titled "welcome" serving three messages each targeting one of the following steps:
   - Greeting - greeting the user and welcoming
   - Introduction - introducing the chatbot service
   - Onboarding - signaling the beginning of onboarding.

5. **Intent**: Intent can be defined as the purpose of a message or a story. A special form of intent is return_intent which is a pre-emptive attempt to set the context labels to inform the chatbot in advance about the anticipated intent of the response.

6. **Target**: A target can be defined as the specific elements of a message or context that the message intends to serve. Example: A message asking for the user's age would have "age" in the targets list of the message. A special form of targets is return_targets which is a pre-emptive attempt to set the context labels to inform the chatbot in advance about the anticipated targets of the response.

7. **Tags**: Tags are essentially a union set of intents and targets and can be used for look-up of messages and stories.

2. Flow of a Conversation

A conversation can be seen as a sequence of interactions between a user and the chatbot. This is realized in the form of incoming messages from the user and subsequent responses from the chatbot. A conversation for each user session can have several possible combinations of messages.

With an open-text conversation, there are many possible use cases of chatbot experience. For example, one common use-case can be someone trying MediBo for the first time. In this case, this user's record would not be present with MediBo. Hence, MediBo must welcome, introduce and on-board the user before beginning a conversation with the user about how he or she is feeling. Another case is when someone wants to know what the weather is like. Such a query falls outside the scope of MediBo and technically, the output should mention a standard response mentioning the situation as such. However, such a strategy would impact the quality of user experience and hence, the chatbot should be trained for such situations as well.
Yet another case can be when someone is seeking urgent help, say when he/she is choking. Furthermore, this person may or may not be a MediBo. The response of MediBo in such cases falls in a grey area when MediBo has no information about the user to suggest an action and at the same time it can be argued that given the critical nature of the situation, MediBo should perform as per its purpose and take the necessary actions to help the person. This is a dichotomy - to be human or not to be human.

The examples given above are just three of the many such possible use cases. Given the time and resource constraints, we have focussed on very common use-cases and would incorporate specific use-cases in the future if the project is continued and/or accordingly funded.

For the purpose of this project, our simple approach is the following:

IF User is New:

Figure 1. Onboarding - Welcome

Figure 2. Onboarding - Introduction

```
    Onboard
    Add User record
ELSE:
    Begin Conversation
```

In the following sections, we will describe these components in detail.

3. Onboarding

When a user texts MediBo and the user's phone number is not in the system, MediBo will initiate the on-boarding processing. This will allow MediBo to recognize the user in the future, save user information such as demographics and medications, and save chat logs. Onboarding consists of some basic questions. The user is asked for their name so that MediBo can address the user with their name, giving a more personalized sensation. Their age and sex and allergies are also be asked. Currently, MediBo does not take this information into account when giving a user an action to take based on their symptoms. Future iterations will use this information to make better decisions since some symptoms may or may not be serious depending on gender, age, and allergies.

The onboarding begins with a welcome message welcoming the user to the app. This is followed by a message to introduce the service. Finally, the chatbot specifies the need to collect information from the user and this is where the onboarding process begins.

4. Experiment: Seq2Seq

In this section, we report some preliminary results about the experiments of the NLP part we tried which, unfortunately, will not likely be used in the final product. One of the most widely used network structure, sequence-to-sequence (Seq2Seq) achieved great success in neural machine translation and chatting tasks [3].

The Seq2Seq model has two parts: an RNN-encoder module and a RNN-decoder module, where the encoder module maps an input sequence to a vector representation, and the decoder maps the vector representation back into an out sequence. The two networks are trained jointly to maximize the conditional probability of the output sequence given the input sequence. One well known trick is adding an attention mechanism [1] to the decoder which can significantly improve the performance [5].

