

Tempura: Query Analysis with Structural Templates

Tongshuang Wu*
University of Washington
wtshuang@cs.washington.edu

Kanit Wongsuphasawat
Apple Inc.
kanitw@apple.com

Donghao Ren
Apple Inc.
donghao@apple.com

Kayur Patel
Apple Inc.
kayur@apple.com

Chris DuBois
Apple Inc.
cdubois@apple.com

ABSTRACT

Analyzing queries from search engines and intelligent assistants is difficult. A key challenge is organizing queries into interpretable, context-preserving, representative, and flexible groups. We present *structural templates*, abstract queries that replace tokens with their linguistic feature forms, as a query grouping method. The templates allow analysts to create query groups with structural similarity at different granularities. We introduce *Tempura*, an interactive tool that lets analysts explore a query dataset with structural templates. Tempura summarizes a query dataset by selecting a representative subset of templates to show the query distribution. The tool also helps analysts navigate the template space by suggesting related templates likely to yield further explorations. Our user study shows that Tempura helps analysts examine the distribution of a query dataset, find labeling errors, and discover model error patterns and outliers.

Author Keywords

Natural Language Processing; Error Analysis; Query Analysis

CCS Concepts

•Information systems → Content analysis and feature selection; Presentation of retrieval results; •Human-centered computing → Visual analytics;

INTRODUCTION

We expect modern search engines and intelligent agents to answer knowledge based questions. While the knowledge representation and natural language processing communities have made great strides [7, 26], these systems still fail at times. To improve performance, developers need to analyze the query data to understand where models work well, and where they can be improved: What are the types of queries that users ask, and how well do their models work for these queries?

*The work was done when the author was an intern at Apple Inc.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CHI '20, April 25–30, 2020, Honolulu, HI, USA.
Copyright is held by the author/owner(s).
ACM ISBN 978-1-4503-6708-0/20/04.
<http://dx.doi.org/10.1145/3313831.3376451>

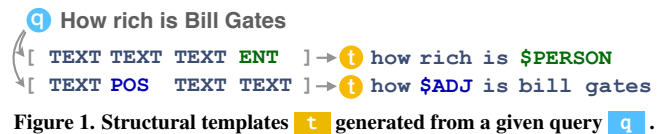


Figure 1. Structural templates **t** generated from a given query **q**.

Such analysis is typically iterative and seeded in two ways. First, developers may start by looking at aggregate performance metrics like model accuracy, and then drill down into more actionable groups [39, 44]. Alternatively, they may start with a specific example that represents an experience, and try to understand if the example is part of a general trend [41]. In both cases, developers need to find coherent query groups larger than one example and smaller than the entire dataset.

Grouping instances to understand performance is easy for structured data. For example, given a tabular data, developers can easily group and slice it by an explicit, categorical feature (or column). However, grouping is more difficult with unstructured data like text. For instance, text clustering [1] or topic modeling [6] can produce groups, but those groups are often difficult to understand. They can hardly distinguish the similarity between “how rich is Bill Gates” with “how rich is Jeff Bezos”, or with “how old is Bill Gates.” Developers often try to rationalize those groups by looking at examples within the group and guessing what the algorithm did. In fact, various existing grouping methods all face certain challenges. We analyze their limitations, and formalize four requirements for a grouping strategy: To inspire actionable analysis, the generated groups should be *interpretable* and *context-preserving*. To enable unbiased analysis, the groups should be *representative* of the dataset. Finally, groups should be *flexible* enough to cope with developers’ varying needs.

While text data are usually tricky, queries are easier to group than longer forms of text (e.g., from articles or books). This is because queries are generally short and often share similar structures. For example, “how rich is Jeff Bezos” and “how rich is Bill Gates” are both one sentence questions. They are also both identically structured “how rich” questions about a person. In this paper, we aim to leverage this compact structure in text queries to help developers explore query datasets, and create meaningful groups for query analysis.

First, to group queries, we propose an alternative grouping strategy via *structural templates*, which replace query tokens with abstract ones based on linguistic features. For instance, the template “how rich is \$PERSON” abstracts the query

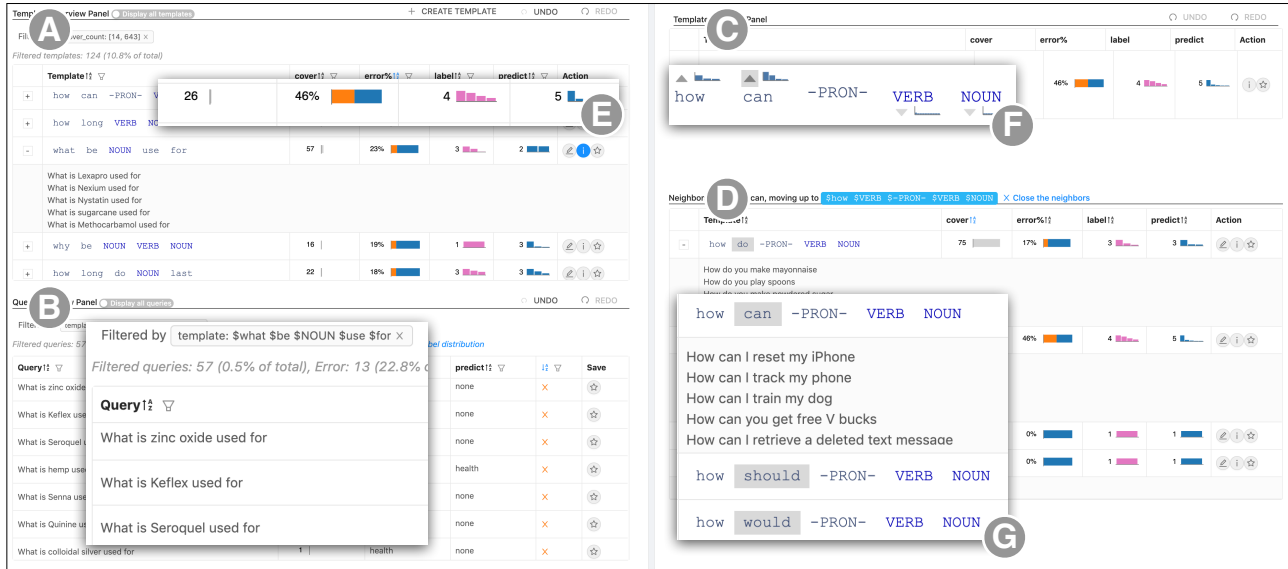


Figure 2. Tempura interface. (A) The *template overview panel* lists structural templates automatically selected to represent the dataset’s query distribution. Each row is associated with attributes summarizing their corresponding queries (E, details in Fig. 5). (B) The *query panel* lists individual queries in the dataset. (C) The *template traversal panel* allows users to navigate related templates. A focused template selected for traversal (F, enlarged in Fig. 8) has visual indicators suggesting related templates to explore. Clicking on the traversal options attached to each linguistic feature opens up the neighbor template table (D). The table displays the corresponding parents or children, with the differing linguistic feature highlighted (G). Like in (G), all templates allow previewing the example queries covered by an expanded template. Video demo: https://youtu.be/s_ODGuZU4G8.

“how rich is Bill Gates” in Fig. 1 by replacing the token “Bill Gates” with its named entity type, **\$PERSON**.

