Алгоритм преобразования нечетких запросов на естественном языке в формат JSON с помощью технологии векторизации

Чан Дык Мань

Факультет компьютерных технологий и информатики
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

duc.manh.tran@mail.ru

Ха Муон

Факультет информационных технологий
Телекоммуникационный университет

Нячанг, Вьетнам
muon.ha@mail.ru

Аннотация: **Нечеткие запросы широко используются в системах поиска информации, где намерение пользователя выражается в неточных терминах, таких как «дешево», «дорого» или «быстро». В данной статье предлагается алгоритм, который сопоставляет каждый нечеткий запрос на естественном языке с ближайшим шаблоном JSON в наборе заданных шаблонов с использованием технологии векторизации. Этот алгоритм не требует каких-либо размеченных обучающих данных. В статье приводится рассчитанная точность преобразования запроса на естественном языке в формат JSON на основе предлагаемого алгоритма с использованием трех технологий векторизации — TF-IDF, Word2Vec и SimHash — на наборе данных из 2000 пар нечетких запросов. TF-IDF достиг наивысшей точности преобразования — 99,2%, за ним следует SimHash с 99,15%, а Word2Vec значительно отстает — 62,6%. Результаты показывают, что технология векторизации TF-IDF подходит для коротких нечетких запросов (короткий текст). Предложенный алгоритм может служить этапом предварительной обработки для инструментов обработки нечетких запросов.**

Ключевые слова: нечеткие запросы, обработка естественного языка, JSON, TF-IDF, Word2Vec, SimHash

I. Введение

Преобразование запросов на естественном языке в машиночитаемые структурированные запросы является давней проблемой в исследованиях взаимодействия человека с компьютером. Сегодня все большую востребованность набирают средства преобразования текстовых запросов пользователя на естественном языке в исполняемые SQL-запросы [1]. Методы преобразования текста в SQL-запрос можно разделить на следующие три категории:

1. Методы, основанные на синтаксическом анализе, анализирующие входной запрос для определения его грамматической структуры, и затем преобразующие эту структуру в SQL-запрос на основе набора правил. Преимущества этих методов заключаются в том, что их легко объяснить и модифицировать благодаря набору четких правил. Они не требуют данных для обучения и обладают высокой скоростью обработки запросов. Однако недостатки этих методов заключаются в том, что разработка набора правил требует больших усилий, и они не могут обрабатывать запросы с незнакомыми структурами, не включенными в набор правил [2].

2. Методы с использованием нейронных сетей: в последние годы использование нейронных сетей, особенно структуры seq-to-seq, для решения задачи преобразования текста в SQL-запрос значительно продвинулось. Например, в [3] авторы предложили модель Seq2SQL – глубокую нейронную сеть для преобразования текстов в соответствующие SQL-запросы. Авторы также опубликовали набор данных WikiSQL, содержащий более 80 000 пар (текст, SQL-запрос), используемых в процессе обучения модели. Набор данных WikiSQL широко используется для обучения других нейронных сетей для преобразования текста в SQL-запрос, таких как SQLNet [4]. Авторы [5] представили метод на основе нейронной сети под названием IRNet для преобразования сложного текста в SQL-запрос. На наборе данных Sprider, используемого для тестирования метода преобразования текста в SQL, IRNet показала высокую точность. Сильные стороны нейронных сетей заключаются в том, что они не требуют написания больших наборов правил и могут хорошо обрабатывать множество различных структур естественного языка. Однако эти методы требуют больших объемов данных для обучения и значительных ресурсов для генерации SQL-запросов. Кроме того, они могут генерировать неточные или некорректные запросы.

