Discover Computing

Research

Boolean interpretation, matching, and ranking of natural language queries in product selection systems

Matthew Moulton[1] · **Yiu-Kai Ng**[1]

Received: 8 February 2023 / Accepted: 18 March 2024
Published online: 03 April 2024

Abstract
E-commerce is a massive sector in the US economy, generating $767.7 billion in revenue in 2021. E-commerce sites maximize their revenue by helping customers find, examine, and purchase products. To help users easily find the most relevant products in the database for their individual needs, e-commerce sites are equipped with a product retrieval system. Many of these modern retrieval systems parse user-specified constraints or keywords embedded in a simple natural language query, which is generally easier and faster for the customer to specify their needs than navigating a product specification form, and does not require the seller to design or develop such a form. These natural language product retrieval systems, however, suffer from *low* relevance in retrieved products, especially for *complex* constraints specified on products. The reduced accuracy is in part due to under-utilizing the rich semantics of natural language, specifically queries that include Boolean operators, and lacking of the ranking on partially-matched relevant results that could be of interest to the customers. This undesirable effect costs e-commerce vendors to lose sales on their merchandise. In solving this problem, we propose a novel product retrieval system, called *QuePR*, that parses arbitrarily simple and complex natural language queries with(out) Boolean operators, utilizes combinatorial numeric and content-based matching to extract relevant products from a database, and ranks retrieved resultant products by relevance before presenting them to the end-user. The advantages of *QuePR* are its ability to process explicit and implicit Boolean operators in queries, handle natural language queries using similarity measures on partially-matched records, and perform best guess or match on ambiguous or incomplete queries. *QuePR* is unique, easy to use, and scalable to all product categories. To verify the accuracy of *QuePR* in retrieving relevant products on different product domains, we have conducted different performance analyses and compared *QuePR* with other ranking and retrieval systems. The empirical results verify that *QuePR* outperforms others while maintaining an optimal runtime speed.

Keywords Natural language query · Interpretation · Matching · Ranking · Data retrieval

This work is based on an earlier work, "Retrieving and Ranking Relevant Products from Boolean Language Queries," appeared in Proceedings of the 2022 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'22), a short paper. The short paper has been thoroughly revised with significant enhancement to the overall system architecture, detailed justification on the design methodology of each major component of the system, and fine-tune empirical study.

✉ Yiu-Kai Ng, ng@compsci.byu.edu; Matthew Moulton, moulton.matthewj@gmail.com | [1]Computer Science Department, Brigham Young University, 3361 TMCB, Provo, UT 84602, USA.

1 Introduction

Customers with the intention to buy products conduct search that will fulfill their specific needs and wants. With the popularity of Internet, the search is made much more convenient for the average customers who choose to shop whenever and wherever they like and are given a broad variety of products on a number of retailer's websites. For example, approximately 75 million distinct products were offered on Amazon in March 2021 [1]. It is well-known that online stores offer greater variety for two different reasons. First, online stores do not have the same restrictions on space as brick and mortar stores and have essentially unrestricted display room. Second, a wider variety is available, since the customer is not limited to a single fixed location. For product variety to be useful, consumers need to be able to find what they are looking for. However, as the number of products sold online increases, finding the right ones becomes more and more difficult. With about 256 million digital customers in the Unites States alone, this problem is common to many people [2]. Furthermore, the number of digital shoppers is expected to grow to 292 million by 2025 [2], making the problem even more endemic.

As users look for products, they often relay their needs and requirements to product retrieval systems, such as the ones offered by Amazon, Walmart, and Apple, which help them locate the best matching products. Some of these product retrieval systems rely on the users entering the desired items in strict forms or based on keyword search [3]. This search strategies impose unnecessary complexity on the user, who must become familiar with the options available, formulate their desires as inputs the system expects, and take time to navigate in order to enter these inputs. In fact, it is more convenient for the user to simply specify the required product in a natural language query. Users can formulate their needs in terms of their language, which of course imposes the burden of the product retrieval system to parse and interpret the desired products and fetch results. Current natural language product retrieval systems exists for languages such as English [4], Japanese [5], and Finnish [6], but lack the accuracy of form-based systems [7, 8] that restrict the search constraints.

The two primary challenges that these product retrieval systems face are the inherent complexity of natural language and ambiguity in the search query. To enhance the effectiveness and efficiency of processing natural language queries on products, we propose a new system, called the *Query Product Retriever* (*QuePR* for short), which addresses these challenges. On a high level, *QuePR* parses the natural language query for product constraints, searches the underlying database (DB) for exact and partial matches, and sorts the results by relevance to be returned to the user. There are three notable contributions of *QuePR* in extracting relevant products to satisfy the needs of users: (i) a novel approach for interpreting and evaluating an arbitrary number of recursively nested explicit and implicit Boolean operators is presented, (ii) a unique strategy of relevancy scoring is introduced with an accurate notion of similarity both for categorical and numeric data based on the constraint specification detected by our typification approach on data attributes, and (iii) a simple and elegant method in computing the relevancy scores for individual constraints for scaling the product attribute types in calculation of the final product score is employed.

To enhance the effectiveness and efficiency of processing user queries on products, *QuePR* (i) first uses the *Multinomial Naive Bayes classifier* to automatically identify the domain of a query on products *Q*, (ii) corrects spelling mistakes using the *noisy channel model* when processing *Q*, (iii) performs an intelligent "best guess" and retrieves answers that most adequately match an incomplete and/or ambiguous selection criterion specified in *Q*, (iv) applies substring matching to speed up the process of retrieving answers to *Q*, and (v) processes each SQL query converted from *Q*.

The effectiveness and merits of *QuePR* have been experimentally verified in terms of achieving high accuracy of interpretation for simple and complex queries. Furthermore, it has been validated that *QuePR* retrieves more relevant results to the user than other contemporary product retrieval systems. This accuracy manages to save customers' time and efforts in product searching and offer them all the relevant and related products available, which leads to higher degree of satisfaction and greater security at the time of purchase.

2 Related works

To answer natural language queries and retrieve the records in DB to answer the queries, *QuePR* applies the hybrid data extraction approach by integrating the information retrieval and data retrieval strategies. Both of these retrieval strategies are popular areas of study. As such, there are a number of notable contributions to both areas [9–12].

There is no luck of publications in the literature that propose procedures to handle natural language queries and transform them into another form for processing. Query rewriting, especially of *anaphora* and *ellipses*, has proven to be effective, especially when used in conjunction with embedded context of previous queries [13]. Other systems opt for a simpler keyword extraction from the natural language query [14, 15], which is simpler, but may not yield comprehensive results. One solution that has been tried is to blend the concepts of rewriting and keyword extraction, i.e., important keywords can be extracted from the original query, then other related keywords may be generated in order to focus the search [16]. Besides identifying significant keywords in a natural language query, *QuePR* applies typification and standarization, two sophisticated approaches, to recognize important terms specified in a query.

It often occurs that natural language queries are ambiguous, and there is no single interpretation that can be generated with confidence. Researchers have attempted to create systems that collaborate with the user to develop an accurate interpretation of the query intent. Li and Jagadish [17] present the query users with several possible interpretations for one to be chosen from, whereas Kum et al. [18] adopt the user-feedback strategy, asking the user to indicate the intent of the targeted questions. These approaches are similar to conversational question-answer systems [19], which save the context of user queries and responses. However, these strategies require user's relevant feedback and could be troublesome to the users.

In different natural language query processing systems, after information has been retrieved by each system, relevant data is expected to be presented to the user. Among these systems, the deep neural network model has proven to be successful, such as the one retrieves data from cells of web tables [20]. If the domain is formatted as a database, SQL queries or similar query languages may be constructed to retrieve the relevant data [17, 21]. Executing these retrieval queries yield accuracy results only if the specification of the search constraints are precise. In systems over single domain, the accuracy design goal could be easier to achieve [22]. However, it is an irresistible challenge to deal with multiple domains, especially domains that are similar in nature.

In a product retrieval system, an extension of the concept of *ranking* is to find related products. Information can be "ranked" by similarity in the source, and based on that similarity, results are presented in a ranked order. Ontological concept hierarchies [23, 24] can be constructed to determine relevance of arbitrary terms. *QuePR*, on the other hand, applies the concept of similarity extensively in ranking the attribute values of partially-matched products.

With very few publicly available datasets to appropriately train systems that handle natural language queries, efforts have been made to resolve this issue. In 2021, Papenmeier et al. [25] released an e-commerce dataset of natural language descriptions of jackets and laptops, which is a good step in the right direction. The descriptions, however, were written as descriptions of desired features rather than queries. Moreover, the products were limited exclusively to jackets and laptops only.

3 Query product retriever (*QuePR*)

Users prefer to express search queries in natural language, since expressing natural language queries is easier for the user and does not require them to learn a service-specific selection language, such as SQL, the complexity of which is directly related to its syntax and semantics. For this reason, we have developed *QuePR*, a natural language querying system for product retrieval. Figure 1 depicts the overall process of *QuePR* in retrieving DB records.