After the review of these papers, we decide to follow the instruction of "Deep Reinforcement Learning for Dialogue Generation" [2] to construct a Seq2Seq based model for our chatbot project. Unfortunately, we significantly overestimated our computational power while we trained the deep LSTM Seq2Seq model at the first time. Later, with the knowledge of our constraint on

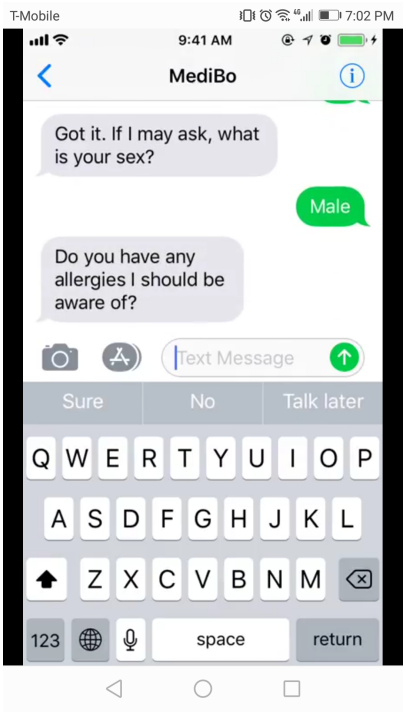


Figure 3. Onboarding - Leadgen

the computational resources, we use a much simpler and smaller Seq2Seq model and use a 10 times smaller sub-dataset for training. The entire training process was done on a laptop-level CPU, therefore the training was extremely slow (80 minutes for each epoch) without GPU support. In the scheduled time, we were only able to train 30 epochs for the attention Seq2Seq model and 600 episodes for the model, hence the results here are very preliminary and may differ quite a bit from the original paper.

4.1 Dataset

Due to the difficulty of finding any medical chatting related dataset, like the original paper, we use the OpenSubtitle dataset which comes up with reliable chatting outcomes. The OpenSubtitle dataset contains millions of English subtitles in different type of movies. The authors used 10 million sentences and extracted 0.8 million from them to ensure the network will not easily generate dull responses. We use a much smaller subset which contains 9487 pairs of sentences for faster training.

4.2 Network configuration

We use an one-layer encoder LSTM and an one-layer attention-based decoder LSTM as the Seq2Seq model. The dimension of word embedding is set to be 150, and the number of hidden units in the LSTMs is set to be 250.

Table 1. An example dialogue from OpenSubtitle dataset.

A: YOU SMOKE?
B: YES , I DO
A: IT'S BAD FOR YOU!
B: BUT YOU SMOKE!
A: PUT IT OUT.
B: YOU'RE STILL A MINOR

Figure 4. Seq2Seq - Training data set - Table 1

Table 2. An example dialogue generated by the Seq2Seq model.

A: YOU SMOKE?
B: YES , I DO
A: SYLVIA?
B: IT'S GOOD.
A: I LOST ONLY STARS.
B: BLASPHEMER UP.

Figure 5. Seq2Seq - Output - Table 2

We use the standard from of soft attention introduced by Bahdanau et al.

4.3 Results and discussion

We first give an example of the sentences in the OpenSubtitle dataset. Table 1 shows an example in the training set.

Table 2 gives an example dialogue generated by our Seq2Seq system with the initial sentence, which is the same as in Table 1, and we ask the agent to "talk" to itself. We can see that the first response generated is the same as what we expected, but as the conversation between the agent starts, the later generations are much different from the ideal case and is somehow meaningless.

We find it somehow hard to understand without a longer context, while the usage of the words are also ambiguous. As result, due to the poor generated dialogue of the Seq2Seq model we have and the limitation of our schedule and computational resource, we ended up deciding to abandon the Seq2Seq model and use other approaches to pursue our product.

Unfortunately collecting the data for a medical chatbot is not currently possible. As a result, MediBo will save the logs for each user with the hope that, after enough logs are generated through user interactions, MediBo can use these logs to train a highly functional Seq2Seq model.

5. PyMetaMap

```
>>> sentence = ['I have head pain']
>>> concepts,error = mm.extract_concepts(sentence,[1])
>>> print(concepts)
[ConceptMMI(index='1', mm='MMI', score='17.80',
preferred_name='Headache', cui='C0018681', semtypes='[sosy]',
trigger='["HEAD PAIN"-tx-1-"Head pain"-noun-0]', location='TX',
pos_info='1/9', tree_codes='C10.597.617.470;C23.888.592.612.441')]
```

Figure 6. PyMetaMap - Association of concepts

5.1 Introduction

MetaMap is a tool for recognizing UMLS Concepts in Text. For example if a user has a headache and a fever, the meta mapping will be as follows:
Meta Mapping (1000):
1000 HEADACHE (Headache) [Sign or Symptom]
Meta Mapping (1000):
1000 FEVER (Fever) [Sign or Symptom]
The main functional responsibility of MetaMap is to find the symptoms given a query. MetaMap however, is a Java based API. Since we are running our code in Python, we make use of PyMetaMap, which is a Python wrapper around MetaMap. Pymetamap takes a list of sentences as inputs and extracts medical concepts via MetaMap, then returns the UMLS medical concepts as outputs with similarity scores.