Structural templates satisfy the aforementioned requirements. Directly generated from the query dataset, they can *represent* the data distribution. Linguistic features used in their generations help *preserve context*. They also help developers *interpret* and map between templates and the corresponding queries. As we generate many templates for each query using different combinations of linguistic features, analysts can *flexibly* select templates to explore the query dataset from various aspects. For example, instead of replacing “Bill Gates” with **\$PERSON** to find rich celebrities, one can instead replace “rich” with a part of speech (POS) tag **\$ADJ** to explore other queries about “Bill Gates” (e.g., “how *old* is Bill Gates”). Analysts can also change the granularity of the grouping. From the template “how rich is **\$PERSON**”, one can further abstract **\$PERSON** into a POS tag **\$NOUN**. The resulting template “how rich is **\$NOUN**” will then include previously omitted queries like “how rich is *feta cheese*”, and reveal ambiguities in the word “rich”.

Next, to help developers analyze queries with structural templates, we introduce an interactive system called *Tempura*—**template based query analysis** (Fig. 2). With Tempura, developers may begin their analyses by looking at the *template overview panel*, which presents a set of templates to summarize the distribution of a query dataset. Alternatively, they may find a particular query of interest in the *query panel*. Via *interactive template traversal*, developers may navigate between query groups with different grouping aspects and granularities. Underlying Tempura, we present an entropy-based measure to suggest templates that developers may want to traverse to. Based on this measure, we also develop a *template summarization algorithm* to select a template subset for the overview.

We evaluate Tempura both in terms of the template selection algorithm and the system. We demonstrate that the algorithm can reveal distributional differences between three datasets including Natural Questions [22], MS MARCO [28], and internal query logs from a commercial conversational assistant. In a user study, eight experienced machine learning (ML) developers analyzed a query dataset and its modeling performance with Tempura. Each of them was able to make around five interesting observations on either the data or the model. These findings included better understandings on dataset distribution, labeling noises, as well as patterns and outliers in the model’s errors — some of which are actionable observations that can help them prioritize their decisions on data cleaning and augmentation. Our users agreed that Tempura helped them make more observations with less effort, and suggested additional tasks where the structural template can be useful.

UNDERSTANDING QUERY ANALYSIS AND GROUPING

We interacted closely with a team of seven ML developers who work on query analysis for six months. We studied their workflows through weekly informal interviews and direct observations. Here, we summarize their two primary query analysis tasks, both requiring coherent query groupings. We then analyze the limitations of existing grouping methods, and identify four requirements for effective grouping.

Key Tasks: Dataset Exploration and Error Analysis

Developers typically need to perform two primary query analysis tasks. The first task is *dataset exploration (T1)*, in which they broadly explore their data and perform sanity checks. These sanity checks help developers ensure that their sampled data meet their modeling needs (e.g., not under-representing certain use cases) and do not have any data quality issues (e.g., conflicting labels on similar queries). Data exploration

	Interpretable (G1)	Context- preserv. (G2)	Representative (G3)	Flexible (G4)
Query properties	✓	✗	✓	✗
Manual labeling	✓	✓	✗	✓
Clustering	✗	✓	✓	✗
Structural templates	✓	✓	✓	✓

Table 1. Four requirements for effective query groupings (G1-G4). Existing query grouping methods all have limitations. Meanwhile, groups created by our structural template satisfy all the requirements.

requires comparing the distributions of different query groups (similarly phrased queries, those with the same labels, etc.)

The second task is *error analysis* (T2), in which practitioners examine wrong predictions (e.g., from bug reports of deployed systems [17]) to understand their models’ deficiencies. In this case, developers instead usually start with an individual query, and then find if certain errors generalize to systematic error patterns. Although these two tasks have different starting points, in both cases developers need coherent groupings of queries to perform their analyses.

Query Grouping: Existing Methods & Requirements

We analyze limitations of existing query grouping approaches, and identify four requirements for effective grouping methods:

- G1 *Interpretable*. Groups should have clear definitions, such that developers can predict what queries are included.
- G2 *Context-preserving*. Groups should consider the actual sentence context of the queries, not just coarse properties.
- G3 *Representative*. Groups should reflect the dataset distribution, without biasing towards or overlooking queries.
- G4 *Flexible*. Groups should help developers assess queries that are similar (G4a) from different *aspects* (“how should the queries be similar”) and at (G4b) different *granularities* (“how similar they should be”).

In practice, developers frequently grouping by *query properties* like query length or query types (e.g., question words). However, such groupings only convey those particular attributes, and thus *preserve too little context* (✗ G2) from the queries to form groups that reveal actionable insights. One developer said “*Knowing my model performs poorly on long queries does not lead to next steps.*” Alternatively, developers would assign query groups based on domain knowledge, either manually or through filtering scripts. Such approach may introduce the developers’ biases and cause the groupings to be *unrepresentative* (✗ G3). One could mistakenly filter and focus on a group of “how rich” questions even when the dataset doesn’t contain many such examples [41].

Clustering algorithms compensate both issues, capturing more sentence meaning than query properties and extracting representative groups directly from the dataset. However, developers only rarely rely on them. They are concerned that clustering algorithms may chain unrelated queries together, making the groups less coherent. For example, “how rich is Jeff Bezos” and “how old is Bill Gates” can be in the same cluster, just because they are both similar to “how rich is Bill Gates.” As a result, developers usually have a hard time *interpreting* the groups (✗ G1). They also cannot *flexibly* inspect how the model respond to “how rich” and “how old” (✗ G4).

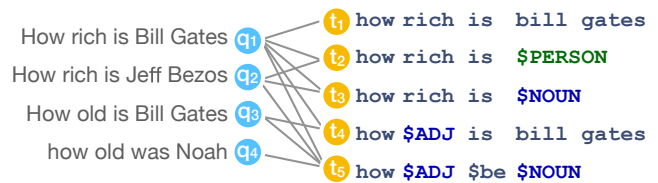


Figure 3. Queries and their generated templates form a many-to-many relationship. Each query generates multiple templates, and each template covers a different group of queries.

THE CONCEPT OF STRUCTURAL TEMPLATE

To develop a grouping method that better satisfies aforementioned requirements, we leverage the fact that queries are typically short and often share similar sentence structure, and propose grouping queries via structural templates. Here, we first describe how we generate structural templates from a query, and how a template can represent a query group. We then explain the hierarchical relationship between templates, which allows analysts to explore the space of similar queries.

The Assumption on Query Structures

Tempura is built on the assumption that queries are properly structured to support part-of-speech tagging. While prior work describes real-world queries as potentially under-specified, terse, and context-dependent [4, 5, 18, 35], our assumption is valid here, as we focus on a subset of real-world queries: *knowledge-seeking questions*. Filtering to a query subset is common in practice for routing queries based on their format, intent, or context. For example, ill-defined queries will trigger hard-coded responses like “Sorry I don’t understand.” As a result, we work on self-contained questions, which are usually cleaner and are in line with the interests of developers we interacted with. The questions are also mostly well-formed in various question answering datasets (e.g., SQuAD [31], VQA [2]) and public query datasets (e.g., Natural Question and MS MARCO similarly discard non-question queries), to which Tempura would directly apply. That said, the syntactic patterns of ill-structured queries are still useful, and we envision how Tempura can be extended for those queries in *Discussions*.