3.1 Classification of product domains

Prior to answering a user query *Q*, *QuePR* first determines the category to which *Q* belongs. *QuePR* adopts the *Multinomial Naïve Bayes classifier* (MNB) [26] to classify *Q* into one of the available domains. MNB, a supervised learning approach, is a probabilistic model widely used in natural language processing [27]. The model is applicable in calculating the probability that a given query *Q* is in a particular category based on the probability value of each word in *Q* being in the category, and the category with the highest probability is given as the output. Equation 1 computes the multinomial probability of *Q* given a particular category *C*. For purposes of testing, *QuePR* was verified by using six different categories: automobiles, furniture, housing, jewelry, jobs, and motorcycles as running examples. MNB, a widely-used supervised-learning classifier, is capable of handing any number of categories [26].

$$P(Q \mid C) = \Pi_{w \in V} P(w \mid C)^{tf_{w,Q}} \tag{1}$$

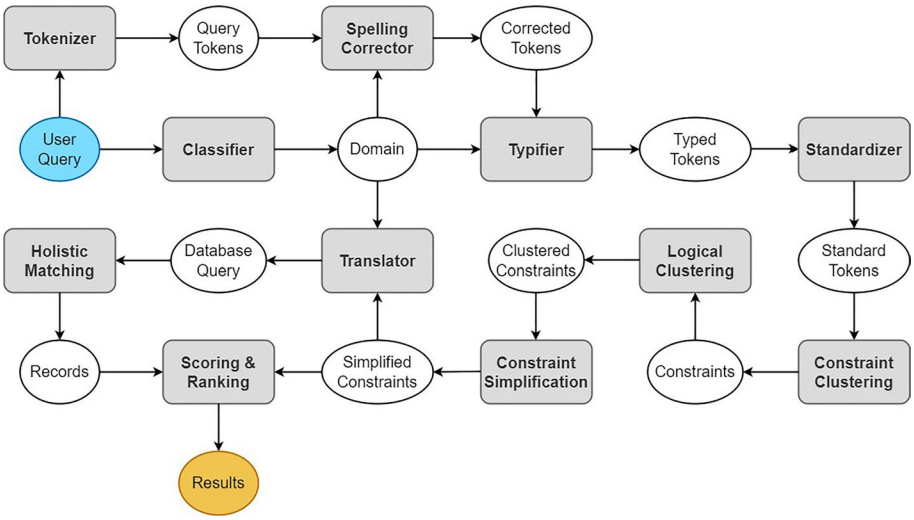

Fig. 1 The process of matching natural language queries with DB records

where V is the set of words in the training dataset, $tf_{w,Q}$ is the count of the word w in Q, and $P(w \mid C)$ is the probability that w appears in C. In case a word in Q never appears in the training data, we adopt the *Laplacian smoothing strategy* [28] to assign $P(w \mid C)$ to a non-zero probability as

$$P(w \mid C) = \frac{tf_{w,Q} + 1}{\mid Q \mid + \mid V \mid} \tag{2}$$

where $\mid Q \mid$ is the number of words in Q, and $\mid V \mid$ is the number of distinct words in the training set C.

For example, given the natural language query SQ, "*hond ared accord or silver civic 2008 less than $6K in NY*", SQ belongs, and should be assigned to the *automobiles* category,[1]

3.2 Tokenization

Once a query has been classified into one of the categories, it is divided into its component *tokens*. Each token is a semantically indivisible combination of symbols, which may include alphabetic characters, digits, and punctuation. Tokens are typically divided by space delimiters, but may be divided further between words and punctuation, numbers and punctuation, or unit symbols (e.g., $ and £) and digits. To extract tokens from an input user query, *QuePR* employs the Natural Language Toolkit's word tokenizer [29], with only an alteration on hyphens.[2] *QuePR* splits tokens which contain a hyphen if all the characters on both sides are exclusively numeric symbols. Thus, the example "10–15" would be converted into three tokens: "10", "-", and "15". For example, SQ, given above, is tokenized into SQ_t: "*hond, ared, accord, or, silver, civic, 2008, less, than, $, 6K, in, NY*".

3.3 Spelling correction

It is common for users to make spelling mistakes when entering natural language queries. It is also very common for users to have abbreviations in natural language queries to save them time typing whole words. However, these spelling mistakes and abbreviations can severely decrease the ability of *QuePR* to properly *identify constraints* and *match* on requested *products*. As such, two subprocesses, *abbreviation expansion* and *spelling correction* transform the query tokens into a more meaningful form.

The first step of spelling correction is abbreviation expansion, where abbreviation tokens are transformed into their more verbose forms extracted from a database of expanded abbreviations. One common form of abbreviations is the two letter *state* acronyms (such as NY and CA), which indicate the location of the product being sold. In the sample

[1] Note that SQ contains an intentional typo *hond ared* which should read *honda red*.

[2] By the typical behavior, hyphens are not used as a delimiter, yet for product searches, hyphens can sometimes be used to indicate ranges of numbers (such as "$10–15").

Table 1 Attribute types and sample values of various types in a product DB

Attr. type	Product description	Automobile values	Housing values	Jewelry values
I	Core features	Honda, Accord	Single family	Nicklace
II	Customization	4 Cylinder	NY County	Blue
III	Quantification	35K, 2015	4 Bedrooms	20K

query, "NY" is identified as the abbreviation for the state of New York. Another common abbreviation is the shorthand "K", which stands for 1000 in a numeric value. The abbreviation expansion on SQ_t yields SQ_a: "*hond, ared, accord, or, silver, civic, 2008, less, than, $, 6000, in, New York*".

The *noisy channel model (NCM)* [30] for spelling correction is a general framework that can address the issues of context and run-on errors. The intuition of *NCM* is that a person chooses a word w to write, based on the probability distribution $P(w)$, i.e., the *language model probability*. The person then tries to write w, but the noise channel (presumably the person's brain) causes the person to write the (misspelled) word e instead, with the *error model probability* $P(e \mid w)$ [30]. To estimate the probability that given an error written word e, the correct word is w, i.e., $P(w \mid e)$, *NCM* computes the *frequency* of different *types of errors* using a product test dataset (see 4.1 on test data for details) as

$$\begin{aligned} P(w \mid e) &\overset{rank}{=} P(e \mid w)P(w), \\ P(w) &= \lambda P(w) + (1 - \lambda)P(w \mid wp) \end{aligned} \tag{3}$$

where wp is a previous word of w, and λ is a parameter, which specifies the relative importance of $P(w)$ and $P(w \mid wp)$, and is experimentally set to be 0.5.

Performing the error checking on SQ_a (based on NCM) yields SQ_e: "*honda, red, accord, or, silver, civic, 2008, less, than, $, 6000, in, New York*".

3.4 Typification

Although there are many different ways for users to express constraints in a natural language query, some broad patterns emerge based on which *attribute* of a desired product the user is constraining. For example, some attributes, such as product cost, product age, etc., are inherently numeric, and thus a constraint on them may include an *upper* or *lower bound*. We categorize each product attribute into one of the three types: Type I, Type II, or Type III, which allows *QuePR* to abstract away in parsing which product attribute is being constrained and instead focus on how the constraint is likely to be seen.

- *Attributes of Type I describe core identity features of a product*. They are commonly the most important feature of the product. Each value of a Type I attribute is always *mutually exclusive* with the other values of that same attribute. As such, if two or more values, which correspond to a single Type I attribute, appear in proximity to each other in a user query, we assume that the user implies a disjunctive constraint with those attributes.
- *Attributes of Type II describe non-numeric ancillary features about the product*. These attributes can describe the aesthetic, location, customization, or other features of the product. Type II values may be mutually exclusive, but are not constrained to be.
- *Attributes of Type III describe a numeric status of the product*. When users describe a constraint on the value of a Type III attribute, it may contain a numeric value, a relative bound, and a unit of number. For example, "< $5000" contains "<" as the bound, "5000" as the numeric value, and "$" as the unit.

QuePR classifies each token in the user query as at most one of the three product attribute types. In some cases, such as prepositions or articles, a token does not describe any attribute, whereas all the Boolean operators (e.g., *and* and *or*) and comparison operators (e.g., less/greater than) are essential in selecting products. For the sake of uniformity, these tokens are also assigned a type, called Type IV. As an example of typification, SQ_e is typified as SQ_y as: "$honda_1$, red_2, $accord_1$, or_4, $silver_2$, $civic_1$, 2008_3, $less_4$, $than_4$, $\$_3$, 6000_3, in_4, $New\ York_2$". Table 1 shows the different attribute types, i.e., Type I–III, of constraints applicable to different attributes specified in a user query that can be handled by *QuePR*.

Identifying the type of each token is a difficult task because there are no token-level abstractions which correspond directly to the type. Utilities which identify the part of speech of each token in a sentence are available (such

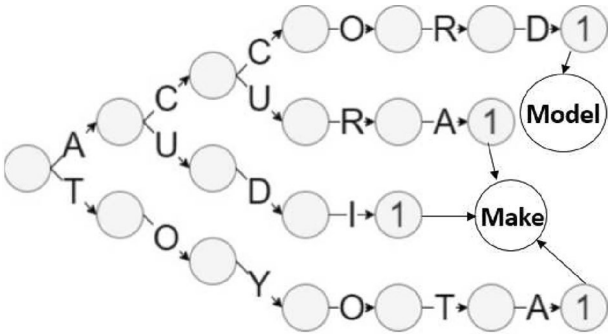

Fig. 2 A segment of the trie used by *QuePR* for typification

as Stanford's Part of Speech Tagger [31]), but value types correlate only loosely to parts of speech. Type I values are often *nouns*, but Type III tokens can often be nouns as well, since their units (such as "dollar" or "miles") are nouns. Type II values are sometimes *adjectives*, but other times can be *nouns*, even when describing the same attribute. For example, in "gold necklace", the Type II value 'gold' can be identified as an *adjective*, but in "necklace with gold", the Type II value 'gold' is a *noun*.

In lieu of a better typification approach, a trie data structure [32] is constructed for all the value types for each product category. Type I values are relatively easy to maintain, since the set of core identity values for a category is limited, whereas Type II values are more difficult to control due to their variation. There is no entry in the trie used by *QuePR* for Type III, since numerical values are infinite; however, bound and superlative phrases of Type IV are included in the trie.

When a token (i.e., word) in a query is encountered, *QuePR* searches for its type in the trie of the corresponding category. To expedite look-ups, tokens are stored in a trie and each letter of the token is a transition to a node in the tree structure. Each node in the trie, including the root, may have a path to another node on a single letter. To find the type of a token *K*, the path formed from the sequential order of letters in *K* is followed. If following the path leads to a *type* (I, II, III, or IV) node in the trie, then the particular attribute of *K* is given as the end node. For example, Accord, Acura, Audi, and Toyota are all words of Type I values for automobiles. (See Fig. 2 for a small segment of the trie used by *QuePR* for identifying the type of a given token in which "Model" and "Make" are Type I attributes of the corresponding strings in the trie.)