5.2 Working

As you can see from the example in Figure 6, given a sentence, "I have head pain," PyMetaMap extracts the term "head pain" and returns a preferred name " headache" with a similarity score 17.80.

There can be multiple UMLS concepts mapped to a single input. As part of the output of PyMetaMap, the similarity score for each UMLS concept is available. The higher the score, the more confident PyMetaMap in matching the user input to the outputted concept.
As you can see from the example below, given the input the sentence(i.e.lightheadedness, dizziness, or fainting), Dizziness Adverse Event has the least similar score compared to Dizziness, Syncope, Lightheadedness, and Vertigo, which makes sense.

We now have a way to extract potential symptoms given an input. There can also be many outputs given a single symptom, i.e. different words could mean the same symptom. For every symptom for each medication through pymetamap, the "preferred names" were obtained and, as a result, the various alternate names for the symptom were found in the metamap rendering of UMLS concepts. This allows for a direct match between a user input and a symptom in the dataset. For example,

```
>>> sents = ['lightheadedness, dizziness, or fainting']
>>> concepts,error = mm.extract_concepts(sents,[1])
>>> for concept in concepts:
...     print(concept)
...
ConceptMMI(index='1', mm='MMI', score='35.81',
preferred_name='Dizziness', cui='C0012833', semtypes='[sosy]',
trigger='["Dizzyness"-tx-1-"dizziness"-noun-0,"Lightheadedness"-tx-1-
"lightheadedness"-noun-0]', location='TX', pos_info='18/9;1/15',
tree_codes='C10.597.751.237;C23.888.592.763.237')
ConceptMMI(index='1', mm='MMI', score='24.11',
preferred_name='Syncope', cui='C0039070', semtypes='[sosy]',
trigger='["FAINTING"-tx-1-"fainting"-noun-0]', location='TX',
pos_info='32/8',
tree_codes='C10.597.606.358.800.600;C23.888.592.604.359.800.600')
ConceptMMI(index='1', mm='MMI', score='17.80',
preferred_name='Lightheadedness', cui='C0220870', semtypes='[sosy]',
trigger='["LIGHTHEADEDNESS"-tx-1-"lightheadedness"-noun-0]',
location='TX', pos_info='1/15',
tree_codes='C10.597.751.237;C23.888.592.763.237')
ConceptMMI(index='1', mm='MMI', score='17.80',
preferred_name='Vertigo', cui='C0042571', semtypes='[sosy]',
trigger='["dizziness"-tx-1-"dizziness"-noun-0]', location='TX',
pos_info='18/9',
tree_codes='C09.218.568.900.883;C10.597.951;C23.888.592.958')
ConceptMMI(index='1', mm='MMI', score='5.18',
preferred_name='Dizziness Adverse Event', cui='C1963093',
semtypes='[fndg]', trigger='["Dizziness"-tx-1-"dizziness"-noun-0]',
location='TX', pos_info='18/9', tree_codes='')
```

Figure 7. PyMetaMap - Model scores and symptom matching

if a user inputs that they have head pain and high temperature, this will map to the umls concepts of headache and fever.
Given that the user is on a certain medication, MediBo can directly check if those UMLS concepts are found in the meta mapped symptoms for that medication. If the symptoms are present, MediBo then looks for the severity of that symptom, which is a field in our data set.

If at least one of the symptoms is deadly or serious, MediBo advises to go to the emergency room or call a doctor, respectively and includes the symptom that motivated the decision. If all of the symptoms experienced are, based upon the dataset, normal, MediBo will advise that no action is necessary. If no UMLS concepts are found from the symptoms inputted by the user, MediBo will confirm that the user is not experiencing any of the serious side effects by asking the user directly.