3. Методы на основе большой языковой модели (LLM): Быстрое развитие LLM в последние годы открыло новые возможности для решения задачи преобразования текстового запроса на естественном языке в SQL-запрос. Авторы [6] провели сравнение точности моделей LLM Falcon и T5, используемых для решения этой задачи. Результаты исследования показали, что модель LLM Falcon с открытым исходным кодом имеет более высокую точность, достигая 70% на наборе данных Spider и 75% на наборе данных WikiSQL. Другая группа авторов оценила возможности преобразования текстового запроса на естественном языке в SQL-запрос с помощью языковой модели Codex. Это исследование показало, что модель Codex показывает высокую точность при ее тестировании на наборе данных Spider [7]. Авторы [8] использовали 12 различных наборов данных для демонстрации возможностей ChatGPT по преобразованию текстового запроса на естественном языке в SQL-запрос. Эти методы включают либо использование подсказок для

направления LLM на генерацию SQL-запросов [9, 10], либо дообучение (fine-tuning) окрытых LLM на наборах данных для преобразования запросов на естественном языке в формат SQL-запросов [11, 12]. Эти исследования показывают большой потенциал LLM для решения проблемы преобразования текстового запроса на естественном языке в SQL-запрос. Однако модели LLM требуют больших ресурсов для генерации запроса, что делает их мало пригодными для коротких, простых запросов. Кроме того, отсутствует объяснимость того, как и почему LLM сгенерировала SQL-запрос определенной конструкции на основе запроса на естественном языке, не говоря о том, что эти сгенерированные запросы могут быть неточными и зависимыми от обучающей выборки.

В данной статье представлен алгоритм, позволяющий преобразовывать нечеткие запросы на естественном языке в формат JSON-запроса без использования дорогостоящих моделей seq-to-seq или LLM, или дополнительных наборов правил. Также в статье приведены результаты сравнительного анализа производительности алгоритма при использовании в нем трех технологий векторизаций, TF-IDF, Word2Vec и SimHash, на 2000 нечетких запросов на естественном языке и соответствующих им запросов в формате JSON.

Материал статьи организован следующим образом. В разделе II приведено описание разработанного алгоритма. Раздел III посвящен сравнительному анализу результатов тестирования. В заключении подведены итоги проведенного исследования.

II. Алгоритм преобразования нечеткого запроса на естественном языке в формат JSON-запроса

A. *Описание предполагаемого алгоритма FindBestMatching*

Входными данными алгоритма являются:

X = {x1, x2, ...xn} – множество нечетких запросов на естественном языке.

Y = {y1, y2, ..., ym} – множество нечетких запросов в формате JSON.

v = vec(t) – функция векторизации, где t – текст, а v – вектор, соответствующий этому текст и построенный с применением некоторой технологии векторизации.

dist(v1,v2) – функция вычисления расстояния между векторами v1 и v2.

Выходными данными алгоритма является множество пар (x,y), где x ∈ X, y ∈ Y, и расстояние между x и y является минимальным.

Алгоритм FindBestMatching создает векторы для каждого элемента множеств X и Y с помощью функции vec(t). Затем для каждого элемента x, принадлежащего множеству X, FindBestMatching находит элемент y, принадлежащий множеству Y, таким образом, чтобы расстояние между векторами элементов x и y было минимизировано.

Приведенный ниже псевдокод содержит подробное описание этого алгоритма, где |X|, |Y| – количество элементов в множествах X и Y соответственно.

Algorithm FindBestMatching (X,Y, vec, dist)

```
1: VecX ← [ ]
2: for each x ∈ X do
3:     append vec(x) to VecX
4: end for
5: VecY ← [ ]
6: for each y ∈ Y do
7:     append vec(y) to VecY
8: end for
9: P ← [ ]
10: for i = 1 to |X| do
11:     d*← +∞; y* ← NULL
12:     for j = 1 to |Y| do
13:         d ← dist(VecX[i], VecY[j])
14:         if d < d* then
15:             d* ← d; y* ← Y[j]
16:         end if
17:     end for
18:     append (X[i], y*, d*) to P
19: end for
20: return P
```

B. *Анализ сложности*

Пусть n = |X|, m = |Y|, Tvec и Tdist – трудоёмкость одного вызова функций vec и dist соответственно. Преобразование всех элементов X и Y в векторы имеет сложность O((n + m)Tvec). Нахождение ближайшего элемента y, принадлежащего множеству Y, для каждого элемента x, принадлежащего множеству X, имеет сложность O(nmTdist). Общая сложность этого алгоритма составляет O((n + m)Tvec + nmTdist).