3.5 Standarization

To simplify the step of constraint clustering, some synonymous phrases, especially bounding or superlative phrases, are systematically converted into a standard, easier to parse, form. For example, the phrases "at most" and "at maximum" are synonymous, meaning that the desired value is *less than or equal* to the subsequent number. *Bounding* phrases are *comparisons* phrases. *Comparisons* specify a desired range relative to a fixed point. Examples of comparisons include "$<$", "$\leq$", "$>$", and "$\geq$". All Comparisons require a numeric value. *Superlatives*, on the other hand, define the optimal value, such as "MAX", "Largest", "MIN", "Least", "Fewest", and "Lowest". As such, they do not use a numeric value. *QuePR* simplifies common phrases to the symbolic or short form. Lastly, Boolean operation keywords *and*, *or*, *not*, and *not equal* are also simplified to $\wedge$, $\vee$, $\neg$ and $\neq$, respectively.

Bounds and superlatives are considered to be either *complete* or *partial*. In a *complete bound* or *complete superlative*, the product attribute to which it is applied can be inferred. Consider, for example, the query excerpts "cheapest" and "less than $6K". "Cheapest" is a *complete superlative*, since the attribute "price" is implied, whereas "less than $6K" is a *partial bound* applies to the price.

Another nuance contained in bounds is their *relative location* to the number they describe. Independent of the bound meaning, some *bound phrases* may only come before the number they describe, whereas others may only come after, or either before or after. For example, "less than or equal to" occurs before a numeric value, whereas "or less" appears exclusively after a numeric value, even though they both mean "$\leq$". Furthermore, a useful structure to standardize is *range phrases* that frequently appear in one of three forms which are identical in meaning. These forms are "between *x* and/to/-*y*", "from *x* to *y*", and "$x - y$". *QuePR* analyzes sequences of tokens in the query for any phrase matches, which if found, are replaced by the simplified form. (Table 2 depicts the most common phrases in a natural language query and their standardized form.) As an example of standardization, SQ_y is standarized as SQ_s: "$honda_1$, red_2, $accord_1$, $\vee$, $silver_2$, $civic_1$, 2008_3, $<$, $\$_3$, 6000_3, $New\ York_2$".

Table 2 Commonly-used phrases in a natural language query and their standardized form (STF)

Phrases	STF	Phrases	STF	Phrases	STF
At most[2]		Less than		At least[2]	
At maximum[2]		Fewer than		At minimum[2]	
Or fewer[1]	≤	Lower than	<	At fewest[2]	≥
Or less[1]		Older than		Or more[1]	
Or cheaper[1]		Cheaper than		Or over[1]	
Or older[1]		Smaller than		Or newer[1]	
More than		Maximum		Minimum	
Greater than		Most		Cheapest	
Exceeding	>	Greatest	MAX	Fewest	MIN
Larger than		Most expensive		Newest/lowest	
Over		Largest		Least/smallest	
Newer than		Oldest		Youngest	
Not (equal), no	≠	And (also), plus	∧	Or	∨

[1]Phrases must come after the bound; [2]Phrases may come before or after the bound; _: All phrases that have the implied units

3.6 Constraint clustering

Constraint clustering is the process of finding groups of juxtaposed tokens that work together to define a single constraint. The simplest form of constraint clustering is to use each token as a different constraint. For example, the short query "blue apartment", refers to a single noun chunk as defined by NLP techniques, but holds two different constraints: the *apartment* housing type and the *color* blue. This simple clustering is applicable to Type I and Type II, but not for Type III, which may contain more than one token in each constraint. For Type III constraints *QuePR* needs to find where one constraint ends and another begins to create clusters. However, in natural language, users will not always provide clear indications of which tokens go together. Consider the query excerpt "$7500 less than 9000 miles highest gas mileage". This excerpt contains two comparison constraints, i.e., "= $7500" and "< 9000 miles", and one superlative, i.e., "highest gas mileage", but there is no punctuation or explicit indication to divide the query into these constraints. *QuePR* creates clusters by identifying constraint forms and looking for corresponding component tokens. There are *two* forms of Type III constraints:

- *Comparison* constraints may have up to three components: a *number* (such as 100), a unit (such as $, miles, or feet), and a *bound* (such as < or ≥). The ordering of these components is not strictly defined, since a user may specify a price with a unit preceding the value (such as "$500"), but a mileage with the value preceding the unit (such as "9000 miles"). We cannot assume a consistent pattern for each unit, since a user may specify the monetary unit with a symbol that precedes the value, or an equivalent word representation that follows the value (such as "$500" or "500 dollars"). We cannot assume a consistent pattern for each bound either: synonymous bounds may occur either before the value (such as "no more than $900") or after the value ("$900 or less").
- *Superlatives*, where an extreme value of the product attribute is described. Some superlatives are *lowest* price, *newest* building, or *cheapest* jewelry, among others. Superlatives may have up to two components: a unit and a phrase, with no strict ordering.

QuePR traverses the token list looking for *comparison* and *superlative* constraints. If a constraint component is found, other adjacent components are added to the cluster, assuming that the components are needed by the constraint form and have not already been fulfilled. For the earlier example, "$7500 less than 9000 miles highest gas mileage", "7500" is identified as a number component, which often appears in comparison constraints. The token before, "$", is identified as a unit and added to the cluster with 7500. Hereafter, "less than" is encountered, which has been standardized to "<" with a location hint that it must appear before the number. Therefore, this bound cannot belong to the same cluster as $7500 and a new cluster must start. The final result is three clusters: ($7500), (< 9000 miles),

(highest gas mileage), in which "gas mileage" is a unit. With that in mind, SQ_s is clustered as SQ_c: "$honda_1$, red_2, $accord_1$, $\vee$, $silver_2$, $civic_1$, 2008_3, $(< \$\ 6000)_3$, $New\ York_2$".

3.7 Logical operators

While searching for constraints in a query *Q*, it is also necessary to simultaneously check for logical operators, i.e., or ($\vee$), intersect ($\wedge$), and negation ($\neg$), or a combination of them, in *Q*. To evaluate each detected operator, the operands need to be determined. In natural language, users do not use punctuation or symbols to demarcate the logical operands. In the example query excerpt, "Honda red Accord $\vee$ silver Civic", there is no indicator whether "Honda" and "red" are included as operands to the $\vee$ operator in addition to "Accord". As such, *QuePR* is required to make the best guess as to what the operands may be. In some simple cases, the operands can be decided as the first appearing constraint. However, in a natural language query, it often has more complexity, and users may include several constraints in a single operand. In the query excerpt, "Honda red Accord $\vee$ silver Civic", it is clear to a human what the user wants, i.e., a Honda car that is either a red Accord or a silver Civic, and each operand of the "or" operator contains two constraints, i.e., the *color* and the *model*. *QuePR* relies on some common patterns to make assumptions about what the intention of a user. The two patterns that *QuePR* uses are *descriptor ordering* and *operand symmetry*.

- *QuePR* uses the *natural ordering* of *descriptors* to determine which query keywords should be included in an *operand cluster*. Descriptors typically precede the object that they describe. For example, a car that is *blue* is the "blue car". Type I values are typically the objects, and Types II and III are typically descriptions. Thus, if a Type I value is detected immediately before an operator, any preceding descriptor values are included in the operand.
- *QuePR* relies upon *operand symmetry*, where operands have matching form. In the sample query *Q*, "Honda red Accord $\vee$ silver Civic", each operand contains values for the same types. The colors "red" and "silver" are Type II, and the models "Accord" and "Civic" are Type I. Once the *pattern* has been decided for one of the operands, the same *pattern* can be searched for in the other. Query SQ_c can be thought of as perfect symmetry, because all terms had an analogous component in the other operand. *Imperfect symmetry*, in which only some of the types match or exist, is also a notable variant. For example, "Honda red Accord or Civic" is an imperfect symmetry. In this case, there is still symmetry of the car model. *QuePR* uses (im)perfect symmetry by relying on the *left operand* to know where the *right operand* terminates.

Consider the list of constraints in SQ_c: $honda_1$, red_2, $accord_1$, $\vee$, $silver_2$, $civic_1$, 2008_3, $(< \$\ 6000)_3$, $New\ York_2$, there is one logical operator to evaluate. Operand evaluation immediately begins on the left of $\vee$, which is red_2 followed by $accord_1$, a Type II and Type I constraint descriptor constraint, respectively. As the leftmost constraint is $honda_1$, a Type I constraint, it is therefore not be a descriptor of $accord_1$. Thus, the left operand group contains red_2 and $accord_1$. The right operand of $\vee$ is grouped accordingly. The logically grouped result for SQ_c yields SQ_g, which is $honda_1$, (red_2, $accord_1$) $\vee$ ($silver_2$, $civic_1$), 2008_3, $(< \$\ 6000)_3$, $New\ York_2$.

3.8 Constraint simplification

Some user queries can be simplified, which can reduce the length of the DB query, and can avoid redundancy in (partial) matching and scoring. The first part of *query simplification* is to remove *redundancies*. In *QuePR*, query simplification occurs after *logical operator evaluation* but before *query translation*. QuePR removes some redundancies by looking for and removing any duplicates within each logical cluster. For Types I and II constraints, a duplicate is an *exact match* of value and linked attribute. For Type III constraints, the idea of duplication may also include the idea of *subsumption*, where one constraint may convey all information of another constraint. Subsumption may occur with the range operators ($>$, $\geq$, $<$, and $\leq$). For example, a constraint "price $\leq$ 1000" is a duplicate of "price $<$ 2000, but not the other way around. In such a case, the *less specific* of the two constraints is removed. Moreover, query simplification can also combine *overlapping* constraints to create a new constraint that captures the information of both. This can happen using the BETWEEN (in SQL) constraints, as well as $<$ and $>$ combinations, since BETWEEN specifies an inclusive range.

