Expertise
and insight
for the future

Dang Nguyen

# Improving Ecommerce Search with Query Named Entity Recognition

Metropolia University of Applied Sciences

Bachelor of Engineering

Name of the Degree Programme

Bachelor's Thesis

30 June 2020

| | |
|---|---|
| Author | Dang Nguyen |
| Title | Improving Ecommerce Search with Query Named Entity Recognition |
| Number of Pages | 40 pages |
| Date | 30 June 2020 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineer |
| Instructors | Janne Salonen, Head of Department (ICT) |

Site Search is an indispensable feature of any successful ecommerce businesses. Shopping experiences will be ruined if users cannot find what they are looking for. Ecommerce search expectation are high as driven by leading players like Amazon and Google. However, many search sites are falling to build a good search experience.

This thesis focuses on the query understanding component of the search – the key to unlock next level of search relevance. Query understanding is an active area of research, giving rises to different techniques that aim at understanding the search intent behind the search query.

Among many tasks of query understanding, Query Named Entity Recognition (QNER) aims to decode user intent by identifying and classifying query segments of the search queries. The QNER process is the enablement behind many query transformations tasks such as query scoping, query relaxation, query expansion. In addition, it will simplify the rest of informational retrieval process and open the opportunities for advanced features such as search suggestion, personalization, and recommendation.

The objective of this thesis is to build search system for ecommerce enhanced with Query Named Entity Recognition. This thesis proposes a practical three-phases QNER process and implements it on top of the leading open source search engine Elasticsearch. A stateless search application was built, benefiting from QNER process by using it for query scoping. The outcome of the project is a performant, scalable search architecture enhanced with query understanding capability.

| | |
|---|---|
| Keywords | Search, QNER, Elasticsearch, Distributed System, Ecommerce |

**Contents**

**List of Abbreviations**

| | |
|---|---|
| NER | Named Entity Recognition – the task of identifying the classifying named entity in free text to predefined class such as persons, movies, … |
| QNER | Query Named Entity Recognition – the NER task performed specifically on search engine queries |
| SERP | Search Engine Result Page |
| JSON | JavaScript Object Notation – an open source, human-readable data interchange format |
| IR | Information Retrieval – the task of retrieving information system resources such as documents from a collection of resources to satisfy the information need |
| RPS | Requests Per Second – the rate of requests per time unit second |

# 1 Introduction

Ecommerce refers to the commercial transactions conducted electronically via the Internet. Example of ecommerce activities are the buying and selling of physical goods and services online. Without a doubt, ecommerce is playing an increasingly important role in modern life. In fact, ecommerce worldwide revenue, while taking a small share of 16.1% of total global retail, is growing strongly with forecasted annual revenue growth rate of 8.1% [1]. As global retail sale is growing at half the speed (4.5% annually), the line between physical and digital ecommerce is blurring.

For ecommerce businesses, site search is an indispensable feature. A well-engineered search is key feature to acquire and retain customers and thus a major contribution to the revenue. It is reported that on-site searcher is 216% more likely to convert then regular users. On-site searchers, in contrast to users who browse for inspiration, do search because they already have something of interest in their mind. A well-engineered search engine acts like a good sale person will quickly presents to user products that are relevant to their search intents. The secret sauce behind a useful search engine lies in the query understanding component. In fact, this is an interesting and active area of research.

Query understanding is the research umbrella for the tasks with goal to detect user intent. When the intent behind the submitted search query is known, search query will no longer be treated merely as string to match against text in the documents, but the meanings extracted from the query's keywords are used to deliver the sensible and relevant search result. Various query understanding tasks such as query classification, query transformation, query segmentation, and named entity recognition contribute to the overall goal of understanding user intent.

This thesis focuses on the ultimate task of query understanding - named entity recognition in search query or QNER for short. QNER is the cornerstone of the truly intelligent search engine. For example, QNER will enable various query understanding transformation tasks such as query scoping, query relaxation, and query expansions. Beyond search, QNER can enable important features that enrich user journey such as recommendation and personalization.

The objective of this thesis is to implement a performance and scalable search engine powered by QNER for ecommerce business. A practical three phases QNER framework is proposed and implemented on top of the leading open source search engine Elasticsearch. In addition, a client web application is built to facilitate the assessment of the search experience.

The thesis is structured as follow. The next section builds the theoretical background for the task of query understanding in search. The third section presents the goals of the project and documents the implementation and design decisions made. The fourth section evaluates the effectiveness of the NER powered search system a data set of 20k ecommerce products. This is followed by the fifth section, which evaluates the goals of project against its outcome and present improvement directions.

# 2 Literature review

## 2.1 Characteristics of search queries

Search query is the sequence of keywords submitted by users to search engine as a mean to fulfill their information need. Users will then be presented with search engine result page (SERP) containing result items matching the search criteria in form of free text. In web search engine, each result item from web page collection, or search snippet, is composed of the site URLs, the web page title, and the description which has relevant sentences from the page with matching keywords. In ecommerce context, result items are products which are often presented with relevant attribute such as titles, images, and prices.

Presentation and content of the SERPs heavily influences user interactions. Users rarely go beyond the first few results contained in the first page [2]. As a result, successful search engines present matching search items in the relevant, ideally personalized order to submitters. Furthermore, as users seldomly interact with other result collections such as images or videos, search engines often build landing result pages with most relevant items from different collections.

User activities within the search sessions are recorded by search engine in search log. During search sessions, users assess the search results presented for their search query, and either navigate to the presented resources or submit other search queries. Often, users refine the search queries further as more attempts to satisfy their information need are made. Search query logs in general contain four type of data [3]:

- Submitter information, such as user ID, browser user agent, IP address of the submitting host
- Submission information, such as original submitted query terms, timestamp of submission, and applied filters
- Search result information, such as number of matching documents, and document ranking
- Interaction information, such as clicked result item, filtering, or navigations.

Search log is invaluable source that gives insight into users' interactions. Because of this, many researches into search log have been conducted to understand the nature of search queries. The first large scale search log analysis work was done by Silverstein et al. in 1999 on a data set of approximately 1 billion queries [2]. The paper presents finding about the analysis of individual queries, query sessions, and query duplication. The subsequent research efforts from Croft et al. [4], Bicardo Baeza-Yates et al. [5], and Spink et al. [6] [7] drilled down further in frequency distribution and length aspects of the search queries. The gist of their finding will be summarized.

The distribution of search query frequencies follows the power law distribution. Alike a myriad of biological, physical, and natural phenomena following the power law distribution, whereas the majority of the search queries appear in low frequency, a fraction of search queries account for the majority of the search requests. For instance, finding in [5] indicated that 50% of search queries collected over one-year period were distinct. Similarly, 63% of queries are unique in research by Silverstein et al. [2].

Users seldomly modify the search query within a search session. This is supported in research by Silverstein et al. where staggering 77% of search sessions included only one query [2]. Furthermore, the share of search session having more than three search queries was merely 4.5% as found in the same research.

Search query is very short in length. In average, search queries contain between 2.3 to 2.6 keywords [6] [7]. The number of queries contained up to 4 keywords make up to 90% of the search volume as reported by Croft et al. [4]. Furthermore, the brevity explains why search queries are often highly ambiguous. For instance, the search query “apple” could mean a type of fruit, or a phone company.

## 2.2 Search query understanding

Modern search engines consist of five major components: query understanding, user understanding, document understanding, document ranking, and monitoring and feedback [3]. Query understanding component detects the search goal behind the query in orders to provide relevant results in appropriate presentation. Document understanding transforms searchable entities into representations carrying content and importance. Document ranking component boosts visibility of relevant documents with regards to the submitted search query. User understanding component further sorts documents based on user profile to provide personalized search experience. Monitoring and feedback component outputs metrics indicating effectiveness of the search system to help steer and refine search.