Template Generation: Abstracting Query Tokens

We define a *structural template* to be an abstract query that replaces their original tokens with *abstract tokens*. To produce abstract tokens, we use four linguistic features, listed from the most specific to the most abstract:¹

1. Lower cased original TEXT (“How” → “how”)
2. Normalized word form LEMMA (“is” → “\$be”)
3. Named entity type ENT (“Bill Gates” → “\$PERSON”)
4. Part-of-speech POS (“Bill Gates” → “\$NOUN”)²

Each unique combination of the query’s linguistic features produces a template. In Fig. 1, the linguistic features [TEXT, TEXT, TEXT, ENT] generate the template that has abstract tokens [“how”, “rich”, “is”, “\$PERSON”]. Because we extract multiple linguistic features from each token, a single query can produce many templates, each with a different

¹ We color the abstract tokens in the templates based on their linguistic features: *text*, *\$lemma*, *\$named entity*, or *\$POS tag*.

² We merge n-grams entities and noun chunks, such that “Bill Gates” can be treated as \$PERSON or \$NOUN, rather than \$PERSON \$PERSON.

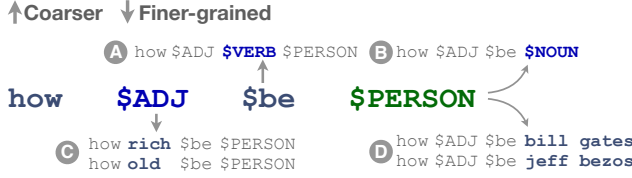


Figure 4. Parent and child templates for “how \$ADJ \$be \$PERSON”, by tweaking its different linguistic features: (A) is a parent template found by merging \$be with other possible LEMMA (e.g., \$do) to \$VERB (POS). Similarly, (B) merges \$PERSON with other possible ENT to get \$NOUN (POS). (C) and (D) are two sets of child templates *splitting* (C) the \$ADJ and (D) \$PERSON into actual LEMMA forms.

set of abstract tokens. For instance, “how rich is Bill Gates” (q_1 in Fig. 3) can generate 5 templates. Each template in turn covers multiple queries, and therefore represent a *query group*. In Fig. 3, t_2 (“how rich is \$PERSON”) represents a group of two queries (q_1 and q_2).

Templates provide several advantages. First, by pivoting query tokens in a controlled manner, we create template-based groups that are *interpretable* (G1) — analysts can easily map between queries and templates. Second, because templates mirror the syntactic structures of the grouped queries, these groups are *context-preserving* (G2). Finally, as we extract templates from every query in a dataset, the resulting templates can *represent* the dataset distribution (G3).

Hierarchy of Templates with Different Granularity

Because the linguistic features have a ladder of granularity, the templates generated with these linguistic features form an *abstraction hierarchy*. For example, person names and locations are both nouns, so tokens with named entity \$PERSON and \$LOC will all have \$NOUN as the POS tag.³ As a result, the set of queries covered by “how rich is \$PERSON” (t_2 in Fig. 3) is always a subset of those in “how rich is \$NOUN” (t_3). In other words, t_3 is an abstraction of t_2 .

The hierarchy provides *flexibility* (G4). Analysts can traverse the hierarchy to examine query groups at different levels of granularity (G4b). By changing a linguistic feature in a template, one can move up to its *parents* (templates that are immediately one level less specific) or down to its *children* (immediately one level more specific). Consider the template “how \$ADJ \$be \$PERSON” in Fig. 4: changing the LEMMA “\$be” to its abstract POS tag “\$VERB” generates the parent (A), which includes other \$VERBs besides \$be. Similarly, replacing “\$PERSON” with its abstract POS tag “\$NOUN” produces the parent (B), which covers more general questions by including other \$NOUNs beyond \$PERSON names. On the other hand, splitting “\$ADJ” and “\$PERSON” to their LEMMA forms generate four children, two for (C) and two for (D).

Analysts can also traverse the hierarchy to examine groups from *different aspects* (G4a). For example, from a template that focuses on a specific attribute for different people (“how rich \$be \$PERSON” in Fig. 4C), we can pivot to concentrate on different attributes of the same person (“how \$ADJ \$be bill gates” in Fig. 4D). To do so, we can first replace

³We merge \$NOUN and \$PROPN for simplicity.

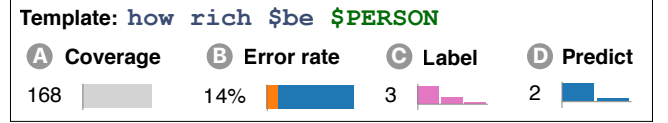


Figure 5. A template’s attributes, summarizing its corresponding query group: (A) the number of queries covered; (B) the error rate; and the label distribution of queries’ (C) ground truths and (D) predictions.

“rich” with its abstract POS tag “\$ADJ”, and then replace “\$PERSON” with the TEXT “bill gates”.

THE TEMPURA INTERFACE

We develop an interactive system, *Tempura*, to help developers use structural templates for query analysis (Fig. 2). In this section, we present how the Tempura interface lets developers explore the queries and generated templates by showing overviews first and providing details on demand [38].

Overviews: Bi-directional Starting Point

The Tempura interface provides both template and query overviews. The *template overview* panel (Fig. 2A) lists a set of templates to summarize the query distribution as a starting point for dataset exploration (T1). We later describe how Tempura selects templates for this overview in the *Template Summarization* section. The *query* panel (Fig. 2B) lists individual queries and lets analysts start error analysis (T2) with one query. To help users locate templates and queries to inspect, both panels support searching with regular expression.

Both panels also display attributes associated with each template and query, and allow users to sort the table entries based on these attributes. The query panel shows a ground truth and a predicted label per query. Meanwhile, the template overview panel presents attributes of each template’s corresponding query group (Fig. 2E), including *coverage* (the number of queries covered by the template) and other statistical attributes relevant to the modeling task (explained below).

The template overview is task-agnostic, as the table can show different statistics for different modeling tasks. For example, Fig. 5 shows attributes for a text classification task. The stacked bar chart in Fig. 5B indicates the model performance with *error rate*, or the ratio of incorrectly predicted queries associated with a template. The distribution bar glyphs in Fig. 5C and 5D show class distributions for the ground truth and the prediction respectively. In this example, the template contains three ground truth labels, but only two predicted labels (with one dominating). These bar glyphs reveal that the covered queries receive similar predictions, regardless of the actual \$PERSON in the query. In contrast, if the template contains only one ground truth label but many predicted labels, the model is likely not robust, and thus can be easily swayed by \$PERSON. To see more details about the distributions, users can click on the glyphs to open a popup window showing an enlarged bar chart (Examples in Fig. 9).

Details-on-demand: Multi-aspect/granularity Traversal

After starting their analyses from the two overviews, analysts can further explore query groups of interest on demand with *template traversal*, and thereby get varying views of the data.

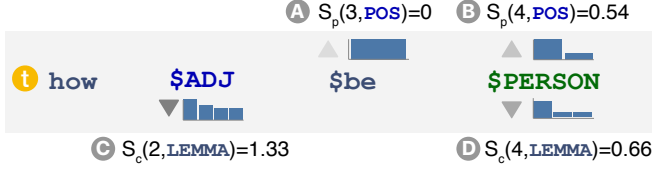


Figure 6. Given a focused template $t = \text{"how \$ADJ \$be \$PERSON"}$, we can traverse to its parents (up arrows on top of $\$be$ and $\$PERSON$), or children (down arrow under $\$ADJ$ and $\$PERSON$). The color density of the arrows indicates how informative a traversal is. Bar charts denote the coverage distribution of the templates affected by the traverse.⁴

Basically, analysts can traverse the template hierarchy, by moving up to inspect coarser (more abstract) parents or down to examine finer-grained (more specific) children templates.