3.9 Query translation

Constraints specified by users must be translated to a query for execution. For the implementation of *QuePR*, we selected SQL as the query language of translation output. The same principles apply to any similar relational query language. There are three distinct components to each SQL query: the SELECT-FROM, WHERE, and ORDER BY clauses.

3.9.1 The SELECT-FROM clause

During the classification process, *QuePR* determines the category of the given query, which determines the corresponding table name in the underlying database scheme. To retrieve all the information to the user about each product, *QuePR* extracts all the columns of the table for each product using the wildcard (*). For the running query SQ_g, *QuePR* creates the SELECT-FROM clause in SQL as "SELECT * FROM Cars".

3.9.2 The WHERE clause

The WHERE clause is a logical combination of each of the previously parsed constraints specified in a user query. *QuePR* links each constraint to the corresponding attribute/column in the database table in the WHERE clause. Since in the natural language queries, users will not identify the correct table column in every case, *QuePR* is required to make inferences of the correct attribute based on the information provided in the queries. *QuePR* performs the matchings based on the "type" constraints.

- For Type I and Type II constraints, the corresponding table column can be determined by a local search of the value, called *value matching*, using our "trie" data structure. For example, if "Honda" and "Civic" are specified as constraints, *QuePR* looks up the corresponding trie on "Make" and "Model", which are created using the column values in the corresponding table to determine where the values appear and their corresponding column names.[3] Value matching has the potential to return multiple matches for a single value, since a constraint value could appear in more than one table column. For example, the word "new" may refer to the condition of the automobile, or it may refer to a part of a location name (such as "New York" or "New Jersey"). If the ambiguity was not able to be resolved earlier in tokenization, *QuePR* allows either alternative for a match by using the "Boolean Or" operator. Such ambiguity results in a WHERE clause "`condition LIKE "new" OR region LIKE "new"`".
- For Type III constraints, they cannot be identified by using their corresponding attributes based on value matching due to the infinite nature of numbers. A similar approach to the value matching, however, can be performed by using the unit of the Type III constraint instead of the value alone. For this strategy to be effective, two conditions should be satisfied: (i) each Type III attribute in the DB table includes a list of related units (which is anticipated) and (ii) there is an attached unit for each Type III constraint in the user query (which is often the case).

The WHERE clause of the SQL query is composed of the translated constraints of Types I, II, and III. The values of Type I and Type II constraints are compared with the column in SQL using the `LIKE` operator, which requires that the given substring appears in all matches. For Type III comparison constraints, if a comparison operator is not given, the equality operation (=) is assumed. For *range* Type III comparisons, the `BETWEEN` operator is used.

3.9.3 The ORDER BY clause

The "ORDER BY" clause is created for Type III *superlative constraints* and *implicit sorting order* of Type III comparisons. These implicit constraints come from sorting each result by its distance from the optimal value of the comparison. Equality comparisons have an optimal value of the number to be matched on, thus results can be sorted by `ABS(x - y)`, where x is the Type III attribute and y is the expected value. $<$ or $\leq$ operations have an optimal value of 0, whereas $>$ or $\geq$ operations have an optimal value of infinity, i.e., ∞. *Ranges* have no optimal value, and therefore do not create an implicit sorting constraint. For an example of query translation, consider the constraint clusters in SQ_g: $honda_1$, (red_2, $accord_1$) $\vee$ ($silver_2$, $civic_1$), 2008_3, ($< \$\ 6000)_3$, $New\ York_2$. In the WHERE clause, the constraint "honda" is determined to be car `Make`.

[3] Note that the trie is updated periodically to reflect the latest changes to the corresponding column values in the actual database table.

The constraints "red" and "silver" are `colors`, where "accord" and "civic" are car `Models`. "2008" is assumed to be a year, since it is within the range of `Years`. The constraint "< $ 6000" matches with `Cost` because of the dollar sign. Lastly, "New York" is matched in the `Cars` table as a `Location` value. Hereafter, the ORDER BY clause is constructed in which "2008" is the optimal value for `Years` and `Price` is minimized. The SQL for the constraints specified in SQ_g is

```
SELECT * FROM Cars
WHERE (make LIKE "honda" AND ((color LIKE "red"
        AND model LIKE "accord") OR (color LIKE "silver"
        AND model LIKE "civic")) AND year = 2008
        AND price < 6000 AND location LIKE "New York")
ORDER BY ABS(year - 2008), price;
```

3.10 Holistic (partial) matching

The DB query is generated through query translation (as presented in Sect. 3.9) and any exact product matches to the query constraints are extracted from the DB. This behavior is sufficient if there are many exact matches, especially if the user query is vague or there are many matched products. However, in the case that no exact matches or very few are found, it may be confusing or frustrating for the user. For this reason, *QuePR* supplements results with *partial matches*. Relevant product are searched for, first exact matches and then partial matches, until some threshold number of products is achieved. This threshold number of records to be extracted is set to be 15 based on the user's attention span determined by [33] to not overwhelm the user.

Partial matches are conducted by iteratively altering search constraints one by one and searching for products not previously retrieved. These alterations occur in rounds, numbered by the number of search constraints changed from the original. The 1st round is responsible for exact matches on all the constraints specified in the user query. The 2nd round is a union of all combinations of subqueries with 1 altered constraint. The 3rd round is a union of all unique subqueries with 2 altered constraints. This pattern continues for all needed rounds.

Example 1 Consider the five constraints specified in SQ_g:

1. Make LIKE "honda"
2. (color LIKE "red" AND model LIKE "accord") OR
   (color LIKE "silver" AND model LIKE "civic")
3. Year = 2008
4. Price < 6000
5. Location LIKE "New York"

The 1st round would use a query of $(1 \wedge 2 \wedge 3 \wedge 4 \wedge 5)$, whereas for the 2nd round $(\approx 1 \wedge 2 \wedge 3 \wedge 4 \wedge 5) \vee (1 \wedge \approx 2 \wedge 3 \wedge 4 \wedge 5) \vee (1 \wedge 2 \wedge \approx 3 \wedge 4 \wedge 5) \vee (1 \wedge 2 \wedge 3 \wedge \approx 4 \wedge 5) \vee (1 \wedge 2 \wedge 3 \wedge 4 \wedge \approx 5)$ would be used, where "$\approx x$" indicates that the xth constraint was altered, and so on. □

Constraints can be altered in two different ways: a *similarity replacement* or a *removal*. A *similarity replacement* captures more information about the original constraint, and is therefore done first. If no similarity replacement can be found for the value or if the similarity replacement does not yield any results, the constraint is *removed*.

To find *similarity replacements* of Types I and II values, *QuePR* relies on a *similarity graph* [34] for values of the classified product category. For example, in the automobiles category, similarity graphs are given to *QuePR* for automobiles *make, model, color, condition*, and *location*. To alter a value, *QuePR* replaces the value with a disjunction of all near replacements. For example, near replacements for the *color* orange include yellow, brown, and red. Thus, the constraint *color LIKE "orange"* can be replaced with *(color LIKE "yellow" OR color LIKE "brown" OR color LIKE "red")*. Figure 3 is the similarity graph for automobile color.

These similarity graphs can be produced by a number of different methods [35–38]. One of the methods is to reduce each value to some other quantifiable measure. For the example of color, each color has a red, green, and blue component. Two color's similarities can be computed by calculating the distance for each of the components. Another method

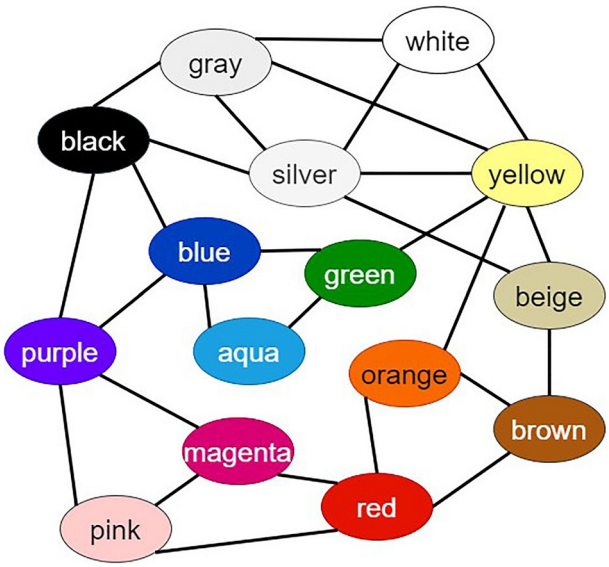

Fig. 3 The similarity graph of colors

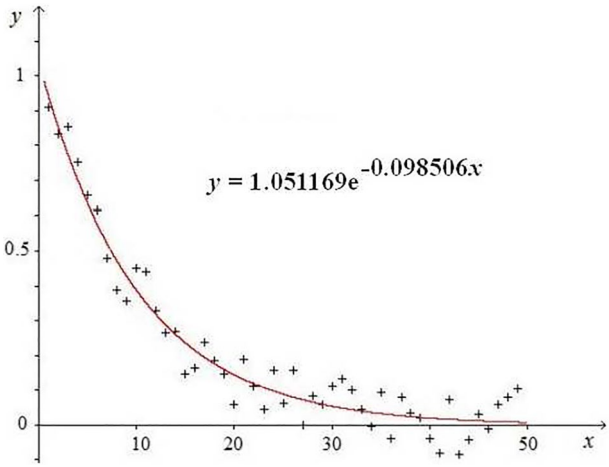

Fig. 4 An exponential regression graph

to create similarity graphs is by survey. A sufficiently large number of people may be asked questions similar to "If you were looking for a product with *X*, would a product with *Y* be acceptable?", where *X* and *Y* are values to compare similarity. Again for the example of color, the question could be asked, "If you were looking for a red automobile, would an orange-colored automobile be acceptable?" *Similarity* is the result of the percentage of respondents who deemed the replacement acceptable. We have derived degrees of similarity of items in a similarity graph based on the approach presented in [35], since the mapping rules developed for deriving the similarity measures based on WordNet [39] have been verified to be accurate.