Research efforts focuses on the field of query understanding can be further classified into the following sub tasks:

- Query classification, which aims to classify queries along different dimensions such as search goals, sematic classes.
- Query transformation, which aims to rewrite the original query without compromising the original search intent.
- Query segmentation, which aims to divide search query into phrases or sematic units.
- Named entity recognition, which aims to locate and classify named entities exist within the query.

Understanding intent of search queries is an integral part of an effective search system. As big players such as Amazon and Google continuously set new expectation for search experience, treating search query as a mere string can hardly satisfy user experience. Understanding search goals of the search queries is the key to the next level of search

relevance. Query understanding component helps the retrieval of relevant documents, improves ranking of relevant search result, and enables relevant search presentations. It can enable the implementation of other features such as query completion [8], and query suggestions [9]. The following sections walk through the aforementioned sub-tasks of query understanding in detail with their applications.

## 2.3 Query classification

In the literatures, major dimensions that queries classification were performed on are goals and topics. In query classification task, the whole query is classified without analyzing further on the its internal structure.

The earliest research on query understanding aim to address the task of identifying search intents. From the search users' point of view, search goals could be navigational, informational, transactional [10]. Users have navigational intent when they want to go to a specific site. Informational intent is present as user want to find information assumed to be present on some webpages. Intent is transactional when users execute web-mediated actions such as buying furniture or register for an event. In any specific case, search intent of the user should be clear, either navigational, transactional, or informational. Researches on this topic outputs various heuristics applied on three main class of features: i) statistics draw from web page in containing query keywords ii) query log-based statistic such as users' clicked links and iii) query feature - query length and query keywords [11]. There are still challenges on identifying search intent for tail queries as much less data is available for the classification. Queries are ambiguous, inherently subjective, and latent. In additional, the same search query might have different intent in different context and time. Because of this, the ability of search engine to accurately identify search goals remains controversial. Research focus shifted towards other tasks of query understanding.

Topical classification tasks aim to assign topic labels such as "Business", "Entertainment", "Commercial" to search query. The task is often useful in determining the search result presentation or advertisement targeting. The semantic class used as label can be coarse (e.g., "commercial" and "non-commercial") or fine-grained (e.g., thousand nodes commercial taxonomy). To overcome the short and ambiguous nature of queries, several approaches leverage query enrichment with search snippets such as works by Shen et

al. [12] and Border et al. [13]. Many approaches make use of clicks through bipartite graph. The learning problem becomes the propagation of the labels from the labelled nodes to the whole graphs, assuming similar nodes have similar click patterns. There exist approaches that simply use search queries mined from the search log but large sets of manually classified queries are prerequisite. Overall, training a topic model is costly as it requires manual human evaluation to create labelled dataset.

## 2.4 Query transformation

The task of query transformation aims to alter the original query to produce queries having exact or similar original intent. For example, query “vp salary” can be rewritten as “vice president salary”, resulting in more search result and still being relevant. Query transformation is also known as as query rewriting, query alteration, or query finding in the literature.

Query transformation techniques in general increases the recall of the search result. Recall, also known as sensitivity, is the fraction of the relevance instances that were retrieved. For example, if recall is 50%, half of the relevant instances were included in the result set. Techniques introduced after this section will serve to increase query precision.

Technique mean of query expansion and query relaxation, or spell checking. Techniques that contributes to increase recall are query expansions, query relaxation. Method such as query segmentation and query scoping help enhancing precision. Spell checking is very important query transformation task to help increase both recall and precision.

### 2.4.1 Query expansion

Query expansion adds to original query more terms and phrases to increase query recall. Additional tokens maybe be the synonyms, abbreviations, and unabbreviated terms. Lemma and stem of the query terms can also be added to the set of the additional terms. Spelling checked terms can also be leveraged.

Additional tokens are employed to provide alternative matches for the original terms. If the original query is an *AND* of tokens, query expansion replaces it with *AND* and *OR*.

Alternative terms mainly help to increase the recall by matching more documents. In addition, expansion terms can also help to increase precision as matches containing additional terms can be more relevant to user search intent. The presence of the expansion terms can also be an indicator of relevance.

### 2.4.2 Query relaxation

In contrast to query expansion, query relaxation, also referred to as query reduction, removes or make the original query terms optional. For instance, omitting term “elegant” in “elegant blue office shirt” query can provide more results that are still relevant original search intent. Query relaxation technique increases recall by not requiring one or more query terms to be present in the matching documents. Query relaxation should be employed carefully because optimizing terms central to original query will completely change the intent of the search query. For example, omitting term “shirt” in “casual shirt” query might lead to completely irrelevant result set. Because of this, advanced query analysis should be employed to identify the main concept in the query and optionalize terms that serve as modifier. In general, it is good to apply this technique for no hits search query, as irrelevant search result page in many cases are still better than empty one.

Major approaches in implementing query relaxation in increasing the order of complexity are stop words, syntactical analysis, and semantic analysis.

Stop words can be viewed as a curated set of words in a certain language that should be filtered out. Stop words are usually the most commonly used word in a language, such as determiners (examples: the, a, an), coordinating conjunctions (examples: and, or, so) and preposition (in, under, before). Such common words are removed from the search queries so that important words can be used to identify relevant documents more effectively. For example, in an ecommerce search context, search query “the dress for office” can have stop words “the” and “for” filtered out.

Knowledge into linguistic structure of the search query can help determine which tokens can be made optional. Large fraction of search queries is noun phrases. Using part-of-speech tagger, the head noun and the modifying adjectives can be identified. A reasonable relaxation strategy is to preserve the head noun and remove its modifiers.

Semantic analysis goes beyond the token frequency and syntax to consider tokens' sematic or meaning. For example, search query "polo shirt" can be simplified as "polo" as the term imply shirt category. However, the query "button shirt" should not be relaxed as "button" as the two queries have completely different search intent.

### 2.4.3 Spellcheck

Search engine queries are often misspelled. According to Cucerzan and Brill [14], the share of misspelled search query is between 10% to 15%. This problem plagues tail queries even more, where approximately 20% are not correctly spelled [15]. Because of this, spelling error correction in query is a very important task of query transformation. The spellchecked queries will deliver improvements in both recall and precision for the search.

Search queries might be spell-checked in different ways. For example, query term "juse" could be spellchecked as "just" or "juice". Spellchecking is then translated to the task of calculating the probabilities of correction $c$, out of all possible candidate corrections, given the original word $w$. The most probable candidate is then proposed as the spellcheck: $max_{c \in candidates} P(c|w)$. By Bayes' conditional property theorem, this is equivalent to $max_{c \in candidates} \frac{P(c)P(w|c)}{P(w)}$. Since $P(w)$ is the same for every possible candidate $c$, it can be factored out, giving the following equation for probabilistic spelling correction:

$$max_{c \in candidates} P(c)P(w|c)$$

Three part of this expression are:

1. The candidate model $c \in candidates$, which provides the set of candidates to be scored.
2. Language model $P(c)$, the probabilities that $c$ appears as a word.
3. Error model $P(w|c)$, the probability that $w$ is used when users mean $c$.

To build candidate model, the main ideas are based on edit distance or phonetic similarity. Candidates can be set of words that are 1, 2, or more edit distance away from the original word. The minimum number of edit operations required to transform a string to

the another is defined as edit distance. It is used to quantify how dissimilar two strings from one another. Levenshtein distance are the most wildly used edit distance with the 4 simple edit operations: insertion, deletion, transposition, or replacement. It was found that the great majority (about 80%) of wrong spelling was 1 edit distance away from the correct one.

Candidate model can also be built with the idea that misspelled word is phonetically similar to the intended word. For instance, central to the Soundex system devised by Knuth 1973 is an algorithm that encodes words to the form that preserves the salient features of the pronunciation. A lookup by Soundex code can be built and used as a lookup for word with similar pronunciation.