III. Результаты

A. *Датасет*

Для тестирования предложенного алгоритма было сгенерировано 2000 элементов множества X и 2000 соответствующих элементов множества Y. Часть полей сгенерированного множества запросов X имеет неоднозначные значения, например, «cheap», «standard» или «expensive» в поле «price». На рис. 1–2 показаны примеры элементов множеств X и Y.

x1: “Need a standard-priced worker for interior finishing work”
x2: “Need craftsmen for plastering, not doing grout sealing, young age no reviews”
x3: “Need an expensive worker for laminate flooring”
x4: “Find specialists who do wall leveling and not doing sanding older age good reviews”
x5: “Recommend a female master near Gostiny Dvor no reviews”
x6: “Looking for a male master near Vosstaniya Square no reviews”
x7: “Looking for a master near Gostiny Dvor for baseboard installation”
x8: “I want to hire a cheap repairman for window replacement”
x9: “Find a standard-priced repairman for grout sealing”
x10: “Need a specialist near Vyborgsky District for countertop installation”
…

Рис. 1. Примеры элементов множества X

y1:{"work": "interior finishing work", "price": "standard"}
y2: {"work": "plastering, no grout sealing", "age": "young age", "reviews": "no reviews"}
y3: {"work": "laminate flooring", "price": "expensive"}
y4: {"work": "wall leveling, no sanding", "age": "older age", "reviews": "good reviews"}
y5: {"gender": "female", "location": "Gostiny Dvor", "reviews": "no reviews"}
y6: {"gender": "male", "location": "Vosstaniya Square", "reviews": "no reviews"}
y7: { "work": "baseboard installation", "location": "Gostiny Dvor"}
y8: {"work": "window replacement", "payment": "cheap"}
y9: {"work": "grout sealing", "payment": "standard"}
y10: {"work": "countertop installation", "location": "Vyborgsky District"}
...

Рис. 2. Примеры элементов множества Y

B. Реализация алгоритма

Предложенный алгоритм реализован на языке Python с использованием трех различных технологий векторизации: TF-IDF, Word2Vec и SimHash. Особенности применения этих технологий:

- TF-IDF: в качестве словаря для обучения TF-IDF использовалось множество X ∪ Y, а в качестве функции расстояния – косинусное расстояние.
- Word2Vec: использовались векторы слов, обученные в [9]; каждый запрос представлен средним значением суммы векторов его слов; в качестве функции расстояния – косинусное расстояние .
- SimHash: текст преобразуется в 64-битное двоичное число; в качестве функции расстояния используется расстояние Хэмминга.

C. Показатель точности преобразования

Точность преобразования определяется по следующей формуле:

$$Acc = \alpha/\beta * 100\%,$$

где α – количество правильных пар (x,y), β – количество элементов во множестве X.

Разметка «правильных» пар проводилась на основе мнения экспертов.

IV. Результаты

В табл. I представлены значения точности преобразования запросов на естественном языке в формат JSON предлагаемого алгоритма в сочетании с одной из трех технологий векторизации.

ТАБЛИЦА I. Точность преобразования запросов на естественном языке в формат JSON предлагаемого алгоритма в сочетании с различными технологиями векторизации на N = 2000 запросах.

Технология векторизации	Точность преобразования
TF-IDF	**99.20%**
Word2Vec	62.60%
SimHash	99.15%

На основе табл. I построена следующая столбчатая диаграмма:

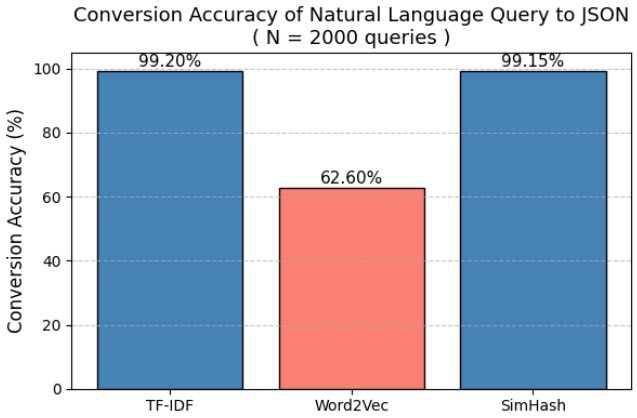

Рис. 3. Точность преобразования запроса на естественном языке в JSON

На рис. 3 показано, что TF-IDF обеспечивает наилучшую точность преобразования – 99,20%.