3.11 Scoring and ranking

After a list of products *L* is retrieval with respect to a given query *Q*, *QuePR* sorts *L* according to the *degree of relevance* of each product in *L*. The degree of relevance of a product is determined by the matched constraints specified in *Q* and the product. If a constraint is met exactly, one point is awarded, whereas a partially-matched constraint receives a value between 1 and 0, exclusively. *QuePR* uses an *exponential decay curve* [40] to model *similarity regression* [41], which is contrast to other approaches which use a linear regression model [42]. The exponential decay formula is useful in a variety of real world applications, most notably for tracking inventory that is used regularly in the same quantity (like inventory for a car dealer) and it is especially useful in its ability to quickly assess the long-term cost of

use of a product over time. *QuePR* uses the *exponential regression model* [43] because it better fits common notions of both fixed and relative points (see Fig. 4).

The exponential model fits notions of similarity for fixed points. Linear regression models are typically set up such that

$$similarity = 1 - \frac{\mid expected - actual \mid}{expected}. \tag{4}$$

Eq. 4 has an inherent bound of applicability. In the case just given, any *actual value* that is greater than twice the *expected value* would result in an undesirable *negative score*.

Therefore, the formula is complicated by requiring the minimum similarity score to be 0. As a result, values that are beyond the bound of applicability are not differentiated for similarity. These cases are rather common for prices and mileages in cars, especially if the user has a low expectation. For example, if the user was looking for a car costing $10K and there were three alternatives, $11K, $20K, and $21K, the car for $11K should be most similar, then the $20K car, then the $21K car. A linear model may incorrectly rank the $20K car and the $21K car to have equal similarity. On the other hand, an exponential regression model has no inherent bound, and would rank the three similarities correctly.

Furthermore, the exponential model better fits notions of similarity for relative points. As the actual value becomes more distant from the expected value, the amount by which the similarity decreases should decrease. This is exemplified in the concept of automobile depreciation: An automobile which is brand new has 100% of its value, but may depreciate in value by 15% each year thereafter, i.e., the worth of the automobile decreases by a decreasing amount each year. Or in other words, a car from 2021 is worth less than a car from 2022, but the difference between the cars from 2022 and 2021 is greater than the difference between 2010 and 2009.

The concept of exponential decay also applies to *similarities of categorical values*. If the customer is looking for a *red* car, then *orange* is the closest color, but the difference between *red* and *orange* is larger than the difference between *black* and *gray*, since neither *black* nor *gray* are close in similarity to red. *QuePR* uses the exponential decay formula, which is Eq. 5 given below, to calculate attribute similarity score.

$$partial = 0.5^{x}, \quad \text{and} \quad x = \frac{2\times \mid distance \mid}{typical} \tag{5}$$

where *distance* is the difference between the *expected* value and the *actual* value, and *typical* is some measurement of how large of a difference is common for values of the specific attribute. This notion of *typical* scales scores to cooperate in computing the final score. If large differences are typical for a specific attribute, then each difference for that attribute will be minimized. Otherwise, a single attribute may trivialized or be trivialized by others. There are two different procedures for determining *distance* and *typical*, where one procedure is for Type III attributes and the other is for Types I and II attributes.

The first procedure for evaluating *partial match* is used for Type III values, which are inherently numeric, and thus it is possible to compare the found product value to the expected product value with arithmetic, i.e., *Distance* = *expected* - *actual*. *Typical* is set as the *standard deviation* of the numeric values in the attribute column.

The second procedure for evaluating *partial match* is used for Types I and II values. These values are not numeric, which poses a challenge for using the numeric formula. *QuePR* solves the problem by relying on the aforementioned similarity graphs. The distance from one value to another can be calculated as the shortest path from the first node to the other in the similarity graph. The typical value can be estimated as the *standard deviation* of shortest paths for all nodes in the weighted graph.

When evaluating constraints, it may be necessary to evaluate *logical Boolean operators*: $\wedge$, $\vee$, and $\neg$. A constraint of $\wedge$ returns the lower score of its two operands, whereas $\vee$ extracts the higher score of its two operands and $\neg$ returns $1 - s$, where s is the score of its operand. The final score for a product can be calculated as a *sum* of weighted constraint scores, where each constraint is weighted by *type*. The definition of each type implies an importance. Type I attributes are the *core* identity attributes of the product, and as such, are most important to the customer. Thus, Type I constraints are scaled by 1. Type II attributes are *customizable details* of the product. As such, Type II constraints are typically *less important* than Type I constraints, but more important than Type III constraints. They are scaled by 0.5. Lastly, Type III constraints are generally the least important than Types I and II, and are scaled by 0.25, which is a commonly-used geometric sequence [44]. Products are sorted in *descending order* before being presented to the user, and a *higher* score indicates a *more relevant* product.

Table 3 Sample natural language queries created by Facebook users

Automobile	Jeep SUV new less than 10,000 miles under $10,000
Furniture	Dining table with 8 chairs no longer than 8 feet long under 1000
Housing	Single family home with 4 BDR, 2 BATH, 2-car garage in Los Angeles, CA for $790,000 built in 1990
Jewelry	Combo of wedding band, ring, necklace, price $500 gold and silver
Jobs	Senior Data Scientist job, Private Company, MS Degree, 90k or 100k in Arizona
Motorcycle	2021 Suzuki Boulevard M109R Up to $15,000 Daytona Beach, FL

4 Experimental results

To demonstrate the effectiveness of the overall design of *QuePR*, we have conducted a number of empirical studies on its individual components, which include domain classification, query interpretation, exact query matching and ranking of partially-matched results. In addition, we have also measured the average run time in terms of processing a user query. In all these studies, *QuePR* either performed competitively with or outperformed contemporary systems. Each of these experimental results is described in details in subsequent sections.

4.1 Test data used for performance evaluation

Since there is no benchmark dataset available for evaluating the performance of natural language query retrieval systems on products, to obtain a representative test dataset for verifying the effectiveness of *QuePR* in retrieving exact- and/or partially-matched answers to users' queries, we have solicited natural language queries on six product domains through the Internet using Facebook. The six product domains we consider are *automobile, furniture, housing, jewellery, jobs*, and *motorcycles*. These domains or categories are diverse and representative of everyday living essentials, i.e., transportation, accommodation, employment, and personal adornment.

In testing *QuePR* and other baseline systems for the performance analysis, we downloaded and created a database of database records in each of the six different product categories listed above. The data for these records was found online for free and in the public domain through Kaggle [45], an online community platform for data scientists and machine learning researchers to collaborate with other users, besides finding and publishing datasets. Out of the different categories and datasets that we could have chosen on the Kaggle website, we found data that covered a diverse range of common products. The extracted test data was downloaded for the purpose of analyzing the performance of *QuePR* on domain classification, typification, implicit/explicit Boolean query analysis, and partial-matching of retrieved results. We also intentionally selected the similar categories of "motorcycles" and "automobiles" to test the classifier adopted by *QuePR*. Of the categories selected, we retrieved tables that contained descriptive records with at least five columns for each. Where possible, we fetched datasets with American dollar prices. For the others, we converted the price of the product by the current exchange rate to the American dollar.

4.2 Appraisers recruited for performance evaluation

To analyze the performance of different components of *QuePR*, it is required that outside appraisers provide objective feedback. In order to avoid any potential bias, we enlisted a number of Facebook users who are friends (of friends) of the authors to participate in the evaluation process. Facebook, which is a free-access social network, was chosen for conducting various empirical studies, since it reaches people from various age groups and backgrounds who can objectively perform the required evaluations. Facebook users have been used extensively in literature to conduct performance evaluation on a number of research projects [46, 47]. We recruited a total of 135 Facebook users who created different natural language queries for the six different product categories. Table 3 includes a sample query of each one of the six categories.

To determine performance accuracy, we relied on realistic user queries for categorization. We created a request for each one of the 135 Facebook users to response who created test queries: four simple queries and four complex queries among the six different categories. *Simple queries* provided constraints for 4 or fewer attributes about the

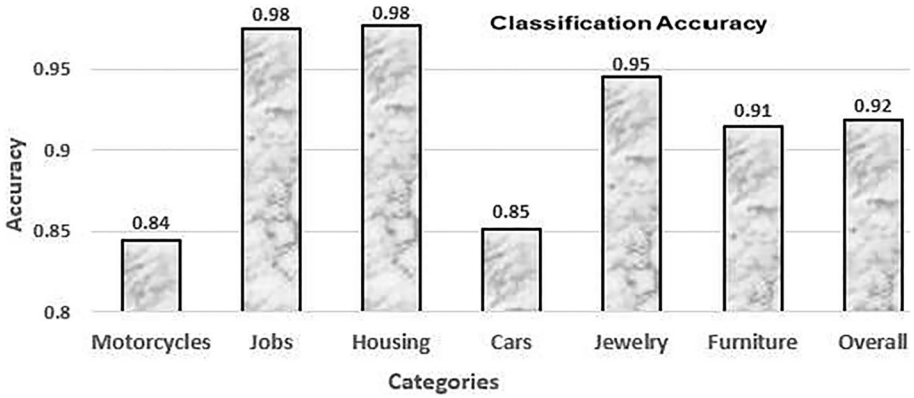

Fig. 5 Classification accuracy of the product classifier

product, whereas *complex queries* included constraints for 5 or more attributes, with the potential for implicit/explicit Boolean operators. The request was set up in September and October of 2021, and we received a total of 1080 natural language queries on products.

4.3 Accuracy on query classification

To evaluate the effectiveness of *QuePR* in *classifying* natural language queries among the six different categories, we rely on the accuracy ratio as defined below.

$$Accuracy = \frac{Correctly\_classified\_instances}{Total\_number\_of\_instances} \tag{6}$$

where *Total_number_of_instances* is the total number of queries to classify, which is 1080 in our case, i.e., 180 for each of the 6 different categories, and *Correctly_classified_ instances* is the number of queries correctly assigned to their corresponding categories by the Naïve Bayes classifier.