A simple language model can be built by computing the frequency that words appear in the corpus – a collection of written text. Large corpuses benefit the spellchecking process as it is more unlikely to encounter words unknown to the dictionary based on them. However, with large corpuses come more rare words and more correction candidates, requiring more sophisticated error models to select the more probable candidates. More sophisticated language models factor in the context around the misspelled words. For example, correcting the word “where” to “were” only make sense in certain context, such as “I where going”.

The error model, which tells how likely a candidate is given the misspelled word, can have a varied degree of complexity. It can be as simple as treating candidates with edit distance of one equally probable and are infinitely more probably than candidates with two edit distance. More sophisticated approaches, such as the one developed at Bell Labs for correcting typing errors, is capable of assigning probabilities for concrete edit operations. For example, it correctly picks *actress* as the most probable candidate for word *acress* among candidates such as *access, across, caress* as the combination of letter t omitted after a c combined with frequency of word *actress* gives higher probability overall. To make this work, a significantly sized corpus of spelling errors is necessary.

### 2.4.4 Mining similar queries

In contrast to rule-based query rewriting approaches like query expansion and query relaxation, search queries with similar intent can also be discovered directly from the

search log. While this approach would perform poorly on tail queries, it is practical for head queries.

Click-through Bipartities graph is a good source sources of information where similar search queries can be mined. It is observed that similar queries leads to similar URLs clicks. Because of this, click-through Bipartities graph can be leveraged to calculate similarity score or to perform query clustering. In [16], Xu calculated query similarity from click-through bipartite using the Pearson correlation coefficient. They found that when the Pearson coefficient is larger than 0.8, more than 82.1% of query pairs were similar query pairs. In the work by Berger and Beeferman [17], a simple agglomerative clustering algorithm for clustering similar queries using a click through bipartite graph successfully discovered high-quality query clusters.

Search session data is also a source for mining similar queries, as user sometimes submit similar queries in the same search sessions. Jones et al. devised a method to calculate the statistical significance of co-occurred search queries [18]. When the likelihood of two queries being submitted within the same session is higher than the defined threshold, two queries can be considered as similar or substitutable.

## 2.5 Query segmentation

Query segmentation works by dividing the search query into a sequence of sematic units. Each semantic unit, composed of one or more tokens, carries single concept or meaning and should be treated as a phrase. Such phrases are then used by the retrieval system as indivisible units in order to improve retrieval precision. Ideally, users assist search engines by quoting the phrases. In practice, users are not usually aware of this option. Hence, the search engines apply the query segmentation algorithm prior to retrieving documents to predict user intended phrases.

The number of possible query segmentations grow exponentially with the number of keywords in query. Concretely, for query with n keywords, the number of possible segmentations is $2^{n-1}$. For example, there are 4 ways to segment search query ***"machine learning courses"***:

- [machine] [learning] [courses]

- [machine learning] [courses]
- [machine] [learning courses]
- [machine learning courses]

The main use case for query segmentation is to improve precision of the search system. Treating query segments as indivisible units mean that for segments containing more than one word, the words of the segments will exist in the retrieved document in the same order as they exist in the segments. At the core, search engine treats search terms independently and return relevant research results containing the search terms ignoring the order in which they are specified in the query. Because of this, without query segmentation, any documents containing “table”, “football” will be returned for search query “table football”. Irrelevant result titled “Premier League Table – Football – BCS Sport” will be included in the result set and negatively affect the retrieval precision. By treating multiple words as single unit when appropriate, false positive can be filtered out. In summary, the intent of the user is more closely captured by looking at the intent unit (query segments) in the queries, rather than tokenizing on white space of individual term-based retrieval.

Query segmentation also assists in others query understanding tasks, such as query transformation. When a query is divided into segments, each representing a semantic unit, query expansion and query reduction should be performed on the segment rather than at word/token level. For example, when query “ac adapter and battery charger for macbook pro” are divided into segments “[ac adapter] [and] [battery charger] [for] [macbook pro]”, the query reduction can be applied to produce new query “[battery charger] [macbook pro]”. This improvement could be huge in term of recall especially for long queries, which often suffer zero-recall problem [19].

Query segmentation is an important part of Named Entity Recognition process. The goal of Named Entity Recognition task is to further identify the semantic meaning of each query segments. Named Entity Recognition will be discussed in detail the next chapter.

Two major approaches to tackle the task of query segmentation are:

- **To segment or not**: given a query and a position in the query, decide if the new segment should start.

- **Segmentations scoring:** select the optimal segmentation with the best overall score composed of individual segment scores.

Bergsma and Wang [20] presented a supervised learning approach in which segmentation decisions are made at each adjacent word pair. Given $k$ word query, $k-1$ segmentation decisions are made. A Support Vector Machine classifier was trained for the problem utilizing lexical features and corpus-based features. Lexical features include the part-of-speech, keyword, and break position; corpus-based features include the cooccurrence frequency of the surrounding keywords and keyword count. After that, Bendersky et al. presented an efficiency improvement to this algorithm by introducing two-stage model [21]. In the first stage, the query is segmented into noun phrase chunks using a Conditional Random Field phrase chunker. The second stage further breaks down noun phrases into smaller segments, using feature similar to those exploited in [20]. The main challenge of supervised learning approach is the requirement of a large set of manually segmented queries. This could be unsustainable as search queries evolve quickly.

Subsequent research focus more on unsupervised approaches. For example, Tan and Peng in [22] built a probabilistic model to score segmentations. The probability of a segmentation is estimated by $P = \prod_{i-1}^{n} P(s_i)$ for query consists of n segments. The probability of each segment $P(s_i)$ is calculated from the language model based on concepts built over a web corpus. To increase segmentation accuracy, segments appear as title of Wikipedia article are assigned higher weight.

As a departure from sophisticated probabilistic approaches for query segmentation, Hagen et al. proposed a naive query segmentation technique that is equally competitive [23]. This method scores all segmentation of a given query by the weighted sum of the frequencies of contained segments:

$$score(S) = \sum_{s \in S, |s| \geq 2} |S|^{|s|} \cdot freq(s)$$

$S$ is a valid segmentation of query $q$, consisting of disjunct segment $s$, each a contiguous subsequence of $q$, whose concatenation equals $q$.The weight $|S|^{|s|}$ factor gives significant boost to long segments compared to shorter ones, compensating the power law distribution of occurrence frequencies on the Web. In follow up work, the authors group leveraged Wikipedia to boost the weight of segments that exist in the article titles.

## 2.6 Query Named Entity Recognition

In simple terms, Named Entity Recognition for search query (QNER) is the task of identifying, and classifying semantic units in the search query to categories or entity classes, such as persons, organizations, brands, categories, colors, times, prices, quantities. Performing this task with high accuracy is considered as major leap in query understanding because it simplifies the rest of the query retrieval system. For example, given search query “blue running shoes adidas performance”, QNER would output identifies *blue* as COLOR, *running shoe* as CATEGORY, and *adidas performance* as BRAND. Color, category, and brand are predefined categories or entity classes, and blue, running shoe, and adidas performance are concrete instances of such entity classes. In the context of search query, QNER is also referred to as Query Tagging.

The set of categories, or entities classes is domain dependent and often predefined. For instance, in fashion domain, useful attributes are brand, pattern, material, color. In contrast, in car domain, the set of attributes could be manufacturer, model, year, travel distance, or rim type.

### 2.6.1 QNER applications

Thanks to QNER, various improvements can be made to the query understanding component. For example, QNER can simplifies the implementation of precision-improving query rewriting techniques such as query scoping, and recall-increasing techniques such as query expansion and query relaxation. Ultimately, QNER breeds search systems where the tradeoff between precision and recall can be fine-tuned.

One of the important application of entities-tagged queries is query scoping. Query scoping matches the query segment to the right attributes. This is an effective measure to improve precision of the search result. For instance, for simple query “blue shirt”, “blue” will be matched against color and “shirt” will be matched against category. NER helps to specifically identified user intent in this case, and thus irrelevant shirts from brands containing the word blue such as “united by blue” can be filtered out. QNER implemented as multiclass classifier might produces multiple labels for given query unit. In this case, the query unit might be matched against multiple, best attributes. Determining the number of attributes to scope the matching is a precision/recall tradeoff.

