AnaBot: Lessons from Building a Serial Chatbot in Collaboration with Analysts and Linguists

Hongche Liu
LinkedIn
holiu@linkedin.com

Jaewon Yang
LinkedIn
jeyang@linkedin.com

Qi He
LinkedIn
qihe@linkedin.com

ABSTRACT

AnaBot (**Ana**lytics chat**Bot**) is the umbrella project for all the chatbots we built for our users to access insights from our knowledge base in the cloud. We'd like to share the lessons learned through several launches of the chatbots end-to-end in collaboration with data analysts and linguists to hone the process of data collection, modeling, dialog management and designs. For any researchers and developers in chatbots and Q&A systems, we believe there are universal values in sharing these.

KEYWORDS

Chatbot, machine learning, query language generation

1 Introduction

Data analysts overwhelmed with requests for insights came up with a request - can you have a self-serve interface for non-tech people to query a knowledge base in the cloud without training, without SQL knowledge or Hadoop experience? This started our journey of building a serial chatbot, freeing up analysts' time to build more complex models for new insights, which are then on-boarded to the chatbot for even better coverage, a positive cycle of increasing productivity for the chatbots, analysts and users.
We call it a serial chatbot because it is one chatbot design running on a single platform, by the same process, on multiple knowledge domains (one for LinkedIn product metrics; the other one in CRM). There have been published work on real-world chatbot for answering FAQ questions [4, 5]. They both use AIML pattern matching method for input and templated output. There are a host of impressive Q&A system on knowledge graph, but we do not know of a released application on it. We are unique in that it is the first end-to-end machine learning model based chatbot on knowledge base through SQL in an enterprise application.

2 Problem Statement and Our Solution

We focus on building a Q&A bot that can handle two types of questions. The first type is *metric* questions, such as "how many daily active users do we have?". The second type is *definition* questions, such as "what is DAU?". We chose these two types based on a user survey. With these two types defined, the problem we solve is the following: Given a natural language question from a user, we provide a appropriate answer based on user's intent of *metric*, *definition*, or otherwise *out of scope*.

To tackle the problem, we decompose the problem into multiple subproblems, each of which involves a simple classification task. This divide-and-conquer approach has proven to be effective in other Q&A benchmark tasks [1, 2]. Here are our subproblems:

1. Question2Intent: It is to determine if the input question belongs to *metric* intent, *definition* intent, or *out-of-scope* (e.g. "what is your favorite metric, Ana?").
2. Question2Definition: If the question has a *definition* intent, it predicts the most likely definition from a dictionary.
3. Question2SQL: If the question carries a *metric* intent, we generate an SQL query to retrieve the answer from our knowledge base. To do this, we formulated a slot filling framework which is known to be a very efficient method for SQL generation [2]. In particular, we define "Metric Query Language (MQL)", a simplified query language format for metric queries. MQL consists of 5 slots: Metric (Metric to be queried), Filter (Filtering clause), Breakdown (Column to group by), Time (Time duration) and Country (Country filtering clause). The latter 4 are collectively called dimensions. Combination of 5 slots uniquely determines a SQL query. By using MQL, we were able to reduce our search space by 100X.

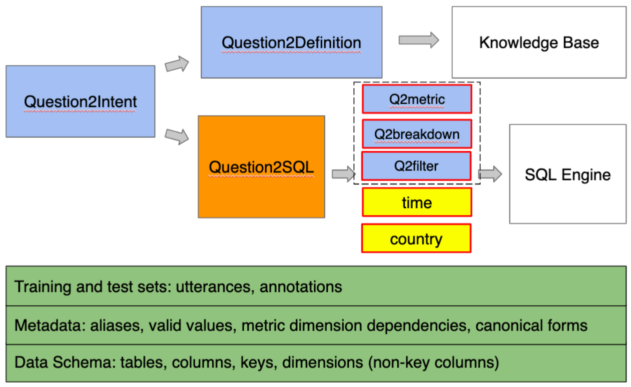

Question2Intent
Question2Definition
Knowledge Base
Question2SQL
Q2metric
Q2breakdown
Q2filter
time
country
SQL Engine
Training and test sets: utterances, annotations
Metadata: aliases, valid values, metric dimension dependencies, canonical forms
Data Schema: tables, columns, keys, dimensions (non-key columns)

Figure 1. Model Architecture

The above diagram shows the hierarchical model architecture and data foundations common to all our chatbots. The components in blue are all using logistic / softmax regression models. The components in yellow are handled by the Spacy Named entity tagger [3]. The component in orange Question2SQL is just an object encapsulating the 5 models.
We summarize our contributions as follows:
Develop an end-to-end Q&A bot by combining multiple classifiers: Q&A models developed in the research community

tend to be evaluated with offline benchmark datasets that focus on a specific classification task. In reality, building an end-to-end Q&A system involves a sequence of prediction tasks. In our work, we show how to combine multiple AI models in a hierarchical way to develop a Q&A bot that can handle users' questions reliably.