QuePR uses a Naïve Bayes classifier (see details in Sect. 3.1) to determine the product *category* to which each query applies. We tested the *accuracy* of the classifier by partitioning the training queries, training the classifier with 80% of them and testing it with the remaining 20%. The classifier determined the *correct* product category for 91.5% of tested user queries. The accuracy of classification was slightly lower for the *motorcycles* and *automobiles* categories, which is anticipated, due to their common attributes, such as Make, Model, Year, and Mileage. Nonetheless, the accuracy for these lowest categories was still in the mid-80 percentile. (See Fig. 5 for the accuracy ratios of our classifying the product queries into each of the six different categories.)

4.4 Accuracy on query interpretation

The recruited Facebook users also helped test the interpretation accuracy of our system. From the 1080 natural language test queries, we selected 210 different queries that varied in product category and query complexity. One hundred and eighty of these queries were run through *QuePR* to generate the system's interpretation. We crafted clearly incorrect interpretations for the other 30 queries, which served as the *controls* in the empirical study. Finally, each of the 135 Facebook users were presented with the same 10 randomly selected queries, with explicit/implicit Boolean operators, out of the 210 test queries, along with the associated interpretation. They were asked whether each interpretation presented was the "most reasonable" and allowed to select *yes*/*no*. This survey was posted for about a week in February of 2022. There were a total of 1283 responses[4] among the 135 Facebook users.

We ran *QuePR* on each of these test queries *Q* to generate the system interpretation on *Q*. These ten interpretations were presented, along with 2 randomly-chosen control interpretations out of the 30 control ones, to the 135 Facebook users between February 12 and February 20, 2022, who deemed *QuePR*'s interpretation to be "the most reasonable

[4] Some Facebook users did not submit their responses to our requests.

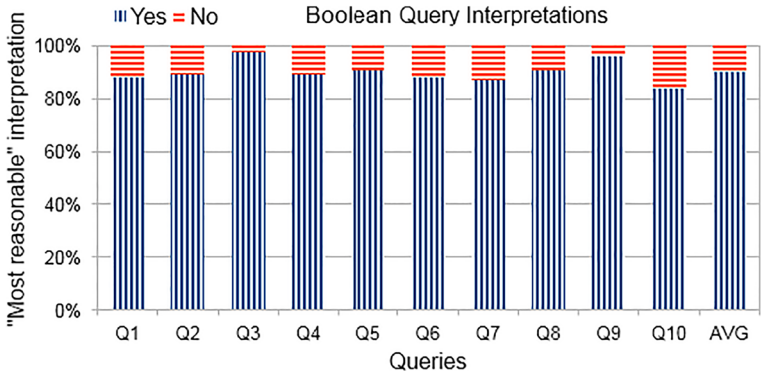

Fig. 6 Boolean query interpretation accuracy achieved by *QuePR* based on the responses of a Facebook survey

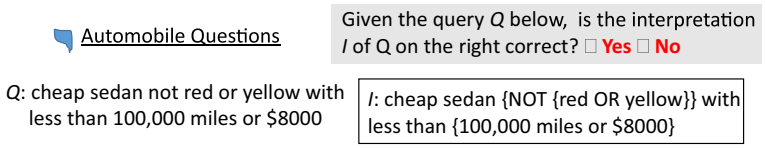

Fig. 7 A screenshot of an interpretation generated by *QuePR* on a query created by a Facebook user

interpretation" 90.0% of the time. We threw out the responses of all the respondents who answered both of the control questions incorrectly, which increased the statistical significance of our test. Figure 6 depicts the accuracy obtained for each Boolean query.

Observe that as shown in Fig. 6, it is not clear what was the intention by some of the natural language queries. For example, Query 10, "cheap sedan not red or yellow with less than 100,000 miles or $8000," is an ambiguous query, with two reasonable interpretations, i.e., "cheap sedan {{*NOT* red} OR yellow} with less than 100,000 miles or $8000" or "cheap sedan {*NOT* (red OR yellow)} with less than {100,000 miles or $8000}". Despite the ambiguity, *QuePR* was able to generate an interpretation for the query that was satisfactory to a majority of respondents. Figure 7 shows the screenshot of the query provided by a Facebook user and its corresponding interpretation created by *QuePR*.

4.5 Evaluation measures of retrieving exact-matched query answers

To measure the performance of *QuePR* in retrieving answers that exactly match the constraints specified in users' queries, we considered half of the queries created by our Facebook users, i.e., 540 queries, in one of our Facebook surveys and evaluated them using *QuePR*. The evaluation metrics for measuring the correctness of retrieving exactly-matched answers to a product query are (i) *precision* (*P*), which is the ratio of the number of *correct matches* retrieved by *QuePR* over the total number of *records retrieved* by SQL, (ii) *recall* (*R*), which is the ratio of the number of *correct matches* retrieved by *QuePR* over the number of *correct answers* in the DB, and (iii) *F-measure* $= \frac{2}{\frac{1}{P}+\frac{1}{R}}$, where a *correct match* is a retrieved (up till the 15th) DB record, that satisfies all the search criteria specified in a question.

We measured *precision* based on the correct (up till the 15th) DB records retrieved, and *recall* without restricting the total number of DB records retrieved. Since it can be non-relevant to measure the effectiveness of *QuePR* using *precision* and *recall* separately, we consider the *F-measure* which combines *precision* and *recall* with the same weight assigned to the two measures.

To analyze the correctness of the retrieved answers to the test queries, The averaged *precision*, *recall*, and *F-measure* for the 540 test queries, yield 94.2% precision, 93.3% recall, and an F-measure of 93.9%. We have observed that most of the test questions yield 100% for *precision* and *recall*, whereas a few yield 0%, i.e., answers are either correct or incorrect.

4.6 Accuracy on ranking partially-matched answers

In order to objectively measure the accuracy on the rankings of partially-matched answers to user queries, we compared the ranking strategy of *QuePR* with four other ranking approaches, with the most relevant retrieved product ranked first, and so on. We have implemented four baseline approaches on ranking, and compared the ranked results of each approach based on the 173 (out of the 1080) Facebook user created queries, denoted *Test_Queries*. Test queries in

Test_Queries do not yield any exact-matched results. Since all the ranking methods were given the *same* set of products, i.e., query results, for ranking, the comparison was fair.

4.6.1 Evaluation metrics

To measure the effectiveness of the ranking strategy of *QuePR* on partially-matched answers to queries, we apply two well-known information retrieval metrics, *average Precision at K* and *Mean Reciprocal Rank* [48].

The average *P@K* measures the overall user's satisfaction with the top-*K* ranked answers (generated by *QuePR*) to a particular query which measures the overall user's satisfaction with the top-*K* answers (generated by *QuePR*).

$$P@K = \frac{\sum_{i=1}^{N} \frac{Number\_of\_Related\_Answers_i}{K}}{N} \tag{7}$$

where *K* is the (pre-defined) number of answers to be considered, *N* is the total number of queries in *Test_Queries*, *i* is the *i*th question in *Test_Queries*, and $Number\_of\_Related\_Answers_i$ is the average number of answers (out of *K*) that are treated as *related* to the *i*th question by the appraisers who evaluated question *i*. Note that in our study, we set *K* to be 1 and 3, respectively, to evaluate the relatedness of the answers positioned at the *top-one* and the *top-three* most relevant results in the ranking, respectively.

Besides average *P@K*, we also evaluated the ranking strategy of *QuePR* using *Mean Reciprocal Rank* (*MRR*). *MRR* is the averaged sum of the reciprocal of the ranking position of the *first* related answer among the top-3 answers, if there is any, or 0, otherwise for each query in *Test_Queries*.

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{r_i} \tag{8}$$

where r_i is the average (position in the) rank of the *first related* answer to question *i* in *Test_Queries*, if it exists; otherwise, $r_i = \infty$, and *N* and *i* are as defined in Eq. 7.

Average *P@K* and *MRR* evaluate the ranking strategy of *QuePR*, such that the *higher* relevant answers are positioned in the ranking list, the *higher* their corresponding *P@K* and *MRR* scores are.

4.6.2 Ranking approaches to be compared

We have implemented and analyzed four other ranking approaches, besides the one used by *QuePR*: random ranking [49], TF-IDF-based FAQFinder [50], cosine-similarity-based Vector Space Model [51], and super-tuples-based AIMQ [4]. For each of these approaches, we analyzed the performance of their accuracy in ranking query results and computed their execution time.

Random ranking shuffles all the retrieved product records into some potentially new order by using a *random number generator*. In this case, we implemented the number generator using the pseudorandomness from Python's random module, which served as a useful baseline measure.

FAQFinder uses *TF–IDF* for ranking the *similarity* of different results to the constraints specified in a query in our case. Generally, *TF* is the number of times the product fulfills some constraint, and *IDF* is the relative rarity of that constraint being fulfilled by other products. The *TF-IDF* score of a product is computed by summing the *TF-IDF* value of all constraints involved.

$$TF_k = \frac{v}{|K|}, \quad \text{and} \quad IDF_k = \log_2 \frac{|R|}{|V|} \tag{9}$$

where *k* is a constraint, *v* is the number of times the product fulfills *k*, which is either 1 or 0 in our case, $|K|$ is the number of constraints in their corresponding query, *R* is the set of partially-matched records with respect to *Q*, and $|V|$ is the number of partially-matched records that fulfill *k*.

Cosine similarity ranking relies on the Vector Space Model (VSM). Each retrieved product is represented as a vector of numeric values, where each value indicates whether the correspondent constraint *C* is satisfied, which is '1' if it is, and '0' otherwise, and the score of the product is calculated as the cosine similarity between the product vector and the query constraint vector.