To explore different query groups with template traversal, analysts can first select a template of interest (Fig. 2A). Based on the selection, the traversal panel (Fig. 2C) shows a *focused template* on the top (Fig. 2F). For each token in the template, the panel presents an arrow and a bar glyph to indicate potential traversal options. The arrow denotes the corresponding token’s traversal direction: moving to children (down) or a parent (up). For example, the arrows in Fig. 6 indicate the corresponding two parent and two children sets in Fig. 4.

The bar glyphs help preview the templates related to corresponding traversals. For a downward traversal, the bar glyph shows the children templates and their coverage. The four bars in Fig. 6C indicates that moving from $\$ADJ$ to its children will generate 4 templates (“how { $\$old$, $\$much$, $\$big$, $\$strong$ } $\$be$ $\$PERSON$ ”). We can also see that the coverage of the four children is roughly even. For an upward traversal, the bars represents *sibling* templates (templates with the same parent). The bar glyph in Fig. 6B shows that moving from $\$PERSON$ to $\$NOUN$ will merge the current template “how $\$ADJ$ $\$be$ $\$PERSON$ ” with another sibling template “how $\$ADJ$ $\$be$ $\$LOC$ ”, which has a lower coverage.

When analysts select a traversal direction (by clicking an arrow), Tempura creates a neighbor template table (Fig. 2D), which lists all corresponding neighbor templates shown in the bar glyph and their attributes. As children or siblings of the focused template, these templates only differ from each other by one token. Therefore, showing them in one table helps contrast similar query groups.

To help analysts avoid tediously enumerating all the traversal options, we further compute *traversal scores*, and use them to compare and rank all the traversal options. The score measures how informative a traversal action is, or which traversal is likely to yield further exploration. For example, moving from a coarse parent template to its children is informative, when the move provides multiple representative subgroups. We use the color density of the arrow to encode the score. The darker the arrow, the more *preferable* the traversal. In Fig. 8C, the dark arrow suggests traversing down $\$ADJ$ — an exact example of “provides multiple representative subgroups.” We formalize this score in the next section.

⁴For simplicity, we omitted t from the entropy equations (e.g., $S_c(2, LEMMA)$ in C represents $S_c(t, 2, LEMMA)$).

SCORING TRAVERSALS

We introduce the traversal score used to color traversal arrows in the interface. We want to guide the analyst towards templates that have only a few abstract tokens, because they are more interpretable (G1) and context-preserving (G2) than those containing only abstract tokens. However, suggested templates should also cover enough queries to ensure that they are representative enough to be worth exploring.

Suppose an analyst starts with the template t in Fig. 6. She can make $\$ADJ$ less abstract (C), to inspect all the child templates that have a LEMMA for that token. This action “splits” the current template into finer-grained templates, as in Fig. 4C. We encourage this traversal when each of the split children covers a significant number of queries. Conversely, this traversal is not useful when only one or a few children cover the majority of queries associated with t . In such cases, analyzing the high-coverage child yield a slightly more specific subgroup, but at the cost of seeing less queries overall.

To help users weigh this tradeoff, we define a *traversal score* for each available traversal starting from a template t . Let t_i be the abstract token at position i in t , and $|t|$ be the template coverage. Let $C(t, i, l) = \{s \in \text{Children}(t) : s_i = l, |s| > 1\}$ be the subset of children covering more than one query, where we have set the i -th token to be a less abstract token, using a less abstract linguistic feature l . In our previous $\$ADJ$ case, the templates in Fig. 4C (also Fig. 6C) are examples of $C(t, 2, LEMMA)$. Because all the templates in $C(t, i, l)$ differ on the i -th abstract token, their covered queries are mutually exclusive. For traversing to child templates, $S_c(t, i, l)$, we use the entropy of the normalized coverage of a template’s children,

$$S_c(t, i, l) = - \sum_{s \in C(t, i, l)} \frac{|s|}{|C(t, i, l)|} \log \frac{|s|}{|C(t, i, l)|}$$

where $|C(t, i, l)| = \sum_{s \in C(t, i, l)} |s|$.

We use the entropy to encourage traversals where each of the children have a similar amount of coverage. We discourage traversals when the distribution of coverage among those children is “peaked”; in those cases, the resulting templates have reduced coverage while providing only a small increase to the coherence for each subgroup of queries (G1).

Similar to traversing towards children, analysts can traverse to parent templates, i.e., use a more abstract linguistic feature l for i -th token in t . While traversing towards children splits a template, traversing to a parent conversely merges *sibling templates* (templates that have the same linguistic features as t and have the same parent). For example, templates in any $C(t, i, l)$ are siblings of each other. Just as before, we define a traversal score which encourages traversing to parents whose child templates each cover a significant number of queries. For the parents of t that arise from making token i more abstract, we compute the same entropy score as before, but instead normalize over the siblings of t , $R(t, i, l) = \{s \in \text{Siblings}(t) : s_i = l\}$. We denote this score $S_p(t, i, l)$.

TEMPLATE SUMMARIZATION

We now present an algorithm that helps select templates for the template overview panel (Fig. 2A). Our goal is to select a set of

representative templates to summarize the query distribution, and use it as a starting point for dataset exploration (T1).

Intuition: Two Aspects to Consider

To construct representative groups of queries, we prefer templates with high *coverage*. Meanwhile, to effectively initiate the exploration, we want to include templates that are *more preferred* than any of its parents or children. If the parents or children are preferred, analysts will need to perform extra traversing steps to reach the interesting grouping structure.

The *traversal score* defined previously provides a ranking between a given template and all its parents and children. A template t should be selected when we *do not* encourage traversing to any of its parents or children. We define $I(t)$ to be the *maximum* of t 's traversal scores — all the $S_c(t, i, l)$ and $S_p(t, i, l)$ for different tokens i and abstract tokens l :

$$I(t) = \max(\max_{i,l} S_c(t, i, l), \max_{i,l} S_p(t, i, l))$$

Recall that traversal scores represent preferences *towards* parents and children. With $I(t)$ being the maximum, the lower it is, the less likely t should be split or merged at any tokens, and therefore the more likely we want to include it in the overview.

Selection as a Weighted Set Cover Problem

To take these two aspects into consideration, we form the template selection as a weighted set coverage problem. We see the query dataset $Q = \{q_1, \dots, q_n\}$ as the entire set of elements. Then, each template t in $T = \{t_1, \dots, t_m\}$ represents a subset of Q that contains a number of queries $|t|$ (the template *coverage*). We use $I(t)$ as the weights, such that templates with low preferences are penalized by having a high weight. Our goal is to find a set $T^* \subset T$ such that (1) T^* covers at least a user-specified ratio, c , of queries: $|T^*| = |\bigcup_{t \in T^*} t| \geq c|Q|$; and (2) the sum of the weights of the subsets in T^* is minimized.

Algorithm 1: Template selection

Data: query set Q , generated templates T , min cover. ratio c

Result: a list of overview templates T^*

```

1  $T^* = \{\}$ ;
2 while  $|T^*| < c|Q|$  do
3    $t^* = \arg \min_{t \in T} I(t) / (|T^* \cup \{t\}| - |T^*|)$ ;
4    $T^* = T^* \cup \{t^*\}$ ;
5 return  $T^*$ 
```

Weighted set coverage is a NP-complete problem. Here we use a classic greedy algorithm (Algorithm 1) to compute an approximate T^* [43]. The algorithm repeatedly chooses a template t that minimizes the weight $I(t)$ divided by number of queries in t not yet covered by the chosen templates $(|T^* \cup \{t\}| - |T^*|)$. It then stops and returns the chosen templates (T^*) when they form a cover of the original set of queries.