There is one **limitation** for the MetaMap program: it may not give accurate similar UMLS concepts for some terms. For example, when "head hurts" is used instead of "head pain," PyMetaMap returned "head" rather than "headache" as an output. In this case, MediBo may not find the right symptoms, and the responses will not be what a user expects. One solution for this is to confirm with the user to make sure the UMLS concepts match the symptoms that the user is referring to. In addition, over time will save the terminologies that PyMetaMap does not recognize (e.g. "hurts") and map those terms

to synonyms that PyMetaMap can recognize (hurts -¿ pain, which would map to the correct UMLS concepts: “headache”). We expect MediBo’s accuracy to become higher through continued use and learning of pitfalls.

6. MediBo Risk Reduction

While the intent is to make MediBo as accurate as possible and lead to better health outcomes, there are serious risks involved in giving advice to individuals. As a result, an analysis of potential failures was performed. The first failure involved given a patient advice, based on an ADR, that would be incorrect. Users of MediBo are offered three pieces of advice: Go to the emergency room/call 911(“Emergency”); Contact a doctor (“Call Doctor); or No action necessary (“No Need”). A major risk comes into play if MediBo were to advise a patient “No Need” when in fact the patient it should advise “Emergency.” In order to understand the likelihood of this occurring, an analysis of all 32,269 ADRs was performed to determine 1) how many ADRs can have multiple pieces of advice given and 2) what is the likelihood that an error would occur that could cause MediBo to become confused.

To understand whether a particular ADR can have multiple pieces of advice, all ADRs with their associated advice were extracted from the dataset, regardless of the medication. If a particular ADR, nausea for example, offered the advice “Emergency” for one drug and “No Need” for a different drug, then nausea would fall into the yellow region (see Figure 8). In order for the error to occur though, MediBo would need to confuse two medications in the yellow region. For that to occur, there would need to be a user error when entering the prescription name.

The team assumed that user error would be due to misspelling of a prescription name, which is highly likely, considering the often complicated names of prescription drugs. To predict the likelihood of a user error, we used a form of fuzzy matching between all prescription names to determine the probability of similarity between drug names. We used the fuzzywuzzy python library, which employs Levenshtein distance to calculate the differences between sequences and strings. The higher the outcome, the more probable that the two words match. After reviewing the pairs of drug names, it was determined that a cutoff of greater than 70% was enough to have observe similar pharmaceutical spelling where confusion by MediBo could be a risk. In the case of the 244 ADRs that could advise “Emergency”

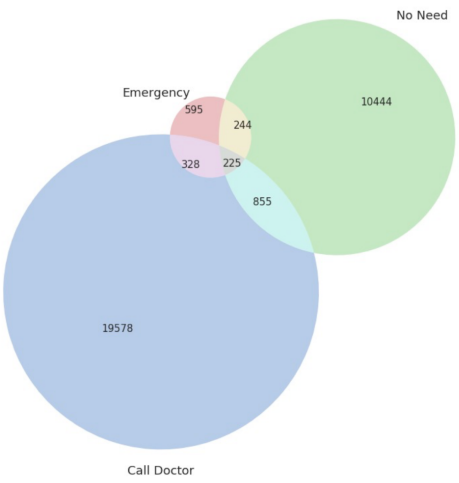


Figure 8. Venn Diagram - Adverse Drug Reactions

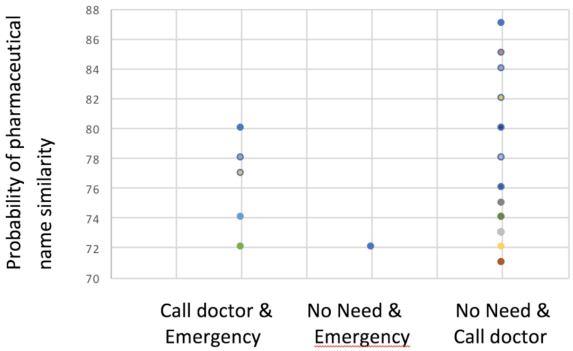


Figure 9. ADR Overlap

and “No Need,” there was only one instance where two drugs could be confused by spelling, and the likelihood of confusion by spelling is 72% (see Figure 9). There were 6 instances of potential spelling confusion between two drugs with the ADR advice being “Emergency” and “Call Doctor” and 24 instances of potential spelling confusion between two drugs with the ADR advice being “No Need” and “Call Doctor.” In order to prevent this failure, it would be necessary to build in a verification system into MediBo which addresses word similarity. A feature that could solve this, without causing excess user input, would be to verify which of the two drugs the user was inputting at the time of prescription onboarding.