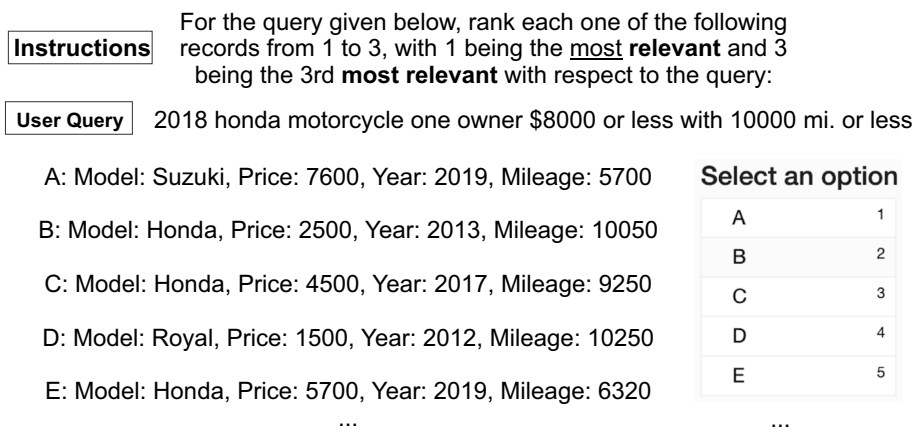

Fig. 8 A sample Facebook user query and (a portion of the) partially-matched DB records retrieved by *QuePR* to be ranked by Facebook appraisers who chose the top-3 most relevant record with respect to the given query

AIMQ relies on attribute-value pairs (denoted AV-pairs) to generate the associated supertuple of each attribute. A *supertuple* is an inferred DB tuple A that contains a set of attribute values, each of which includes a summary of values in the corresponding table column, and is used for calculating the similarity of categorical attributes. AIMQ determines the similarity between query Q and tuple A using Eq. 10.

$Sim(Q, A) = \sum_{i=1}^{n} W_{imp}(A_i) \times$

$$\begin{cases} VSim(Q.A_i, A.A_i), & \text{if } Domain(A_i) = Categorical \\ 1 - \frac{|Q.A_i - A.A_i|}{Q.A_i}, & \text{if } Domain(A_i) = \text{ Numerical} \end{cases} \tag{10}$$

where n is the number of attributes in Q, A_i in an attribute (in either Q or A), $Wimp(A_i)$ is the importance weight of A_i, which in our implementation of AIMQ is set to be $\frac{1}{n}$ for each attribute, $Q.A_i$ ($A.A_i$, respectively) is the value of attribute A_i in Q (A, respectively), $1 - \frac{|Q.A_i - A.A_i|}{Q.A_i}$ generates the *similarity* between any two numerical attributes $Q.A_i$ and $A.A_i$ in which $| Q.A_i - A.A_i |$ is the absolute difference between the two numerical attribute values $Q.A_i$ and $A.A_i$, and $VSim(Q.A_i, A.A_i)$, which is the similarity between two categorical attributes $Q.A_i$ and $A.A_i$, is computed using

$$VSim(Q.A_i, A.A_i) = \sum_{i=1}^{n} J(C_1.A_i, C_2.A_i) \tag{11}$$

where A_i and n are as defined in Eq. 10, $C_1.A_i$ ($C_2.A_i$, respectively) is the supertuple of $Q.A_i$ ($A.A_i$, respectively), and $J(C_1.A_i, C_2.A_i)$ is the *Jaccard Coefficient* computed as $| C_1.A_i \cap C_2.A_i | / | C_1.A_i \cup C_2.A_i |$, which is the proportion of attribute values in common between C_1 and C_2 for A_i and the distinct number of attribute values in C_1 and C_2.

4.6.3 Comparisons of the different ranking approaches

To avoid bias, the results for each test query were ranked by 135 Facebook appraisers. We conducted two different types of empirical studies to determine the *ranking accuracy* of each ranking approach as presented in Sect. 4.6.2. First, we had the Facebook appraisers rank the relative accuracy of results for each system. In other words, the appraisers were presented with a set of 15 answers to each test query for each system and asked to choose the top-3 results. This required that the top-3 results were more relevant than the remaining ones based on the corresponding query. Second, we asked the appraisers to rank the *relevance* of the top-3 result of all five systems for the corresponding query, i.e., to provide a *proper ordering* such that the 1st result was more relevant than the 2nd, and so on. This empirical study was performed between April 2 and April 11, 2022. Figure 8 depicts a sample Facebook user query and (5 out of the 15) partially-matched answers retrieved by *QuePR* to be evaluated by Facebook appraisers.

As shown in Fig. 9, *QuePR* outperforms the other four ranking systems based on *P*@1, *P*@3, and *MRR*, which verifies the *effectiveness* of the partial-matching strategy adopted by *QuePR*. A high *P*@1 (*P*@3, respectively) score implies that the ranking strategy of *QuePR* is highly effective in presenting first (top-3, respectively) answers that users are interested in (early to be examined by users, respectively). Finally, the *higher MRR* score obtained by *QuePR* compared with others indicates that *QuePR* users browse through *less* partially-matched answers before locating the ones

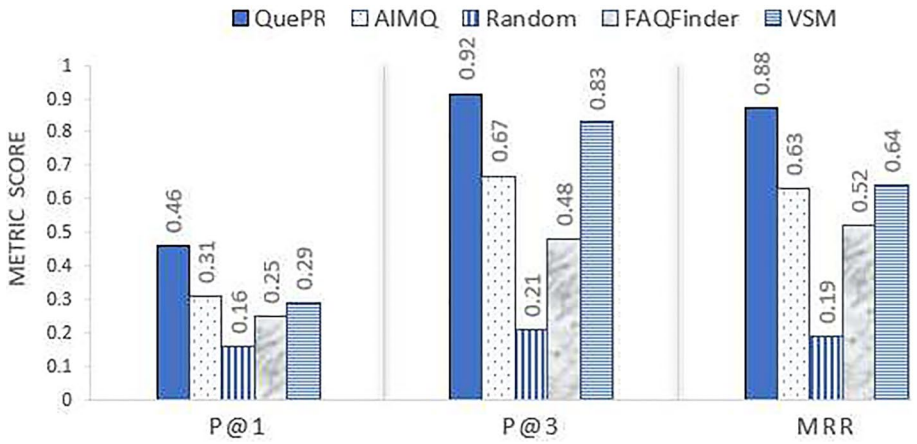

Fig. 9 *Precision@K* ($K = 1, 3$) and *MRR* scores on the (top-3) answers achieved by *QuePR* and other ranking approaches for the 173 test queries

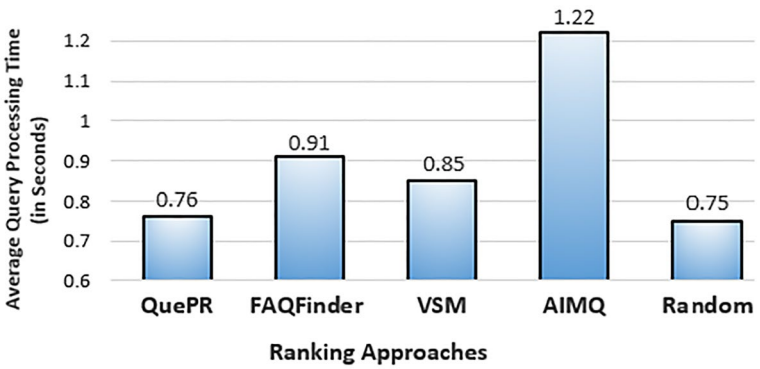

Fig. 10 Average query processing time of *QuePR* and others

relevant to their query. The results are *statistically significant* based on the Wilcoxon Signed-Ranks Test ($p < 0.01$). The three different metric scores were computed based on the rankings provided by the 135 Facebook users on the 173 test cases, which serve as the *ground truth* of the rankings for this empirical study.

Among all the five approaches, the *P*@1, *P*@3, and *MRR* values for FAQFinder are the *lowest*, except the Random approach, since FAQFinder uses a simple method that does not compare numerical attributes. On individual category, we observed that the lowest scores on the three measures for *QuePR* occur in the jobs category. For this category, appraisers did not rank the answers based on their similarity to the original query. For example, a Java programmer job is closely related to a C++ programmer job, but the appraisers ranked the answers based on which result is more relevant to their own expertise and experience, which is different from one user to another.

4.7 The efficiency of *QuePR* and other ranking methods

The *efficiency* of *QuePR* and four other ranking approaches are presented in Fig. 10, which shows the average query processing time for the 173 test queries obtained from the Facebook survey. *QuePR* outperforms all ranking approaches except the Random ranking strategy (by 0.1 s, on the average). This is due to the fact that the Random approach does not perform any processing or employ any similarity measures, but rather selects records randomly. Unlike the compared ranking methodologies, *QuePR* retrieves exact matches besides partially-matched answers, if needed. Thus, there is an additional time imposed on *QuePR* for retrieving (partially-)matched answers. As shown in Fig. 10, the query processing time for *QuePR* is the fastest compared to FAQFinder, SVM and AIMQ when partially- and exact-matched answers are retrieved by *QuePR*. *QuePR* was implemented using PHP and Javascript and runs on a MacBook Pro with Apple M1 processor and 16 GB of memory.

5 Conclusion

E-commerce is a large industry that is rapidly growing. Customers are most attracted to the sites that help them find the products that they are satisfied with. Product retrieval systems thus play an integral role in helping the user and thereby generating revenue for the site. For this reason, it is very essential to develop the ever-increasing relevance of efficient product retrieval systems. Existing product retrieval systems, however, have revealed a significant number of design faults and shortcomings, such as their inflexibility in user interface, inability to handle complex semantics in a natural language query, or inaccuracy in retrieving the most relevant results a product catalog has to offer. To address the drawbacks of these systems, we propose a new product retrieval system, called *QuePR*, which can (i) handle implicit/implicit Boolean operators in a natural language query, (ii) rank retrieved results accurately with an exponential similarity model and scalar weights for types of attributes, and (iii) perform best guess on incomplete or ambiguous queries. *QuePR* is easy to use, quick to execute, and yields accurate results. Furthermore, *QuePR* has been experimentally verified to extract relevant results and perform better than a number of baseline approaches in terms of (i) positioning properly ranked partially-matched products in satisfying the user's needs and (ii) optimal time in processing a natural language user query.