We experiment our selection algorithm on different query datasets. Compared to alternative weighting strategies, our algorithm selects a larger number of templates, but tends to select more interpretable and context-preserving “*what \$be \$NOUN*” rather than “*what \$VERB \$NOUN*”). We find the algorithm can reduce the exploration burden: Heuristically, to

Dataset	$l(q)$	$ T $	$\# t > 1$
NQ	7.6 ± 1.8	2,662,618	16,976 (0.66%)
MS MARCO	5.2 ± 2.5	992,483	6,601 (0.66%)
Assistant	5.3 ± 1.6	509,734	29,732 (5.83%)

Table 2. The tested datasets with their query lengths $l(q)$, and the attributes on generated templates. From the total number of generated templates ($|T|$) and the proportion of those covering at least two queries ($|t| > 1$), we see NQ and MS MARCO are more sparse than Assistant.

cover 75% queries of a dataset, the method selects a number of templates that is around 10% of the dataset size (for a 10,000 query dataset, the method selects around 1,000 templates).

Case Study: Templates Selected from Different Datasets

To test whether our automatic template generation and summarization can reveal dataset characteristics, we use Tempura to process three datasets:⁵ (1) Natural Question (NQ) [22] and (2) MS MARCO [28], which are collections of real queries issued to Google and Bing Search Engines, respectively, as well as (3) anonymized search queries from a commercial conversational assistant (Assistant) that occur on more than 10 different devices. The queries are all seeking general knowledge (not related to personal information). They are automated speech recognition transcripts with no audio attached.

These datasets are used for training question answering systems [20]. We are interested in understanding if queries from different sources differ. To explore this, we sample 10,000 queries from each dataset (with the training and the development set combined) and generate templates from them. We report two major results below.

Crafted research datasets are sparser. The number of generated templates in Table 2 reveals NQ’s apparent distributional difference. It generates many more unique templates (around 2.5 million) compared to the other two alternatives (within 1 million). This is likely because additional filters on NQ only keep queries that have (1) more than eight words⁶, and (2) a closely related Wikipedia page. The cartesian product of linguistic features then generates more templates from the longer queries. In other words, while the filters help locate complex questions, they also shift the distribution to be less representative of the natural queries. While MS MARCO generates fewer templates than NQ, it still contains more diverse queries than Assistant: Both NQ and MS MARCO only have 0.66% templates with coverage greater than 1.

Real-world queries have different distributions. Further comparing the less sparse MS MARCO and Assistant, we notice that the most representative templates differ. While “*what be \$NOUN*” is an important template in both MS MARCO and Assistant, this template has a much higher coverage in the former. The selected templates diverge afterwards: MS MARCO has many “*\$NOUN*” (e.g., “401k loan requirements”, “employee turnover types”) while Assistant has more “*what \$do \$NOUN mean*” queries. The “*\$NOUN*” case in MS MARCO is especially interesting: all the query datasets were filtered to only keep knowledge-seeking question queries, yet the other two didn’t

⁵We augment linguistic features with SpaCy (<https://spacy.io/>).

⁶NQ’s average query length appears to be less than eight in the table, as we merged noun chunks.

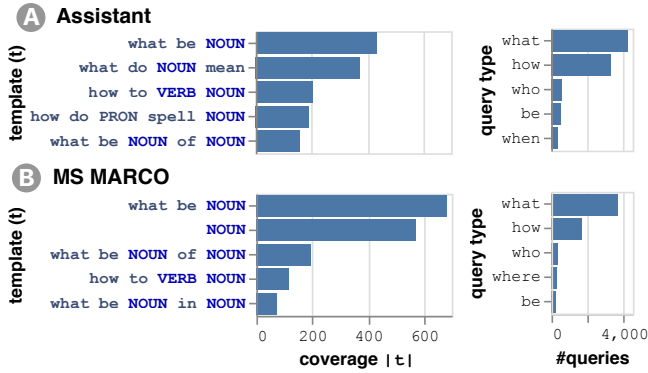


Figure 7. The five most representative templates and most frequent query types for the sampled queries in (A) Assistant, and (B) MS MARCO. They have similar query types, but their templates differ after “what \$be \$NOUN”. This indicates that the templates can capture more dataset characteristics than the conventional query types.

show any \$NOUN queries, indicating different filtering strategies. Such dataset differences are difficult to notice from just the query types (in Fig. 7) or query length (in Table 2).

USAGE SCENARIO

We present a scenario to demonstrate how developers can use Tempura to better evaluate a BERT-based classification model [7] on a query dataset. The task is to predict if a query is in one of four categories: *food* (“can I freeze spaghetti”), *health* (“what is Keflex used for”), *procedural* explanations on how to perform some tasks (“how to clean white shoes”), or *none* of above. The dataset contains 10,000 anonymized, knowledge-seeking queries from Assistant (75% of which are covered by the selected overview templates). It is a sample of the development set that the developers are building a model on. This scenario is inspired by the analyses that participants perform in our user study, which uses the same data.

The developer first filters the templates to only keep those covering more than ten queries, and sorts them by their error rate in descending order (Fig. 10). The template with the highest error rate is “how \$can \$-PRON- \$VERB \$NOUN”. Out of the 26 covered queries, 46% are incorrectly predicted. Inspecting them, he notices **noisy labels**, namely similar queries are labeled as asking for a *procedural* explanation (“how can I boil eggs”⁷) and about *food* (“how can I cook salmon”). The developer verifies that queries on cooking procedures make these two label pairs non-exclusive. He treats this observation as supporting evidence to switch to a multi-label classifier.

Afterwards, the developer explores similar templates via template traversal. With the focused template panel (Fig. 8), he notices the dark arrow in B, which suggests that merging \$can into its parent POS form (\$VERB) is the most informative traversal. Other traversals indeed seem less useful. The bar charts associated with \$VERB and \$NOUN (C) show that they have a large number of sparse and unrepresentative children. Meanwhile, merging \$how (A) yields a very little gain, as its two neighboring templates have few queries associated with them.

⁷All the queries referred in the paper have a query frequency larger than 10, i.e., occur on at least ten unique devices.



Figure 8. Traversing a focused template. The dark arrow in B suggests moving from \$can to its parent form \$VERB.

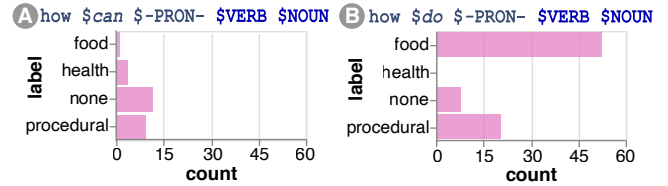


Figure 9. The ground truth label distributions for queries covered by two closely related neighboring templates, with (B) “how \$do \$-PRON- \$VERB \$NOUN” having a much larger portion of *food* related queries.

Thus, the developer follows the suggestion, and traverses up from \$can. Doing so triggers the neighboring template panel (Fig. 2D) to present sibling templates with different \$VERBs. He notices that the dataset contains a large number of queries in the form of “how \$do \$-PRON- \$VERB \$NOUN”. By expanding the bar charts for the ground truth labels (Fig. 9), we see an interesting **label distribution difference** between the two templates: the queries under the \$can template are labeled as *health*, *none*, and *procedural* while those under \$do are mainly *food* related. The developer notes that queries in these two forms should be similar, and proposes to augment the data (i.e., to create another batch of food questions by replacing “do” with “can”).