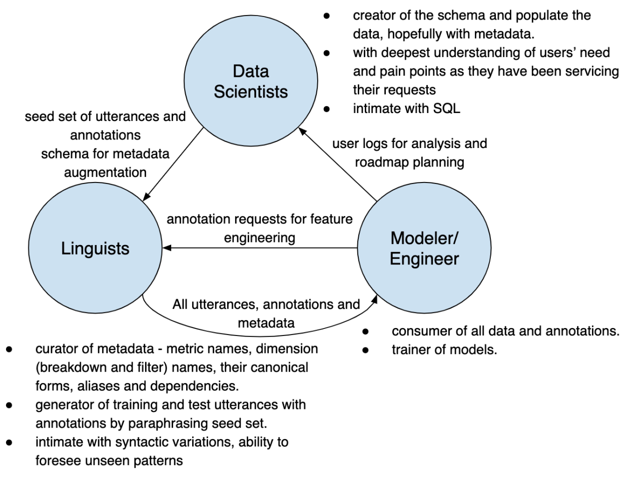
Leverage metadata for handling cold-start cases: Existing Q&A models assume that there exists training data large enough to train complex models, and a well-managed knowledge graph (or database) that the models can leverage. In enterprise setting, these assumptions do not hold; there is no training data (questions and answers) and knowledge graph needs to be built by the bot developers. In this study, we show how we can overcome lack of training data by constructing high quality metadata.

Design a slot filling method for SQL: As noted, we developed the efficient slot filling framework named "MQL" for the SQL generation.

Telling the SQL query back to the user in natural language: To help the user understand what is going on inside the Q&A bot, we show the converted SQL query to the user. Instead of showing the SQL directly, we show the meaning of SQL in natural language. In particular, for each MQL slot value, we present its English name (canonical name). For example, if MQL slots were (metric = DAU, country = US), we show the user that "We will show the answer for `The number of daily active users is USA`". This way, the bot can communicate with the users who do not understand SQL at all.

Repeatable and transferable process: We repeated this process multiple times whenever we expand our Q&A bot to a new domain. For example, we applied the process to cover the metrics in user engagement in the Flagship app, and then repeated to cover metrics in the sales domain. In this study, we showed that our process can be transferred across different domains. This result will help the KDD audience who wants to build a Q&A bot for her enterprise.

Collaboration with linguists, engineers and data scientists: Building a bot without training data requires significant contributions from linguists and domain expert (e.g., data scientists). We developed a collaboration process between the three groups of contributors as follows:


Data Scientists
Linguists
Modeler/ Engineer
creator of the schema and populate the data, hopefully with metadata.
with deepest understanding of users' need and pain points as they have been servicing their requests
intimate with SQL
seed set of utterances and annotations
schema for metadata augmentation
user logs for analysis and roadmap planning
annotation requests for feature engineering
All utterances, annotations and metadata
consumer of all data and annotations.
trainer of models.
curator of metadata - metric names, dimension (breakdown and filter) names, their canonical forms, aliases and dependencies.
generator of training and test utterances with annotations by paraphrasing seed set.
intimate with syntactic variations, ability to foresee unseen patterns

3 Lessons

Metadata: Most insights start as ad hoc requests but end up being used by many other users. There should be a policy to review the metadata along with the code and insights. The metadata including semantics, keywords and acronyms of the metrics and dimensions. Such a policy can avoid the pains of recreating them after the fact.

Interdependencies between metrics and dimensions: Insights developed over time in the cloud usually cover multiple datasets, each with several tables. Tables don't join across different datasets. In the hierarchical models depicted in Fig. 1, the dashed box contains 3 independent models Q2metric, Q2breakdown and Q2filter. It is actually the outcome of a practical design choice between 2:

1. A monolithic one that models all valid combinations of metrics, breakdowns and filters. This is precise but hard to scale in terms of training data and model complexity.
2. Three independent slot models plus a validation module that predict the most likely and valid combinations from the probabilities of all models.

Managing Dialog and Context:

- if you are building a single chatbot, you may use frameworks such as Microsoft BotFramework or api.ai's Dialogflow, where messages, prompts, flow, state are coded in;
- if you plan to build a serial bot covering different domains of knowledge, we suggest encapsulating all the above 5 types of variables in a graph-like structured file (e.g. Json). This separates the design of conversation flow from programming functionalities, which require very different skills.

SQL polymorphism ushers in MQL: The design of MQL, introduced earlier, is necessitated by many conceptual metrics, e.g. "active members", which would require a join in our normalized schema, e.g. a member table and an activity table. When we started with the SQL approach, we found that annotated SQL statements have multiple correct forms with different join order and keys. Among the 2 solutions:

1. De-normalize the tables with redundant fields to save joins
2. Develop a higher-level language that encapsulates the joins and exposes the new metric as if they are already joined. The translation library converts MQL to SQL queries.

We find the 2nd one to be a worthwhile investment for a serial bot.

Design conversation around model imperfections. Our model predicts ranked classifications. A practical conversation design should always leave room for the user to request partial corrections, upon which the bot presents the next set of candidate results for the user to choose from.

REFERENCES

[1] Mohammed et al., Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks, In NAACL-HLT 2018
[2] Zhong et al., Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning, https://arxiv.org/abs/1709.00103
[3] Spacy, http://spacy.io
[4] Supratip Ghose, Jagat Joyti Barua, Toward the implementation of a topic specific dialogue based natural language chatbot as an undergraduate advisor, 2013 International Conference on Informatics, Electronics and Vision
[5] Anamika Gupta, Gunjan Gupta, Arjun Malhotra, Khushboo Chitre, Ojasvi Aggarwal, Nitesh Kumar Gupta, Intello: An Intelligent Chatbot for Replacing FAQs, JOURNAL OF WEB ENGINEERING & TECHNOLOGY