QNER can help in query transformation tasks such as query expansion and query relaxation. Query expansion can be more performed more accurately if the classes of the query unit being expanded are known. For instance, additional terms can be introduced to broaden the sematic scope of the search query. A search query for “running shoe” can be expanded as “[running shoe] OR [sport shoe]” if “running shoe” is identified as category whose parent category is “sport shoe”. This can make a big difference if original query returns no result. Similarly, with the understanding of the semantic unit of the search query, query relaxation can more accurately optionalize or relax the less significant semantic units in the search query. For instance, “colorful puma sweatshirt” can be relaxed to “puma sweatshirt” as colorful color is most likely less significant to search intent compared to the puma brand and sweatshirt category.

QNER can also be leveraged to classify the search intent. For instance, if the search query is identified as product identifier, user search intent can be detected as navigational. Then, it would be best to direct users at the product detail page of the shop. As another example, if the search query “Adidas” is labeled as brand, search system can help users to directly reach the brand page.

Besides being a game changer to the query understanding process, result produced by QNER can be leveraged to enhance multiple touch points across user journey. Search experience can be made more meaningful, educative and fun with the knowledge boxes. For example. for search queries about single entity instance, users can be presented with knowledge and fact around it. Detected named entity in search query may also help with providing more relevant search suggestions. Personalization can be improved by taking the signal in submitted search queries. For instance, user’s affinity with certain brands is spoken when the submitted search queries contain the brands. User profiles can be built dynamically based on this. User experiences will improve if along the journey, more relevant products are surfaced. Related entities can be recommended to user throughout the journey to eliminate dead end and to make the navigation more fluid.

### 2.6.2 Methodologies

In the literature, there are many approaches to NER in search query. In general, approaches to QNER can be classified as statistical, semi-supervised, and unsupervised.

There exist many QNER models that relies on large human input effort in form on hand-crafted rules or training data set. For example, in [24], the NER model for 200 entity classes was backed by manually tagged entity 130000 entities and 1400 hand-made rules. Similarly, McCallum and Li trained a Conditional Random Fields (CRF) model for QNER task with lexicons obtained from the web [25]. In recent work, Eiselt et al. presented a supervised QNER approach that trains a two-step CRF models using 80,000 manually labeled search queries [26].

One common supervised approach to NER in search query is bootstrapping. Started with few seed instances of a target class, all search queries containing the seed instances are extracted from the query corpus. Search queries containing seed instances are parsed for contextual information, which will be used to induce new instances having similar contextual clue. When new instances are identified, search queries containing the new instances are used to build richer contextual clue for the entity class. The algorithms alternate between extracting instance and extracting contextual information.

Different bootstrapping approaches have different way to induce the contextual clues and induce the new class instances. In [27], Pasca presented an approach where remainders of all search queries containing the known entities belonging to same class are aggregated into a single vector called class search-signature. The frequency of the query in the query log is given as the weight to each query pattern (the remainder of query). The query patterns are exploited to produce new candidate instances having identical patterns. For each candidate instance, a search signature vector is built from the remainder of all search queries containing it, which is then used to compare with the class search-signature. Jensen-Shannon divergence that quantifies distributional similarity is used to compute similarity score between search query and class search-signature, which is used to decide its membership to the target entity class.

Alasiry et al. improved upon existing bootstrapping approach by exploiting top search result snippets of the search query as contextual clue [11]. Starting from seed entities mined from semi-supervised algorithms, contextual clues of search snippets of search queries containing seed entities are used to build Bag of Word Context (BoWC) Class Vector, where each context word is assigned the weight as its frequency. As new candidate named entities are assigned to the target entity classes on condition that similarity of its Entity Vector is close enough, the Class Vectors are expanded with every iteration.

The similarity measure employed was cosine similarity commonly used in text mining especially for spare vectors [28].

In contrast to deterministic approach presented by [11], probabilistic approaches where one entity might be long to multiple class was presented by Guo et al. [29]. In their work, the problem of classifying entity e appearing in query context $t$ – words in the query surrounding $e$ – as a class $c$ is formulated as:

$$\Pr(e, t, c) = \Pr(e)\Pr(c|e)\Pr(t|c)$$

This is a topic modelling problem, where named entity corresponds to the document, the contexts of entity t corresponds to words of the document, and the classes of a named entity correspond to topics of the model. A weakly-supervised version of Latent Dirichlet Allocation – a topic modelling technique – was exploited. In the first phase of offline training step, bootstrapping technique in [27] was used to generate training data set in the form $(e_i, t_i)$, which is used to train the topic model that gives $\Pr(c|e)$ for each of the seed named entity and $\Pr(t|c)$for each class. In second phase, query corpus is scanned again to get all queries containing the contexts, extract candidate entities. With $\Pr(t|c)$ fixed, $\Pr(c|e)$is given by the topic model, and $\Pr(e)$ as total frequency of query containing e, all component probabilities were computed and indexed for all entities and classes. In online prediction step, given query q is exhaustively segmented to entities and contexts, and the segmentation with highest $\Pr(e, t, c)$ is the result of the prediction.

In contrast, Pennacchiotti described in [30] an unsupervised technique for QNER in an open-domain fashions, where the entity classes are automatically discovered. The first phase - Entity Extraction - mines entity candidates from the search log based on heuristic and assumption that users oftentimes construct the search query by copying them from existing texts. In second phrase, Clustering by Committee clustering algorithm was used to support the scale of open domain. In the algorithm, remainder entity in search queries and click through data are used as features.

## 2.7 Measuring search relevance

In the literature, popular measure of relevance is the combination of recall and precision.

Precision is defined as the fraction of relevance instances among the retrieved instances. Recall is the fraction of the retrieved instances that are relevant to the total amount of relevance instances. In the information retrieval context, precision and recall are defined mathematically as follow:

$$precision = \frac{|\{relevant\ documents\} \cap |\{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

$$recall = \frac{|\{relevant\ documents\} \cap |\{retrieved\ documents\}|}{|\{relevants\ documents\}|}$$

Precision indicates the usefulness of the result set, while recall indicates completeness of the result set. For a concrete example, suppose the total number of documents relevant to the search query “blue office shirts” is 100. If a search engine returns 50 results, among which 30 documents are actually relevant, the precision is 60% (30/50), and the recall is 30% (30/100). This means only 60% of search results is useful to user, and the search result is not complete.

In practice, there exists a tradeoff between precision and recall. Increase one score is often done at the cost of decreasing the other. Achieving recall score of 1.0 is trivial by simply returning all documents. However, the precision would suffer as only small fraction of the retrieved document is relevant. In contrast, search engines may aggressively filter out less relevant document, returning few documents that is the most relevant. Although this conservative approach results in increased precision, recall will be low as small fraction of the relevant documents will be surfaced.

Because of the intrinsic tradeoff between recall and relation, the two indicators are rarely discussion in isolation. Instead, both are combined into single score, or the values of a measure are compared when another value is fixed. An example of combinational measure is the weighted hormonic mean of recall and precision also referred to as F-measure.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Building perfect search engines that offer 100% precision and recall is impractical. In practice, by employing different query understanding methods introducing in the

subsequent sections, a balance tradeoff between precision and recall can be achieved, resulting in high quality search system.

# 3 Implementing QNER enhanced ecommerce search

## 3.1 System requirements

### 3.1.1 Functional requirements

The goal of the project is to implement a fully functional full text search solution for ecommerce enhanced with query understanding capability with Named Entity Recognition at its core.

The first part of the objective is working full text search engine for ecommerce. Concretely, structured ecommerce documents or products must be indexed and made retrievable with free-text query. The precision and recall of the retrieval system should be reasonably good. Documents should be returned in the order of relevance. Search query with typo should be tolerable and relevant search result would still be delivered.