Despite the distribution differences between “\$can” and “\$do”, the developer deems these templates identical, and moves to inspect “how \$VERB \$-PRON- \$VERB \$NOUN”. Afterwards, he instead traverses down the second \$VERB to understand what are the actions being queried. Fig. 10 shows a neighbor template table for the traversal. Among the four child templates covering most queries, “how \$VERB \$-PRON- \$cook \$NOUN” has the most indicative verb, with all queries labeled and predicted to be *food* related. Meanwhile, “how \$VERB \$-PRON- \$do \$NOUN” has high error rate with only one predicted label (*none*). Retrieving its queries, the developer notices that all but one are labeled as *procedural*. He suspects the model neither understands the template, nor recognizes the \$NOUNs (“division”, “a screen recording”, etc.) This **model error pattern** suggests the model has underfitted to this form of query. The lemma \$snake has the highest coverage. The model predicts most of the “\$make” queries to be *food*. However, 6 of 33 queries are labeled as *procedural* despite involving alcoholic beverages (“how do you make vodka”, “how do you make a margarita”). In hindsight, these queries could arguably have both labels. Furthermore, in \$play, we see a **model error outlier**: with “chess” and “old maid” both being board games, “how do you play chess” is labeled as *procedural* but predicted as *none*, whereas “old maid” shows the reverse: labeled as *none* but predicted as *procedural*. Our developer notes the conflict and decides to double check the labels, as well as the model stability on various similar \$NOUNs.

Hoping to see a comprehensive list of procedural queries, the developer samples more queries with the *procedural* la-

















	Coverage	Error rate	Label	Predict
\$make	34 	21% 	3 	3 
\$cook	10 	0% 	1 	1 
\$play	6 	50% 	2 	2 
\$do	5 	80% 	2 	1 

Figure 10. Neighbor template table showing the top four LEMMA (\$make, \$cook, \$play, and \$do) after splitting the second \$VERB in “how \$VERB \$-PRON- \$VERB \$NOUN”.

bel. He notices many queries asking “how to do (something).” He generalizes from those queries to a template, “how to \$VERB \$NOUN”. He finds that this template (with 35 queries) has a higher portion of food queries (85.7%), but most queries are semantically similar to those in “how \$VERB \$-PRON- \$VERB \$NOUN”. He creates a single, regular expression-related template capturing both subgroups: “how (\$VERB \$-PRON-|to)+ \$VERB \$NOUN”, so he can revisit these queries in future investigation.

USER STUDY

We ran a user study to answer the following three questions:

- Q1 Can Tempura support data exploration and error analysis, and help participants make actionable observations easily?
- Q2 How do analysts decide which templates to inspect?
- Q3 Are Tempura’s components useful for exploration?

Our user study lasted for one hour. We first surveyed participants’ query analysis experience. Next, we provided a tutorial outlining the features of Tempura. Participants then used Tempura to explore the same data and model as described in *Usage Scenario*. We encouraged participants to think aloud and describe their observations while they explored. We noted down their observations throughout the session (Q1), and also logged their clickstreams for analyzing their exploration process (Q2). Afterwards, we confirmed the recorded observations with the participants, and they rated each observation by the ease of discovery, relative to their prior experience working with queries in the wild. Participants further self-assessed how much more they learned about the dataset and the model, and rated the usefulness of different components in Tempura (Q3). All the ratings were collected in the form of five-point Likert scale [24]. Eight ML developers at Apple participated in our study, all with prior query analysis and modeling experience (worked on 3-10 datasets, $\mu = 6.2$). Below, we answer the questions with the study outputs.

Q1: Users made actionable observations in Tempura.

Observations are on datasets and errors.

As mentioned in *Usage Scenario*, participants noticed both general patterns and outlier behaviors in dataset and error analysis. Their observations can be divided into four categories, as in Table 3. On average, subjects reported 5.4 ± 1.83 observations.⁸ In total, participants made 26 unique observations. Participants made around the same number of observations

⁸We counted participants’ self-reported observations. Recall that we confirmed and asked them to rate each one of their observations.

on datasets and on models, indicating Tempura can support both tasks. They consistently rated that it was easier to make observations in Tempura (4.5 ± 0.73).

Observations are actionable.

Five out of eight participants noted that their observations were *actionable*, and proposed several potential next steps. For example, templates helped them improve their labeling tasks. With templates covering interchangeably labeled queries (data noise issue), they would not only clean up existing labels, but use example queries in those template groups to revise the labeling instructions. Templates’ label distributions could also guide training data revision. Participants would overcome the template overfitting issue (i.e., queries covered by one template are all predicted the same, regardless of their labels) by collecting data with the same syntax but different label. In contrast, for underfitting templates (i.e., the correlation between a template and a label is not correctly learned), participants would augment the data with semantically neighboring templates. Templates could also help build targeted and challenging validation sets for testing specific syntax phenomena.

Tempura could be better at error analysis.

Participants agreed that they *understood more* about both the dataset (4.4 ± 0.45) and the model (3.9 ± 0.90). Generally, participants believed their analysis on data distribution was more thorough, because various factors that Tempura does not consider — model architecture, transfer learning effect — can all account for an imperfect model. They offered suggestions on improving Tempura for error analysis. For example, to prioritize *problematic* templates, we could select overview templates using entropies on model related measurements (e.g., error rates). Such measurement can further surface templates related to error analysis, but we worry it would focus too much on a specific model, at the cost of general dataset information. One participant also mentioned that adding word frequencies could help understand whether the model is overfitting to particular templates or keywords. In fact, structural templates can be paired with many conventional attributes (query length, sentiment, etc.) Further investments are needed to understand which additional debugging attribute provides the most comprehensive view.

Q2: Attributes as the primary clue.

We retrieved participants’ clickstreams affecting the overview table, and programmatically labeled them into four types: finding templates that (1) have a certain attributes pattern like high error rate or coverage (*Attr*), (2) contain queries with specific ground truth or prediction label(s) (*Label*), (3) satisfy a partial template search filter, in the form of regular expression pattern (*Regex*), and (4) are generated from a specific query (*Query*).

The results revealed that participants surfaced templates they would like to inspect via all strategies, but the most common ones were by sorting or filtering *Attrs* (used by all participants), or particular *Labels* (seven participants). Users’ free form responses reflected that they highly valued the coverage and error rate. Three participants (P3, P6, P8) also commented that the label distributions were useful. P6 said he actively searched for structural overfitting (templates with only one prediction label). P8 expressed particular interest in templates with more

Observation Type	Count	Examples
Dataset distribution	7 (27%)	“how long \$do \$-PRON- \$VERB \$NOUN (for)*” (148 queries) are all <i>food</i> related (e.g., “how long do you boil corn”), whereas “how long \$do \$-PRON- take for \$NOUN to \$VERB” (16 queries) has half of <i>health</i> queries (e.g., “how long does it take for a piercing to heal”).
Dataset noise	7 (27%)	2 out of 86 queries in “can dog (have eat)+ \$NOUN” are labeled as <i>health</i> , but the \$NOUNs are all <i>food</i> (“corns”, “rice”). All the 9 queries in “what \$be the benefit of \$NOUN” are predicted to be <i>health</i> , but 4 are labeled as <i>none</i> .
Model error pattern	10 (38%)	The model cannot understand rare tokens. The \$NOUNs in “what \$be \$NOUN used for” (57 queries) are all medical related, but the model only predicts 28 (49.1%) to be <i>health</i> — possibly affected by their training frequencies. Overfitting to templates. “is \$NOUN bad for \$-PRON-” (5 queries) are all predicted to be <i>health</i> related, even for “is college bad for you” (supposed to be <i>none</i>).
Model error outlier	2 (8%)	In “can \$-PRON- \$VERB \$NOUN” (29 queries), similar queries are predicted differently: “can I block a contact” is predicted as <i>none</i> , whereas “Can I block unwanted phone calls” is <i>procedural</i> .