There is an additional risk of failure of MediBo when a patient is taking multiple medications. For example, if a patient is taking both medication A and medication B, and both of the medications have an ADR of “nausea” but the advice for medication A is “Call Doctor” and medication B is “Emergency,” MediBo may become confused. It would be simple to solve this problem by

always defaulting to the more serious of the advice. The challenge comes into play when a patient has only provided one of the two medications at the time of prescription onboarding. During data exploration, we discovered that our dataset only included the generic drug names rather than the commercial drug names. This is fine for a proof-of-concept, however, it may create an additional user-input challenge if deployed. This could be solved by utilizing the GoodRx API.

With all of the prescription drug name data, we could determine if the multiple medication challenge is a real problem, by examining the top ten two and three drug combinations could be examined. Although we do not have all of this data due to the missing commercial names, it was determined that some of these two drug combinations fall into the overlapping regions of the Figure 8. This could present a challenge to the patient as MediBo would advise based solely on the input by the patient of their prescription information. A feature that could potentially solve this, without causing excess user input, would be to verify, at the time of advice, whether the patient is taking specific commonly co-prescribed drugs and utilize that information to default to the more serious advice. Additionally, this information could provide valuable data on drug interactions, which would could provide additional commercial viability and provide additional guidance to patients, doctors, pharmacists as well as pharmaceutical companies the U.S. Food and Drug Administration.

Acknowledgments

We would like to thank Prof. Smaranda Muresan for her guidance and support. Her prowess in linguistics, especially Natural Language Processing is very relevant to this topic and we seek her support in throughout this study. We also sincerely wish to thank Mr. Victor Gonzalez for sponsoring this interesting project and for investing his valuable time in helping us take the right direction. Last, but not the least, we would like to thank Aayush Mudgal for his consistent efforts in organizing the meetings and calls and accommodating our concerns despite his understandably busy schedule.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. eprint: arXiv:1409.0473.

[2] Jiwei Li et al. *Deep Reinforcement Learning for Dialogue Generation*. 2016. eprint: arXiv:1606.01541.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. eprint: arXiv:1409.3215.

[4] *Twilio - Communication APIs for SMS, Voice, Video and Authentication*. https://www.twilio.com/. (Accessed on 10/30/2017).

[5] Tsung-Hsien Wen et al. *A Network-based End-to-End Trainable Task-oriented Dialogue System*. 2016. eprint: arXiv:1604.04562.

7. About the Team

7.1 Andrew C Satz

Andrew Satz is a Master's candidate in data science at Columbia University's School of Engineering. He comes with nearly 20 years of experience in the health insurance industry, with roles in risk management and safety. More recently, Andrew founded a data science consultancy firm focusing on the healthcare industry. Andrew is passionate about music, travel, and discovering innovative and realistic ways to utilize technology to solve the challenges of today and tomorrow.

7.2 Janak A Jain

Janak is currently an MS candidate at Columbia University's Data Science Institute. With a rich experience in market research and consulting across several categories, he wishes to explore future steps of data across industries such as marketing, smart cities and healthcare.

An MBA Tech alumnus of SVKM's NMIMS, Mumbai - his work has moved along an interesting intersection of technological and managerial spaces.

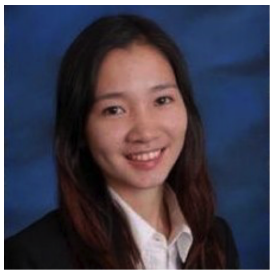
He knows Hindi, English, Marathi and French with qualifications of various levels.

7.3 Jonathan Galsurkar

Jonathan Galsurkar is a final year Master's student at Columbia University studying Data Science. His team came in first place at the 2017 Columbia Data Science Hackathon.

