NL2SQL: NLI Matching Model

Qingning Yao (Chris), Tao Yu, Dragomir Radev

Introduction

Summarizing SQL patterns from training data, identify which pattern matches the input question and construct SQL with question and pattern.

What is a Pattern?

A SQL stripped down to skeleton. For example:

Question: "**Find the name of customers who are living in Colorado.**"

```
SELECT t1.customer_name FROM
customers AS t1 JOIN
customer_addresses AS t2 ON
t1.customer_id  =  t2.customer_id
JOIN addresses AS t3 ON
t2.address_id  =  t3.address_id
WHERE t3.state_province_county  =
"Colorado"
```

SQL

```
SELECT {COLUMN}
WHERE {COLUMN} = {VALUE}
```

Pattern

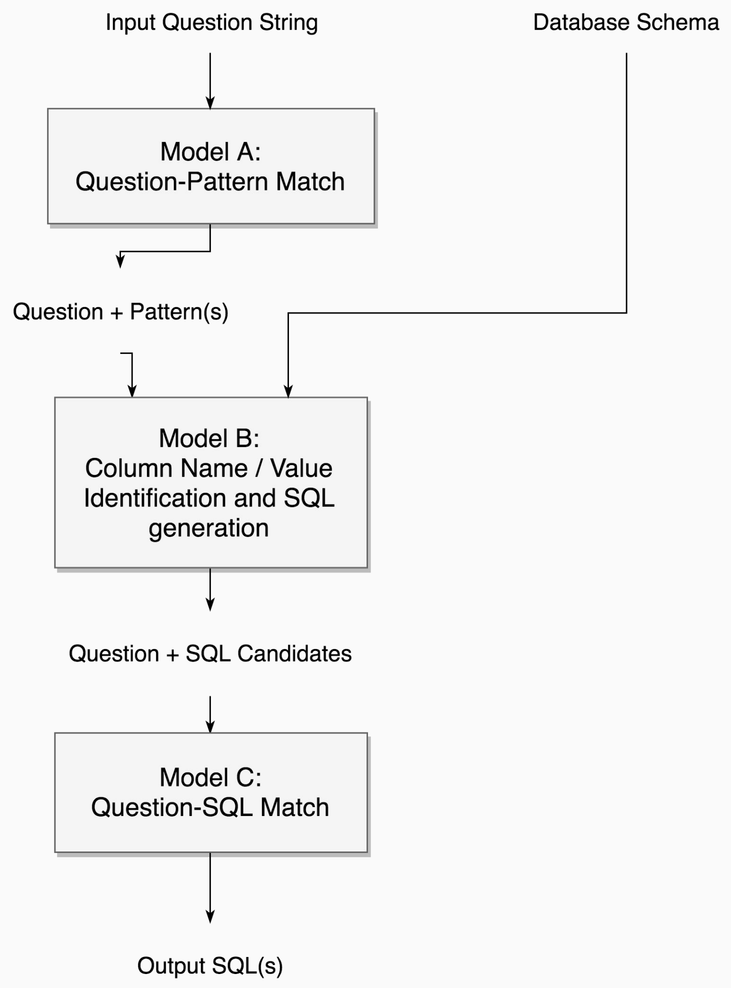
```
select where equals
```

Pattern (Simplified and Naturalized)

How many patterns are there?

Out of the 5000+ preprocessed SQLs that we are having right now, there are around **70** to **400** patterns depending on the level of simplification, and these should cover the vast majority of all SQLs in real life.

Model Structure



Model A: Question-Pattern Match

Given a question, predict which pattern would be the best fit for this question.

- Using existing Natural Language Inference (NLI) models
- Like paraphrase identification
- Instead of "contradiction" / "entailment" etc., use "match" and "not match"

Training Model A

We used an NLI model by Zhiguo Wang, Wael Hamza and Radu Florian: ***Bilateral Multi-Perspective Matching for Natural Language Sentences*** (BiMPM).

- Generate training data by
  a. random mismatch of training question-pattern pairs
  b. slightly tweaking a correct pair (to capture details)
- Changed the evaluation metric
  a. from "correctly identifying match/mismatch"
  b. to "correctly identify the pattern out of all patterns"

Model B: SQL Generation

Given:

1. Question
2. Predicted pattern of the corresponding query of that question
3. Database schema

Generate the **most likely SQLs that would retrieve the answer to this question**.

We can potentially reuse any other models in this project for Model B.

Model C: Question-SQL Matching

Given a question and a list of SQL candidates, identify which one among them would be the one that could correctly answer the question.

- Potentially another NLI model
- Could be trained just like model A

Pros and Cons

Pros:

- Intuitive, analogous to the way humans learn and write SQL
- Simple and straightforward

Cons:

- Cannot guarantee 100% SQL pattern coverage

Potential Byproduct: Data Augmentation

1. Strip SQLs to skeleton, leaving behind replacements tokens
   - Like `{COLUMN NAME}` and `{VALUE}`
2. Use the non-keyword stripped from SQL to strip questions
   - Non-keywords: either column names or values
   - With other statistical approaches like word frequency
3. (Possibly) Hand pick some reusable high-quality Question-Pattern-SQL-Pattern pairs
4. Deduct column types statistically
   - E.g. can column `order_type` be `GROUP BY`'ed over?
5. Generate new Question-SQL pairs