Table 3. Participants’ example observations on (1) Dataset distribution (they observed frequent query patterns); (2) Dataset noise (they discovered labeling issues or illegitimate queries); (3) Model error pattern (they concluded systematic error categories); (4) Model error outlier (they noticed model’s specific strange behaviors that are not generalizable, or that related queries are mostly correctly predicted).

varying labels: “Sometimes we locally improve models for queries with certain patterns. This helps me understand, for example, if the data augmentation on a specific label has negative effects on queries with the same syntax but different labels.” Only four participants generalized template structures from a Query. P7 explained that generalization required more of a mental model on the template-query mapping, and thus was harder than starting from template overviews.

Q3: Tempura is effective; Users learned more on dataset.

When assessing the usefulness of different components, users rated the algorithm selected overview templates (4.2 ± 0.63), cross-filtering between templates and queries (4.4 ± 0.68), and supportive attributes (e.g., error rate) & interactions (e.g., sorting) (4.7 ± 0.41) as very useful, and rated template traversal (3.6 ± 0.83) as potentially useful. They especially liked the combination of templates and attribute summaries, saying it greatly helped redirect their attentions to important query groups. The lower rating for traversal could be due to the size of the dataset. As P4 pointed out, “With 10k queries, interesting hierarchical groups are less common. I found myself sometimes getting many low coverage child templates.”

Participants all thought Tempura was intuitive to learn, and at the same time it greatly enhanced their query exploration experience. Five users commented that Tempura offered a finer-grained and systematic analysis process, making their observations more precise. They mentioned that structural templates provided more efficient and intuitive starting points for categorizing raw queries, helped them to bypass the large coding overhead needed, and sped up their otherwise tedious manual process. The speed up was significant. As P1 described: “I learned 7 new things in the last half an hour! Without the tool, I would spend all those time writing Python scripts without knowing if it will lead to anything significant.”

RELATED WORK

Query Text Analysis

Seeking to reveal systematic and actionable patterns in a dataset, existing analysis methods usually slice text queries into groups-of-interest in various ways. However, grouping unstructured queries is challenging. Groups created manually or via grouping scripts [41, 32] are usually not representative of the dataset distribution. Filters on query properties can scale (e.g., word frequency [9], query length or answer type [39, 21]), but prior work has noted that such features usually could

either overlook the context of the whole sentence [27, 34], or get too abstractive to be precise and interpretable [6].

Meanwhile, without isolating features, researchers have also tried to organize the dataset by automatically classifying [36, 3] or clustering [1] similar queries together. To date, such algorithms measure similarities with TF-IDF [1] or embedding-based [40] distances, which usually result in query groups with mixed semantic (“when was the colored TV invented” v.s. “when did we invent the colored TV”) and structural similarity (“when was the car invented”). Though mixed groups are noisy, both semantic- and structure- based grouping support unique analysis tasks in isolate: Semantically similar queries can help identify paraphrases, reveal specific contents being queried, or evaluate model robustness on semantics-preserving perturbations [33]. Meanwhile, structurally similar ones identify queries with common intents (e.g., queries under “is \$NOUN good for me” are all associated with health related issues), and can serve as the basis for query dataset augmentation [10] (e.g., augment \$NOUN with additional medicine names in “is \$NOUN good for me”, or rewrite all the covered queries to “what does \$NOUN do to me”). Our work prioritizes structural similarity, as this features could be more explicitly measured, and thus are more interpretable.

Template/Pattern-based Analysis

Structural templates have been implicitly used for various tasks. In question answering [39, 37], analysts inspect queries based on their question types (“what”, “who”), which could be viewed as structures of a short phrase in a query. However, these “templates” are usually too shallow, covering up to three words in their lemma forms. In a more explicit manner, rule/pattern-based methods have been extensively used in information extraction [16, 25, 29]. Hearst [15] identified a set of lexico-syntactic patterns (again, structures of part of the sentence) to recognize hyponyms from free-form text. This approach helps ground various studies on extracting semantic phrases (drug entities from online medical form [12], product features from reviews [30], etc.) Later, Ratner et al. [32] recognized pattern-based heuristics as one of the primary source for writing labeling functions in their data programming system, Snorkel. In Errudite [41], Wu et al. similarly allowed linguistic pattern query for grouping instances or doing counterfactual analysis. Mohasseb et al. [27] enumerated typical syntactic structures for three different types of web queries, and used such structures as features for query classification. However,

these studies require manual compilation of templates, which is tedious and prone to human errors and biases. In contrast, Tempura automatically mines templates from a given dataset. If used as a basis for Snorkel or Errudite, it could compensate the potential biases in analysts’ prior knowledge.

Prior work has also explored automatic template generation. Li [23] structured queries’ noun phrases by identifying intent heads (primary objects) and their associated attributes. His templates capture semantic aspects, and tend to include more concrete ontology definitions. While our templates rely on standard POS tags and named entities by default, the preprocessing step can be easily customized and embed more sophisticated linguistic abstractions (e.g., ontologies from knowledge graphs [8]). Hu et al. [19] proposed to automatically abstract frequent sentence patterns from social media posts. Similarly, SENPAI [34] mined patterns for social computing related measurements — credibility, politeness, and sentiment. Both have slightly different definitions than ours, as they focus on raw token-based templates or frequent subspans of the sentences. On the contrary, Tempura takes advantage of linguistic features, and helps answer more targeted analysis questions from multiple aspects, at various granularities.

DISCUSSIONS

We contribute Tempura, an interactive tool that lets analysts explore a query dataset with structural templates. To help analysts navigate the template space, Tempura uses a traversal measure that suggests related templates likely to yield further explorations. To provide an overview, we present a weighted set cover algorithm to select a template subset that represents the dataset’s query distribution. We show that the generated overviews can expose distributional differences between industrial and academic datasets, with the former being more sparse. Our user study shows that Tempura lets developers use meaningful query groups to investigate modeling issues and improve their models. As the improved models get deployed, we believe Tempura can help enhance the end-user experience.

Implications beyond Data Exploration and Error Analysis

Our work has broader implications beyond direct use cases.

First, as a framework, the structural template helps us engage with non-technical ML stakeholders. For example, to enhance model evaluation, *quality analysts* and *designers* can build template-based diagnostic sets to address model biases (e.g., requesting uniform predictions on “\$-PRON- is a \$NOUN”, with \$-PRON-=[“she”, “he”], and \$NOUN=[“doctor”, “nurse”].) To de-noise the data, *dataset requesters* can implement quality controls in the crowd-sourced labeling process, and flag queries that are labeled inconsistently from structurally similar ones.

Second, our analysis results encourages future explorations on data understanding and wrangling. The distributional differences between datasets in our case study encourage researchers to design dataset comparison tools, so to help developers assess whether pre-trained models are suitable for a seemingly similar domains. On the other hand, half of the errors in our user study are data related, emphasizing the importance of data qualities. More in-depth studies can be conducted to

explore the impact of various data wrangling techniques on model improvement. For example, rectifying distribution gaps between neighboring templates with data augmentation might be effective for fixing annotation artifacts [13] (e.g., “how \$do/\$can \$-PRON- \$VERB \$NOUN”).

Limitations and Future Extensions

We discuss the limitations introduced by our design decisions and their corresponding future enhancements.