The second part of the goal is to enhance the search engine with query understanding capability. Concretely, Named Entity Recognition will be implemented. In addition, the major application of NER will be implemented and integrated, including query scoping, query relaxation, and query expansion. The applications of NER will allow the fine tuning of precision and recall.

The full-text search operation is exposed via an HTTP endpoint. In addition, there will be an UI to make search request and render the search result page for testing purpose.

### 3.1.2 Technical requirements

The search system should be suitable for small to medium scale ecommerce business having hundreds of thousands of products in its catalog.

The system should be performant, highly available, and horizontally scalable. The reason is that search infrastructure plays a critical role in the enabling various application in ecommerce domain beyond search, such as recommendation, personalization.

This thesis aims at building a search system that can easily achieve the throughput of at least 100rps (requests per second) at 100ms p95 latency in a single node setup.

## 3.2 Ecommerce QNER methodology

In this section, a practical method for conducting QNER is presented. The primary goal is to have QNER execution model that is extensible. As seen in the literature, QNER can be conducted in many different approaches, ranging from semi-supervised, to supervised, to unsupervised techniques. However, the common patterns as seen in research such can be factored out. In the literature, such as [11] [26] , Named Entity Recognition task is tackled in two separate phases: query segmentation, and segment classification. The output of the first phase is single query segmentation. This is unnecessarily strict in the context of ecommerce search: if one of the alternative query segmentations is considered more effective, it will be selected. Similarity, we allow for multiple entity classes per query segments. With this flexibility, we need a way to pick up the best NER. The concrete detail is discussed in the following paragraphs.

The sequence of query keywords that may refer to an entity instance of any entity classes is defined as a candidate named entity.

The framework proposed for NER for ecommerce search query is consisted of following phases:

1. Query segmentation of queries to split query into segments where each represent a candidate named entity. There can be multiple segmentations produced by this step.
2. Segment classification assigns multiple weighted class labels to each query segments for segmentations produced in the first phase.
3. Scoring phase identifies the best query segmentation and best class labels for each segment of the segmentation.

Query segmentation tasks aim to identify the most likely semantic units or phrase – a sequence of keywords that means one thing – in the search query. For example, segmentation of query “Metropolia University principal lecturers” should give two phrases: “[Metropolia University] [principal lecturers]”. In NER, query segmentation is used to identify the segment boundary where named entities might occur.

Formally, query segmentation task is defined as follows. Given query Q consisted of $n$ keywords $w_1, w_2, \dots, w_n$ in the original order as submitted by user, the set of all possible segmentations is $S(Q) = \{S_1, S_2, \dots, S_m\}$, where each segmentation $S_i \in S(Q)$ is a sequence of disjunct segments $s$, each a contiguous subsequence of $q$, whose concatenation yield $q$. The query segmentation phase output most probably segmentations $S_p(Q) = \{S_1, S_2, \dots, S_t\}$, where $S_p(Q) \subseteq S(Q)$. There are $2^{n-1}$ possible segmentation for $n$ terms query, giving upper bound $t \leq m \leq 2^{n-1}$.

Segment classification assigns one or more weighted class labels to each segment $s_{i,j}$ of all segmentation in $S_p(Q)$. For instance, for segmented search query “[harry potter] [walkthrough]”, label *GAME*, *MOVIE*, and *BOOK* can be assigned to the first segment. Based on the remainder of the search query, smart algorithm would give *GAME* more weight. It is normal if segment cannot be classified into predefined set of classes. In this case, it can be assigned to special class *UNKNOWN*.

Scoring phase score multiple query segmentations having segments assigned to one or more entity classes. Output of the scoring phase is the segmentation with highest score. In the literature context, QNER would often be conducted in only two phases: named entity detection (query segmentation) that output single best segmentations and classification that give single best classification per segments. However, in practical engineering context, it is simple to validate different query segmentations by evaluating it against the search engine to quantify the effectiveness of the interpretation. To make the overall system more flexible by allowing multiple segmentation and multiple classifications, this step is necessary to identify the optimal named entity interpretation of given search query.

As presented in the literature, there are multiple approaches to implement each phase of the proposed NER framework, ranging handcrafted rules, to pure statistical technique, to highly sophisticated ones employing semi-supervised to unsupervised methods.

To keep it simple, this project implements the three phases of NER as described in the following paragraphs.

In the first phases, all possible n-gram segmentations of the submitted search query are returned. The maximum number of numbers of query token, $n$, is configurable.

In the second phase, the statistic about each query segment is retrieved to assign one or more weighted class labels to it. Concretely, for each segment, the classes in which the segment appears as an instance are used as class labels, and the frequency of the occurrence is used as the weight. For instance, if query segment “adidas performance” exists as brand in 30 documents, it is assigned label BRAND with weight of 30.

In the last phase, all classified segmentations are given score and best interpretation will be chosen. In the implementation, query builder component is used to convert NER interpretation to actual low-level search engine query to get the number matching documents $d_S$. The numbers of matching documents are then used to derive the best NER using the following formula:

$$score(S) = (\sum_{s \in S} |s|^{|s|} \cdot freq(s)) \cdot d_S$$

The scoring formula is the variation of the scoring function for naïve query segmentation as presented by Hagen in [23]. Concretely, all segments will contribute to the final score, and the number of documents matching the QNER interpretation directly influence the score. It is noteworthy that the length of the query segment has significant contribution to the score with the factor $|s|^{|s|}$. The exponential weight is meant to compensate for the power law distribution of the query n-gram.

## 3.3 System architecture

From high level point of view, the search application is composed of a stateless search application that provides search API and implements business logics and algorithms on top of the underlying search engine. To keep it simple, the search service is used by directly the customer facing application. The main components of the search architecture is described in c4 diagram in figure 1.

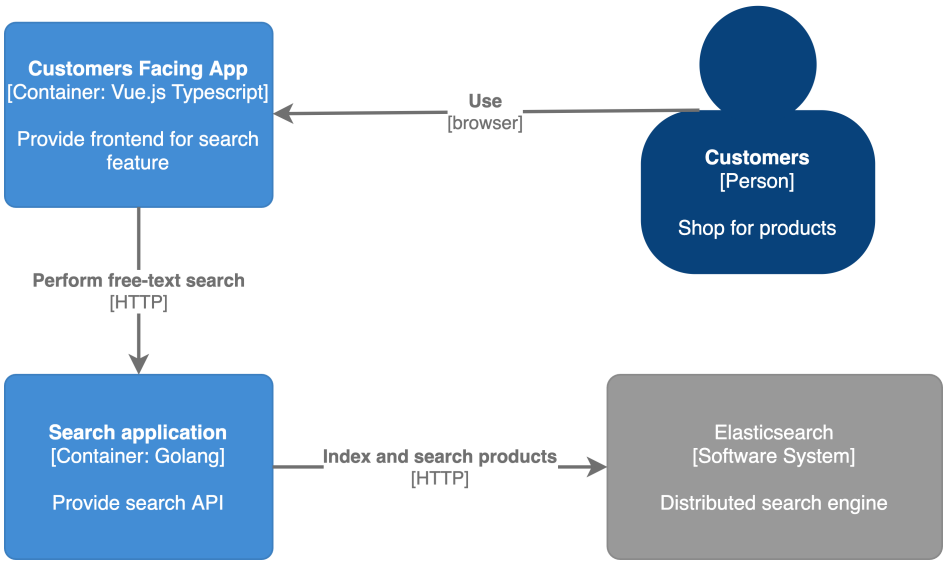


Figure 1. End-to-End high-level architecture of the search application

In the following sub sections, technical decisions regarding the Search Engine and the Search Application are documented. More importantly, the three-phase QNER approach used to implement the search system enhanced with query understanding is also presented.

### 3.3.1 Search engine

At the heart of any ecommerce search system is the search engine. Basically, search engines indexes documents in an efficient manner for supported data retrieval operations.