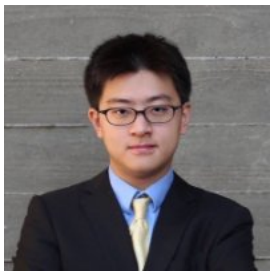
He received his Bachelor's degree in Computer Science and Mathematics from CUNY Hunter College. He is currently a Data Science Research intern at IBM, focusing on sentence/paragraph embedding and semantic searching techniques & applications. His main goal is to use data for social good. For fun he likes to play guitar, snowboard, and travel.

7.4 Minghong Zheng



I am very interested in applying advanced models with my business sense to do user behavior analysis. In addition, I like travelling and outdoor activities to try different things and visit as many interesting places as I can.

7.5 Shengyang Zhang



I am a graduate student at Data Science Institute, Columbia University, currently in my last semester and plan to graduate in Dec. 2017.

Data Structure - Messages

```
[
    {
        "1":{
            "_id": 1,
            "title": "introduction",
            "targets": ["welcome"],
            "intent": ["hello"],
            "text": "👋😄 Hello! I am MediBo. Your personal medical chatbot.",
            "entities": ["medibo","chatbot"],
            "tags": ["hello","yourself"],
            "weights":
                {
                "information": 0,
                "remedy": 0,
                "general": 1,
                "query": 0,
                "leadgen":0
                },
            "embed": []
        },

        "2":{
            "_id": 2,
            "title": "description",
            "targets": ["welcome"],
            "intent": ["description"],
            "text": "I am here to keep you healthy by keeping a track of your 💊",
            "entities": ["medibo","chatbot"],
            "tags": ["medicines"],
            "weights":
                {
                    "information": 0,
                    "remedy": 0,
                    "general": 1,
                    "query": 0,
                    "leadgen": 0
                },
            "embed": []
        },

        "3":{
            "_id": 3,
            "title": "onboarding",
            "targets": ["onboarding","register"],
            "intent": ["onboarding"],
            "text": "Let's get started, shall we? 🙂",
            "entities": ["start","begin","register"],
            "tags": ["register","medibo"],
            "weights":
                {
```

Figure 10. Messages: JSON file snapshot

Data Structure - Stories


```
[
    {
        "first_welcome": {
            "_id":1,
            "title": "welcome",
            "targets": ["welcome","onboarding","register"],
            "intent": ["hello","description","onboarding","leadgen"],
            "messages": [1,2,3,4],
            "return_intent":["name"],
            "return_targets": ["name","demog"]
        },

        "onboarding_age": {
            "_id": 3,
            "title": "onboarding_age",
            "targets": ["demog","name"],
            "intent": ["leadgen"],
            "messages": [5],
            "return_intent":["leadgen"],
            "return_targets": ["age","demog"]
        },

        "onboarding_sex": {
            "_id": 4,
            "title": "onboarding_sex",
            "targets": ["demog","age"],
            "intent": ["leadgen"],
            "messages": [6],
            "return_intent":["leadgen"],
            "return_targets": ["sex","demog"]
        },

        "onboarding_allergies": {
            "_id": 5,
            "title": "onboarding_allergies",
            "targets": ["demog","sex"],
            "intent": ["leadgen"],
            "messages": [7],
            "return_intent":["leadgen"],
            "return_targets": ["allergies","demog"]
        },

        "onboarding_complete": {
            "_id": 5,
            "title": "onboarding_complete",
            "targets": ["demog","allergies"],
            "intent": ["leadgen"],
            "messages": [11],
            "return_intent":["leadgen"],
```


Figure 11. Users: JSON file snapshot

Data Structure - Users


```
[
    {
        "+17868777177": {
            "name": "Andrew",
            "age": 42,
            "sex": "male",
            "phone": "+17868777177",
            "allergies": []
        },

        "+19518010312": {
            "name": "Minghong",
            "age": 23,
            "sex": "female",
            "phone": "+19518010312",
            "allergies": []
        },

        "+18056377924": {
            "name": "Shengyang",
            "age": 25,
            "sex": "male",
            "phone": "+18056377924",
            "allergies": []
        }
    }

]
```


Figure 12. Users: JSON file snapshot