Author contributions All authors contributed to the study conception and design. MM, the first author of the submitted manuscript, investigated different methodologies in solving the product retrieval problem, implemented the design of the proposed retrieval system, and wrote the major portion of the manuscript text. Y-KN, the second author and the advisor of the first author, was involved in designing different components of the retrieval system, reviewed the manuscript text prepared by the first author, and worked with the first author in revising and finalizing the manuscript.

Funding Partial financial support for the submitted work was received from Brigham Young University in the form of research assistantships offered through the Department of Computer Science.

Data availability The *source code* of *QuePR* and the *datasets* are available at https://drive.google.com/drive/folders/1Re0yt2iK6qtK24itK6fFTo0WNbPySsSO?usp=share_link.

Declartions

Ethical approval This declaration is *not applicable*.

Completing interests The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Scrapehero. How many products does Amazon sell? (2021). https://www.scrapehero.com/how-many-products-does-amazon-sell-march-2021/.
2. Statista. Number of digital buyers in the United States from 2017 to 2025. https://www-statista-com.erl.lib.byu.edu/statistics/273957/number-of-digital-buyers-in-the-unitedstates/.
3. Wu J. A design methodology for form-based knowledge reuse and representation. Inf Manag. 2009;46(7):365–75.
4. Nambiar U, Kambhampati S. Answering imprecise queries over autonomous web databases. In: Proceedings of the 22nd international conference on data engineering (ICDE'06). IEEE; 2006. p. 45.
5. Sugiki K, Matsubara S. Product retrieval based on semantic similarity of consumer reviews to natural language query. Int J Knowl Web Intell. 2010;1(3–4):209–26.
6. Nurmi P, Lagerspetz E, Buntine W, Floréen P, Kukkonen J, Peltonen P. Natural language retrieval of grocery products. In: Proceedings of the 17th ACM conference on information and knowledge management. 2008. pp. 1413–1414.
7. Goddeau D, Meng H, Polifroni J, Seneff S, Busayapongchai S. A form-based dialogue manager for spoken language applications. In: Proceeding of fourth international conference on spoken language processing (ICSLP'96). vol 2. IEEE; 1996. pp. 701–704.
8. Yoshitaka A, Ichikawa T. A survey on content-based retrieval for multimedia databases. IEEE Trans Knowl Data Eng. 1999;11(1):81–93.
9. Croft W. Combining approaches to information retrieval. In: Advances in information retrieval. Springer; 2002. pp. 1–36.
10. Gregory K, Groth P, Cousijn H, Scharnhorst A, Wyatt S. Searching data: a review of observational data retrieval practices in selected disciplines. J Am Soc Inf Sci. 2019;70(5):419–32.
11. Kobayashi M, Takeda K. Information retrieval on the web. ACM Comput Surv (CSUR). 2000;32(2):144–73.

12. Lazarinis F. Combining information retrieval with information extraction for efficient retrieval of calls for papers. In: Proceedings of the 20th annual BCS-IRSG colloquium on IR 20. 1998. pp. 1–13.
13. Tredici MD, Barlacchi G, Shen X, Cheng W, de Gispert A. Question rewriting for open-domain conversational QA: best practices and limitations. In: Proceedings of the 30th ACM international conference on information and knowledge management (CIKM). 2021. pp. 2974–2978.
14. Ko J, Nyberg E, Si L. A probabilistic graphical model for joint answer ranking in question answering. In: Proceedings of the 30th international ACM SIGIR conference on research and development in information retrieval (SIGIR). 2007. pp. 343–350.
15. Ko J, Si L, Nyberg E, Mitamura T. Probabilistic models for answer-ranking in multilingual question-answering. ACM Trans Inf Syst (TOIS). 2010;28(3):1–37.
16. Liu X, Pan S, Zhang Q, Jiang Y, Huang X. Generating keyword queries for natural language queries to alleviate lexical chasm problem. In: Proceedings of the 27th ACM international conference on information and knowledge management (CIKM). 2018. pp. 1163–1172.
17. Li F, Jagadish H. Understanding natural language queries over relational databases. ACM SIGMOD Rec. 2016;45(1):6–13.
18. Kumar V, Raunak V, Callan J. Ranking clarification questions via natural language inference. In: Proceedings of the 29th ACM international conference on information and knowledge management (CIKM). 2020. pp. 2093–2096.
19. Li Y, Li W, Nie L. Dynamic graph reasoning for conversational open-domain question answering. ACM Trans Inf Syst (TOIS). 2022;40(4):1–24.
20. Sun H, Ma H, He X, Yih W, Su Y, Yan X. Table cell search for question answering. In: Proceedings of the 25th international conference on world wide web. 2016. pp. 771–782.
21. Sondhi P, Zhai C. Mining semi-structured online knowledge bases to answer natural language questions on community QA websites. In: Proceedings of the 23rd ACM international conference on information and knowledge management. 2014. pp. 341–350.
22. Omari A, Carmel D, Rokhlenko O, Szpektor I. Novelty based ranking of human answers for community questions. In: Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval. 2016. pp. 215–224.
23. Doan-Nguyen H, Kosseim L. Improving the precision of a closed-domain question-answering system with semantic information. In: RIAO. 2004. pp. 850–859.
24. Vargas-Vera M, Lytras M. AQUA: a closed-domain question answering system. Inf Syst Manag. 2010;27(3):217–25.
25. Papenmeier A, Kern D, Hienert D, Sliwa A, Aker A, Fuhr N. Dataset of natural language queries for e-commerce. In: Proceedings of the 2021 conference on human information interaction and retrieval. 2021. pp. 307–311.
26. Xu S, Li Y, Wang Z. Bayesian multinomial Naïve bayes classifier to text classification. In: Advanced multimedia and ubiquitous engineering. Springer; 2017. pp. 347–352.
27. Friedman N, Geiger D, Goldszmidt M. Bayesian network classifiers. Mach Learn. 1997;29(2):131–63.
28. Manning C, Raghavan P, Schütze H. Introduction to information retrieval. Vol. 39. Cambridge University Press; 2008.
29. Bird S, Klein E, Loper E. Natural language processing with python: analyzing text with the natural language toolkit. O'Reilly Media Inc.; 2009.
30. Brill E, Moore R. An improved error model for noisy channel spelling correction. In: Proceedings of the 38th annual meeting of the association for computational linguistics (ACL). 2000. pp. 286–293.
31. Toutanova K, Manning C. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In: Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora (EMNLP/VLC-2000). 2000. pp. 63–70.
32. Ghasemi C, Yousefi H, Shin K, Zhang B. On the granularity of trie-based data structures for name lookups and updates. IEEE/ACM Trans Netw. 2019;27(2):777–89.
33. Jones B, Kenward M. Design and analysis of cross-over trials. 2nd edn. Chapman and Hall; 2003.
34. Parambath SP, Usunier N, Grandvalet Y. A coverage-based approach to recommendation diversity on similarity graph. In: Proceedings of the 10th ACM conference on recommender systems. 2016. pp. 15–22.
35. Kwon J, Choi O, Moon C, Park S, Baik D. Deriving similarity for semantic web using similarity graph. J Intell Inf Syst. 2006;26(2):149–66.
36. Ma G, Ahmed N, Willke T, Yu P. Deep graph similarity learning: a survey. Data Min Knowl Disc. 2021;35(3):688–725.
37. Putra J, Tokunaga T. Evaluating text coherence based on semantic similarity graph. In: Proceedings of TextGraphs-11: the workshop on graph-based methods for natural language processing. 2017. pp. 76–85.
38. Zager L, Verghese G. Graph similarity scoring and matching. Appl Math Lett. 2008;21(1):86–94.
39. Fellbaum C. WordNet. In: Theory and applications of ontology: computer applications. Springer; 2010. pp. 231–243.
40. Provencher S. A Fourier method for the analysis of exponential decay curves. Biophys J. 1976;16(1):27–41.
41. Lambert S, Yang A, Sasse A, Cowley G, Albu M, Caddick M, et al. Similarity regression predicts evolution of transcription factor sequence specificity. Nat Genet. 2019;51(6):981–9.
42. Fang T, Lahdelma R. Evaluation of a multiple linear regression model and SARIMA model in forecasting heat demand for district heating system. Appl Energy. 2016;179:544–52.
43. Altun E. The log-weighted exponential regression model: alternative to the beta regression model. Commun Stat Theory Methods. 2021;50(10):2306–21.
44. Klapper A. Cross-correlations of geometric sequences in characteristic two. Des Codes Crypt. 1993;3(4):347–77.
45. com K.: https://www.kaggle.com/.
46. Nadkarni A, Hofmann S. Why do people use Facebook? Pers Individ Differ. 2012;52(3):243–9.
47. Wilson R, Gosling S, Graham L. A review of Facebook research in the social sciences. Perspect Psychol Sci. 2012;7(3):203–20.
48. Croft W, Metzler D, Strohman T. Search engines: information retrieval in practice. Addison Wesley; 2010.
49. Meng X, Ma Z, Yan L. Answering approximate queries over autonomous web databases. In: Proceedings of the 18th international conference on world wide web. 2009. pp. 1021–1030.
50. Burke R, Hammond K, Kulyukin V, Lytinen S, Tomuro N, Schoenberg S. Question answering from frequently asked question files: experiences with the FAQ finder system. AI Mag. 1997;18(2):57–57.
51. Li B, Han L. Distance weighted cosine similarity measure for text classification. In: International conference on intelligent data engineering and automated learning. Springer; 2013. pp. 611–618.