Functionally, the following support is desired across various phases that searchable documents goes through from data source to the search results:

- Content awareness: robust support for push of documents via API.
- Content processing: support for content normalizations at different level such as characters, tokens. Normalization technique might be stemming, lemmatization, stop words, synonym expansion.

- Indexing: fine-grain control over indexing of document fields to optimize use-case specific data access patterns; support for zero down time re-indexing of data.
- Query processing: support full-text query, term query, and structured query.

Operationally, the chosen search engine must offer high performance, availability and scalability.

The candidates for Enterprise search include both open source and proprietary solutions. Popular open source software offerings are Elasticsearch, Apache Solr, Sphinx, Terrier. Notable proprietary offerings for enterprise search are AWS CloudSearch, Algolia, and SearchSpring. In general, proprietary offerings provide ready-made search capabilities such as spell checking, named entity recognition, and faceted search. Because of the limited extensibility, the cost, and limited value in learning a close-source software solution, proprietary software will not be used. Among open source search engines, Elasticsearch and Apache Solr stands out as the most prominent. They are both created with an open source search engine library named Apache Lucene. Elasticsearch is the emerging winner that is gaining strong interest within the community as it offers superior distributed architecture and ease of use in comparison to Solr.

In the wild, Elasticsearch is used as the underlying search engine powering complex features and requirements. Elasticsearch offers simple RESTful API for indexing and searching of document. For ingestion of searchable documents, there are many composable out-of-the-box components such as character filters, tokenizer, and token filters. Those components that transform full text in different ways at characters and tokens level are combined into analyzers, such as language analyzers, which play a crucial role in processing to increase recall. Elasticsearch provides powerful search feature such as phrase search, faceted search, spelling correction, auto suggest.

Elasticsearch is a distributed, highly scalable, highly available full-text search engine. In high availability setup, Elasticsearch cluster is composed of multiple nodes or servers that participate in the cluster's indexing and search capabilities. Data is organized into shards replicated into multiple replicas shards. The replications and rebalancing of both primary and replicas shards across multiple nodes in the cluster is managed transparently and automatically. Search query is executed in parallel across all the shards, bringing exceptional scalability and performance. In figure 1, an Elasticsearch cluster setup with three nodes, six shards, one replica is shown.


Figure 2. Three nodes elastic search cluster with six shards and one replica

### 3.3.2 Programming language

For this project, Golang programming language is used as it offers good development experience, performance, and powerful concurrency programming model.

Golang code is maintainable, reliable and fun to develop. Golang is statically typed language that compile instantaneously into executable binary across platforms. Golang is very simple programming language with small set of language keywords, well-engineered garbage collection, and simple scoping rules. The language has powerful standard library that is extensive and well designed. It is backed by rich ecosystem maintained by strong open source community. Last but not least, it has modern package management solution that greatly ease the developer workflow.

Golang shines with first class support for concurrency. Primitive such as goroutine and channel makes programming concurrency easy and intuitive while efficiently capitalizing on the multithreading capability of modern hardware in a transparent manner. Designed for usage of multicore computers, and being compiled directly into machine code, Golang can power web system to achieve remarkable performance and concurrency.

Search is critical infrastructure of any ecommerce businesses. The goal is to build a search system that is maintainable yet offer high performance to support business growth for years. Because of its offering, Golang is a great fit for this project.

### 3.3.3 QNER enhanced Search architecture

The technical implementation the search application is simple. It is a Golang HTTP server which implements three-phased QNER on top of Elasticsearch engine.

In high level, the behavior of the QNER component is as follows: given a search query, it gives back single Named Entity interpretation, which is the NER classification and segmentation that is most effective and best reflect user's intent. Three-phase QNER system implements this behavior and is composed of three sub-components that implement segmentation, classification, and scoring logics. Main interfaces of the application materialize in listing 1.

```
type NEREngine interface {
     Analyze(query string) (QNER, error)
}

type threePhasesNEREngine struct {
     classifier NERClassifier
     segmenter  NERSegmenter
     scorer     NERScorer
}

type NERSegmenter interface {
     Segment(query string) ([]Segmentation, error)
}

type NERClassifier interface {
     Classify(segmentation []Segmentation) ([]QNER, error)
}

type NERScorer interface {
     Score(interpretations []QNER) ([]ScoredQNER, error)
}
```

Listing 1. Interfaces of NER engine and composition of three-phase NER system.

To enable search functionality of the ecommerce system, NER interpretation is then converted to a structured search engine query and is executed against search engine to give back search result. These steps are represented as high level interfaces in listing 2.

```
type QueryBuilder interface {
     Build(query QNER) (interface{}, error)
}

type QueryEngine interface {
     Query(qry interface{}, pagination Pagination) (Result, error)
}
```

Listing 2. A Python subroutine that outputs information about objects in possession of a player.

Finally, a search service composed of NEREngine, QueryBuilder, and QueryEngine will delivery search result to the search requests. This service composition is illustrated in listing 3.

```
type Service interface {
     Search(query string, paging Pagination) (Result, error)
}

type nerSearch struct {
     nerEngine    NEREngine
     queryBuilder QueryBuilder
     queryEngine  QueryEngine
}

func (s *nerSearch) Search(query string, paging Pagination) (Result, error) {

     qner, err := s.nerEngine.Analyze(query)
     // err handling

     query, err := s.queryBuilder.Build(qner)
     // err handling

     res, err := s.queryEngine.Query(query, paging)
     // err handling

     return res, nil
}
```

Listing 3. Search service interface and implementation

The use of interfaces enables different implementations for each of the processes to be composed and plugged in in different way. For instance, two different QNER scoring implementation can be composed in to one, enabling AB test to benchmark the effectiveness of the different solutions. This enable great flexibility for the whole system. Implementation section will describe how each of the interface will be implemented on top of the Elasticsearch engine.

## 3.4 Implementation

In this section, implementation of the ecommerce search system powered by NER engine on top of Elasticsearch is described.

### 3.4.1 Search indexing

The first challenge is to index structured product data from eCommerce domain into Elasticsearch to facilitate the NER process. Regardless of the data source and

eCommerce domain, the searchable documents must provide three data fields to enable the QNER feature. The first field is named *_props*, which contains an array of properties object with field *key* indicating the property class and field *val* for the property value. An example of property key is Color and value is Blue. The other required fields are *fulltext_search* and *fulltext_search_boosted*, which contain the free text relevant to the documents and will be searched against query segments with no classes label.

Documents providing the specified three fields required for NER feature are then indexed into Elasticsearch with the indexing mapping in listing 4. It is worth noting that _props field are indexed as nested property. The property keys are indexed as keyword since it is not used for full text matching but for aggregation and filtering purpose. The property keys and full-text search fields are indexed as text with english analyzer to benefit from the advanced language specific text analysis provided by Elasticsearch.

```
{
  "properties": {
    "_props": {
      "type": "nested",
      "properties": {
        "key": {
          "type": "keyword"
        },
        "val": {
          "type": "text",
          "analyzer": "english",
          "index_phrases": "true"
        }
      }
    },
    "fulltext_search": {
      "type": "text",
      "analyzer": "english"
    },
    "fulltext_search_boosted": {
      "type": "text",
      "analyzer": "english",
      "boost": 4
    }
  }
}
```

Listing 4. Search service interface and implementation

### 3.4.2 QNER on top of Elasticsearch

In this section, three phase QNER implementation using Elasticsearch is described.

Similar to the work by Hagen [23], the implemented NER segmentation produces n-gram segmentations of given query. In other words, it outputs all possible segmentations where length of each segment is at most the configured $n$ words. It requires a tokenizer that split the search query into consecutive words. The simple tokenizer implementation where the search query is split by empty spaces are used.

QNER classification is implemented as follow. First, statistic for each query segments are retrieved from Elastic Search. For each query segment, the number of documents having entity instances per class are retrieved. Then, the overall score is given for each possible class labels of query segment are given based on the statistics.