Assume queries are well-structured. Tempura is currently implemented and tested for well-structured queries. However, the syntactic patterns of ill-structured or incomplete queries (e.g., “\$PERSON net worth”, “weather \$CITY”) can still be quite useful to users. In fact, with these queries being shorter and more to the point, we can potentially generate templates with higher coverage. We believe Tempura can handle these queries if we switch to more advanced taggers (e.g., Ganchev et al. [11] reported 94% tagging accuracy on real-world search logs). On the other hand, complete but long queries could generate templates too sparse to explore. One possible solution is to automatically mine a compact set of partial templates by omitting insignificant structures from queries. The insignificance can be defined by either statistics (similar to frequent pattern mining [14]), or the parsing tree structure (trimming subclauses on a tree, removing stopwords.)

Prioritize syntactic structures over semantics. As mentioned in *Related Work*, Tempura primarily focuses on the unique benefits of syntactic similarities. While syntactic structures can capture semantics to some extent, one important future direction is to incorporate *more* semantic understandings. The most straightforward method is to enable more semantically meaningful annotations. For example, with knowledge graphs, tokens like “apple” in [35] can have word sense labels (*fruit* or *company*). Using embedding space similarities, synonyms like “deadliest” and “fatalities” [18] can be grouped beyond \$NOUN. Beyond token-level semantics, we can also enhance sentence-level grouping by merging structural paraphrases into larger semantic groups. Tempura currently achieves such merging via manually created, regular-expression based templates, but more advanced paraphrasing detection models [42] can further automate it.

Select template overview with a pre-defined objective.

While our traversal score and selection algorithm enable the overview+details user experience, depending on analysts’ objective, alternative methods could be more effective. Like mentioned in *User Study*, entropy scores on error rate could help surface structures that a model perform poorly on. While *stable* templates save analysts’ time, doing the reverse and selecting templates with informative traversal options can expose more interactive exploration options. Future work exploring the algorithmic space is needed, such that Tempura can cope with analysts’ different primary objectives.

ACKNOWLEDGMENTS

We gratefully thank Carlos Guestrin, Charlie Maalouf, and Silviana Ciurea-Illcus for their helpful comments. We also appreciate the valuable input from our study participants, and the constructive comments from the anonymous reviewers.

REFERENCES

- [1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. 2017. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919* (2017).
- [2] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*. 2425–2433.
- [3] Jaime Arguello, Fernando Diaz, Jamie Callan, and Jean-Francois Crespo. 2009. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 315–322.
- [4] Jay Budzik and Kristian J Hammond. 2000. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 5th international conference on intelligent user interfaces*. ACM, 44–51.
- [5] Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. 2011. Efficient diversification of web search results. *Proceedings of the VLDB Endowment* 4, 7 (2011), 451–459.
- [6] Jason Chuang, Margaret E Roberts, Brandon M Stewart, Rebecca Weiss, Dustin Tingley, Justin Grimmer, and Jeffrey Heer. 2015. TopicCheck: Interactive alignment for assessing topic model stability. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 175–184.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [8] Lisa Ehrlinger and Wolfram Wöb. 2016. Towards a Definition of Knowledge Graphs. *SEMANTiCS (Posters, Demos, SuCCESS)* 48 (2016).
- [9] Cristian Felix, Anshul Vikram Pandey, and Enrico Bertini. 2017. TextTile: an interactive visualization tool for seamless exploratory analysis of structured data and unstructured text. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 161–170.
- [10] Adam Fourney and Susan T Dumais. 2016. Automatic identification and contextual reformulation of implicit system-related queries. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 761–764.
- [11] Kuzman Ganchev, Keith Hall, Ryan McDonald, and Slav Petrov. 2012. Using search-logs to improve query tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, 238–242.
- [12] Sonal Gupta, Diana L MacLean, Jeffrey Heer, and Christopher D Manning. 2014. Induced lexico-syntactic patterns improve information extraction from online medical forums. *Journal of the American Medical Informatics Association* 21, 5 (2014), 902–909.
- [13] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation Artifacts in Natural Language Inference Data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 107–112. DOI:<http://dx.doi.org/10.18653/v1/N18-2017>
- [14] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery* 15, 1 (2007), 55–86.
- [15] Marti A Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 539–545.
- [16] Raphael Hoffmann, Luke Zettlemoyer, and Daniel S Weld. 2015. Extreme extraction: Only one hour per relation. *arXiv preprint arXiv:1506.06418* (2015).
- [17] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 600.
- [18] Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. 2017. Applying pragmatics principles for interaction with visual analytics. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 309–318.
- [19] Mengdie Hu, Krist Wongsuphasawat, and John Stasko. 2017. Visualizing social media content with sententree. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 621–630.
- [20] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* (2017).
- [21] Kushal Kafle and Christopher Kanan. 2017. An analysis of visual question answering algorithms. In *Proceedings of the IEEE International Conference on Computer Vision*. 1965–1973.
- [22] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics* 7 (2019), 453–466.

- [23] Xiao Li. 2010. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 1337–1345.
- [24] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).
- [25] Andrew McCallum. 2005. Information extraction: Distilling structured data from unstructured text. *Queue* 3, 9 (2005), 4.
- [26] Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. 2018. Efficient and robust question answering from minimal context over documents. *arXiv preprint arXiv:1805.08092* (2018).
- [27] Alaa Mohasseb, Mohamed Bader-El-Den, and Mihaela Cocea. 2018. Analysis of the syntactical structure of web queries. In *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, Vol. 2. IEEE, 557–562.
- [28] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human-Generated MACHine Reading COMprehension Dataset. (2016).
- [29] Marius Pasca and Benjamin Van Durme. 2007. What You Seek Is What You Get: Extraction of Class Attributes from Query Logs.. In *IJCAI*, Vol. 7. 2832–2837.
- [30] Ana-Maria Popescu and Oren Etzioni. 2007. Extracting product features and opinions from reviews. In *Natural language processing and text mining*. Springer, 9–28.
- [31] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).
- [32] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.
- [33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 856–865.
- [34] Mattia Samory and Tanushree Mitra. 2019. SENPAI: Supporting Exploratory Text Analysis through Semantic & Syntactic Pattern Inspection. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 13. 452–462.
- [35] Vidya Setlur, Melanie Tory, and Alex Djalali. 2019. Inferencing underspecified natural language utterances in visual analysis.. In *IUI*. 40–51.
- [36] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. 2006. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 131–138.
- [37] Yelong Shen, Xiaodong Liu, Kevin Duh, and Jianfeng Gao. 2017. An empirical analysis of multiple-turn reasoning strategies in reading comprehension tasks. *arXiv preprint arXiv:1711.03230* (2017).
- [38] Ben Shneiderman. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*. IEEE, 336–343.
- [39] Soumya Wadhwa, Khyathi Raghavi Chandu, and Eric Nyberg. 2018. Comparative Analysis of Neural QA models on SQuAD. *arXiv preprint arXiv:1806.06972* (2018).
- [40] Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. 2015. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 352–357.
- [41] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2019. Errudite: Scalable, Reproducible, and Testable Error Analysis. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*. 747–763.
- [42] Runqi Yang, Jianhai Zhang, Xing Gao, Feng Ji, and Haiqing Chen. 2019. Simple and Effective Text Matching with Richer Alignment Features. In *Association for Computational Linguistics (ACL)*.
- [43] Neal E Young. 2008. Greedy Set-Cover Algorithms: 1974–1979; Chvátal, Johnson, Lovász, Stein. *Encyclopedia of algorithms* (2008), 379–381.
- [44] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 364–373.