V. Заключение

В данной статье представлен простой, но эффективный алгоритм преобразования нечетких запросов, написанных на естественном языке, в формат JSON. Предложенный алгоритм рассматривает преобразование запроса как поиск ближайшего элемента в векторном пространстве к запросу на естественном языке и не зависит от выбранной технологии векторизации текста.

Тестирование 2000 пар запросов показал, что технологии векторизации TF-IDF и SimHash обеспечивают точность преобразования выше 99%, значительно превосходя Word2vec. Однако в текущем алгоритме отсутствует механизм исключения в случаях, когда ни один элемент множества Y не соответствует запросу на естественном языке. Этот механизм исключения будет разработан в будущем. В настоящее время ведется разработка алгоритмов нечетких запросов, принимающих JSON в качестве входных данных.

Список литературы

[1] T. Yu et al., “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task,” presented at the Proceedings of the 2018 conference on empirical methods in natural language processing, 2018, pp. 3911–3921.

[2] G. Katsogiannis-Meimarakis and G. Koutrika, “A survey on deep learning approaches for text-to-SQL,” The VLDB Journal, vol. 32, no. 4, pp. 905–936, Jul. 2023, doi: 10.1007/s00778-022-00776-8.

[3] V. Zhong, C. Xiong, and R. Socher, “Seq2sql: Generating structured queries from natural language using reinforcement learning,” arXiv preprint arXiv:1709.00103, 2017.

[4] X. Xu, C. Liu, and D. Song, “SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning,” Nov. 13, 2017, arXiv: arXiv:1711.04436. doi: 10.48550/arXiv.1711.04436.

[5] J. Guo et al., “Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation,” in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy: Association for Computational Linguistics, 2019, pp. 4524–4535. doi: 10.18653/v1/P19-1444.

[6] V. Câmara, R. Mendonca-Neto, A. Silva, and L. Cordovil, “A Large Language Model approach to SQL-to-Text Generation,” in 2024 IEEE International Conference on Consumer Electronics (ICCE), Jan. 2024, pp. 1–4. doi: 10.1109/ICCE59016.2024.10444148.

[7] N. Rajkumar, R. Li, and D. Bahdanau, “Evaluating the Text-to-SQL Capabilities of Large Language Models,” 2022, arXiv. doi: 10.48550/ARXIV.2204.00498.

[8] A. Liu, X. Hu, L. Wen, and P. S. Yu, “A comprehensive evaluation of ChatGPT’s zero-shot Text-to-SQL capability,” 2023, arXiv. doi: 10.48550/ARXIV.2303.13547.

[9] Chang S., Fosler-Lussier E. How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot, Single-domain, and Cross-domain Settings. arXiv, 2023. doi: 10.48550/arXiv.2305.11853.

[10] Sun R. и др. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (extended). arXiv, 2023. doi: 10.48550/arXiv.2306.00739.

[11] Kulkarni A., Srikumar V. Reinforcing Code Generation: Improving Text-to-SQL with Execution-Based Learning. arXiv, 2025. doi: 10.48550/arXiv.2506.06093. doi: 10.48550/arXiv.2506.06093.

[12] Sarker S. и др. Enhancing LLM Fine-tuning for Text-to-SQLs by SQL Quality Measurement. arXiv, 2024. doi: 10.48550/arXiv.2410.01869

[13] E. Gruss, eyaler/word2vec-slim. (Apr. 14, 2026). Python. Accessed: May 14, 2026. [Online]. Available: https://github.com/eyaler/word2vec-slim.