In QNER classification process, the Elasticsearch query employed utilize analytics feature of the search engine. Concretely, aggregation functions are composed to give the count of matching documents per entity class that are associated with a requested entity instance. For instance, given the phrase “adidas performance”, the query will return statistic that are 90 documents with the value as brands, and 24 documents having it as names. The actual query instance is documented in listing 4, and the construction of the query in described in the next paragraph.

The Elasticsearch query used for QNER classification process is composed of three nested aggregation functions. The top-level nested aggregation enables aggregating on nested documents. The nested documents, referred to as property documents, are properties of the searchable document, each containing information about the property class, and property instance/value. Inside this aggregation is the filters aggregation, which define multiple buckets, each collecting all property documents that match given filter. The filter used is phrase query for each query segments of the given search query. For each bucket of properties document, term aggregation is used to create multi-bucket of property class and count the entity instances in each sub-bucket.

```
{
 "aggregations": {
  "spec_objs": {
   "nested": {"path": "_props"},
   "aggregations": {
    "filtered_spec_objs": {
     "filters": {
      "filters": [
       {"match_phrase": {"_props.val":
          {"operator": "AND","query": "Eco Friendly"}}},
       {"match_phrase": {"_props.val":
          {"operator": "AND", "query": "Adidas Performance"}}},
      ]
```

```
    },
    "aggregations": {
     "spec_key": {
      "terms": {"field": "_props.key"}}}}}}},
 "size": 0
}
```

Listing 5. Search service interface and implementation

In the last QNER phase, the scorer is implemented by scoring each QNER interpretations produced by the classification steps. It uses the query builder component to convert QNER interpretations to search engine queries. Using Elasticsearch filters aggregation, the number of matching documents for each QNER interpretation is retrieved. This is the input for the scoring function described in the Architecture section.

It is not always possible to classify a query segment into as any of the existent search properties. In such case, the query segment is classified as UNKNOWN, a special property class.

### 3.4.3 QNER powered Search

QNER powered search engines take the output of the QNER process and convert it to search engine result. This section discusses the conversion of QNER interpretation to search result.

The query builder component takes the QNER interpretation as the input, which are multiple query segments, each labelled as one or more properties of the searchable documents. Since supporting query relaxation is not in scope of this project, all query segments are required to match the documents. Furthermore, since each query segment may have one or more property type labels, the query segment must match the instance of one of the property classes. Concretely, for interpretation “[a]X|Y [b]Z”, which classify “search query” into segment a, an instance of property class X or Y, and segment b – instance of class Z, the resulting Elastic Search query used to get search result is presented in the listing 6.

```
{
  "query": {
    "bool": {
      "must": [
        {
          "nested": {
```

```
            "path": "_props",
            "query": {
              "bool": {
                "must": [{
                  "match_phrase": {"_props.val": "a"}}],
                "should": [
                  {"term": {"_props.key": "X"}},
                  {"term": {"_props.key": "Y"}}],
                "minimum_should_match": 1}}}},
        {
          "nested": {
            "path": "_props",
            "query": {
              "bool": {
                "must": [ {
                  "match_phrase": {"_props.val": "b"}}],
                "should": [
                  {"term": {"_props.key": "Z"}}],
                "minimum_should_match": 1}}}}]}}
}
```

Listing 6. Search service interface and implementation

In listing 6, a Boolean query is used to compose a boolean condition for multiple queries. It's must clause has multiple sub queries; each specifies the matching condition for labelled query segments. For sub queries, nested query is used to query nested property documents stored at "_props". The query used in nested query requires that the query segment must phrase-match as property value, and one of the segment labels must match as the property keys.

If the class labels are missing for a query segment, the query segment will be matched against the *fulltext_search* and *fulltext_search_boosted* fields of the indexed documents.

### 3.4.4 Search front-end

To simplify the testing and evaluation process, a client webservice is developed in Vue.js, a modern, open-source model–view–viewmodel JavaScript framework. The frontend is developed in Typescript to take advantage of static typing.

A simple user interface is implemented with a search input and a search button. The product data can be inspected further in JSON format with a pop-up model when the product is clicked. Paging functionality is also included.

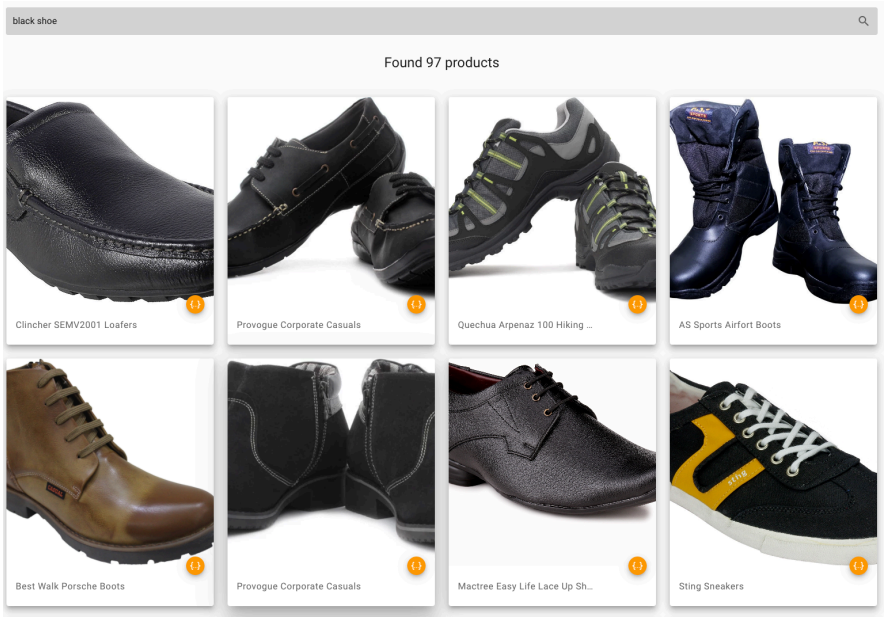


Figure 3. Three nodes elastic search cluster with six shards and one replica

The search frontend enables the evaluation of

## 4 Evaluation

To test the solution, the Flipkart products data set containing 20k products was obtained from Kaggle website. The dataset was pre-crawled from the website of Indian ecommerce giant Flipkart. The data set contains rich product features, such as brand, name, description, images, category tree, price, ratings, and rich property sets.

Before products data can be searched, it is processed to provide data fields required for QNER feature and then indexed into Elasticsearch. The following product properties will be made available for NER classification: *PRODUCT_TYPE, USER_GROUP, OCCASION, COLOR, FABRIC, MATERIAL, BRAND, UID* (Unique Identifier). Field *fulltext_search_boosted* is generated with product name, brand, and unique identifier; the *fulltext_search* field incorporates product description and all product properties.

## 4.1 Precision and recall

Measurements of precision and recall are conducted for 60 search queries that contains at least one named entity in the predefined classes. Search queries are divided into three types according to [11]:

- Focused query (FQ) that is the target candidate named entity itself.
- Very focused query (VFQ) that contains single candidate named entity with one or more query keywords. Additional keywords are used to further refine the search result and cannot be classified to any entity classes.
- Unfocused query (UF) that contains more than one named entity and optionally have the refiner keywords.

For each group of search query, three assessments are conducted:

- QNER accuracy. Is the QNER result expected? The metric shows the percentage of correctly identified QNER produced by the search system.
- Precision of the search result, which is the fraction of the relevant documents among the retrieved documents. The metrics represent the average precision of the retrieval for search queries in the group.
- Recall of the search result, which is the fraction of the relevant documents that was actually retrieved.

Input search queries are manually annotated with Named Entity by human. The manual NER annotation is then compared with the system output. This enables the calculation of QNER accuracy metric. To evaluate the precision and recall metrics of the retrieval, comparisons between the actual result set and the expected result set must be made. Author identifies the actual result set by evaluating search result where documents must match at least one of the search keywords. It is noteworthy that this result set compromises precision to maximize recall.

The result is presented in table 1:

Table 1. QNER precision and recall evaluation

| Query type | QNER accuracy | Precision | Recall |
|---|---|---|---|
| FQ | 98% | 97% | 94% |
| VFQ | 80% | 72% | 76% |
| UQ | 96% | 99% | 95% |
| All | **92%** | **89%** | **88%** |

## 4.2 Performance test result

To perform the performance test, Locust – a modern load testing tool is used. Locust can generate distributed and scalable load. It is capable of simulating millions of simultaneous users. Locust test is easy to write in Python code.

The test was conducted in single local machine (an iMac with 3.4 GHz Quad-Core Intel Core i5 CPU and 8GB 1600MHz DDR3 memory) with single container instance each for Elasticsearch and the search application. Elasticsearch container had resource limit of 2G memory and 2 CPUs. The application had resource limit of 512Mb memory and 1CPU.

On this infrastructure, a simulation was run in which 800 concurrent users searched for products with complex three term queries. The hatch rate was 5 new users/seconds. As seen in figure 3, the load gradually reached 115rps and stayed stable. The error rate was 0%.

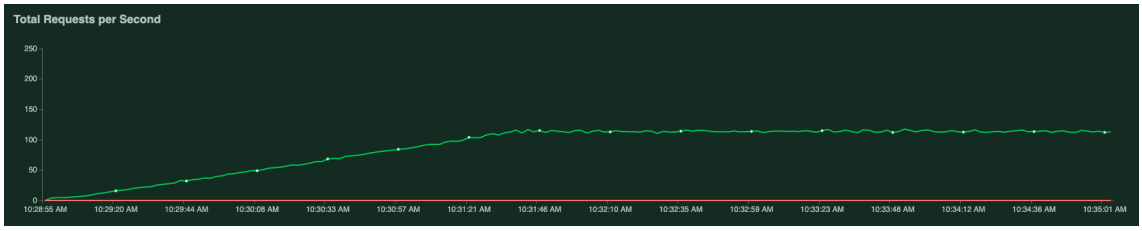


Figure 4. Search throughput as the number of concurrent users increase from 0 to 800

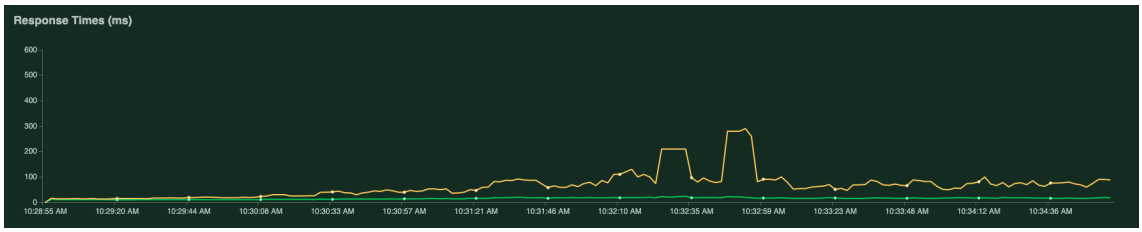


Figure 5. Search latency as the number of concurrent users increase from 0 to 800

As seen in figure 4, the p95 latency of the system stayed around 80ms and well under 100ms except during two brief latency spikes. In addition, when the load is under 60rps, the p95 latency is only around 40ms. This performance was achieved without any caching mechanism in place.

# 5 Discussion

In this section, the QNER powered search engine implementation is evaluated against the initial requirements. The evaluation is based on the correctness assessment and performance test result in the previous section. In addition, this section also describes the future improvement to this project.

## 5.1 QNER evaluation

Overall, the three phase QNER implementation correctly identifies and classifies the named entity for the majority of cases. From the assessment of the QNER powered search, it is found that the current implementation works well if all query segments can be classified as one of the known property types. As a concrete example, search query “adidas sport sandal” undergoes three phases query as described in table 2 and correctly outputs the interpretation “[adidas]Brand [sport sandal]ProducType”.

Table 2. Example of QNER process

| Segmentation | Classification | Score |
|---|---|---|
| [adidas] [sport] [sandal] | [adidas]Brand [sport]Occasion [sandal]ProductType | 3.0 |

| | | |
|---|---|---|
| [adidas sport] [sandal] | [adidas sport]Unknown [sandal]ProductType | 1.04 |
| [adidas] [sport sandal] | [adidas]Brand [sport sandal]ProductType | 5.0 |
| [adidas sport sandal] | [adidas sport sandal]Unknown | 0.27 |

The algorithm implemented fails for case where the identified class label for a query segment does not combine with others classified query segments. For instance, for query *"adidas car"*, the query segmentation would produce the NER "[adidas]Brand [car]ProducType". This gives 0 search result for the query. To address this limitation, the classification step might tag the query segment as Unknown in addition to the matching labels, giving more QNERs for the scoring phase.

## 5.2 System evaluation

Overall, the resulting system is well-built and fulfill the operational objectives of being scalable and performant.

The resulting search system is highly scalable. Because the search application is stateless, the scalability bottle neck would be in the underlying search backend - Elasticsearch. However, Elasticsearch is designed from scratch to be distributed and massively scalable.

Generally, system designers aim for maximum throughput with acceptable latency. The performance test result indicates that this goal is achieved. The system responds under 100ms latency, which is the threshold under which users perceive the interactions as instantaneous. In addition, the single instance throughput is high at 115rps, which translates to 10M requests per days. This is achieved without any caching mechanism implemented. The performance test result conducted for 20K products and single Elasticsearch node might not transfer to huge collection of products (hundreds of thousands to millions) where a multi-node setup is needed.

It is noteworthy that the resulting search system behaves in eventually consistent manner. According to CAP theorem, only two out of three properties of distributed system can be supported: partition tolerance, consistency, availability. As networks are not reliable, partition tolerance must be supported, or otherwise a single network hiccup will

bring whole offline. Since the system is designed to be scalable, it cannot support strong data consistency where every read returns the most recent write or an error. This is clearly acceptable for search use case. It is of no harm if users are potentially exposed to out-of-stock/deleted products via search.

## 5.3 Next steps

At this point, a baseline implementation of the QNER enhanced search engine is in place. The current implementation can work with any structured eCommerce product data and offer great performance and scalability. The baseline implementation does not implement any sophisticated QNER approaches as seen in the literature, but it encapsulates well the execution steps of characteristic of the query understanding task. The QNER is exploited to implement query scoping, where query segments are only matched against sensible properties of the documents.

Current state of the project may be evolved in three main directions: to improve existing QNER process, to leverage QNER to implement other tasks of query understanding, to enhance current search system with more essential search features.

There is plenty of room for improvement for existing three-phases QNER process. For each phase – segmentation, classification, QNER scoring, new algorithms can be implemented and evaluated against the base line implementation. For instance, the naïve query segmentation algorithm proposed by Hagen et al. for can be implemented for first phase. For query classification phase, the context (remainder of the query) of the query segment should be taken into account to improve precision of the classification.

Many practical tasks of query understanding beyond query scoping can be implemented thanks to QNER. For example, query relaxation can be implemented by optionalize query segments tagged with classes that deem inessential to the user intent. Similarly, query expansion can be implemented by firstly building a knowledge graph of entities with synonyms. For any entity present in search query, its synonyms can be used to match more documents. Query relaxation and expansion tasks will improve the recall of the retrieval; they will complement query scoping feature which improves search precision.

Most importantly, aiming to deploy the current search system into the wild with more essential search features are the most practical direction. Exposing the implemented system to the harsh reality of user input is the best way to understand its value to user and iterate on practical improvements. It will be important to build monitoring and feedback to the search solution so that future improvements to the overall search, or specifically QNER process can be AB tested, enabling data driven decision making. Besides, search suggestion and spellcheck are among essential search features that will greatly improve user experience.

# 6 Conclusion

The objective of the thesis is to build a performant and scalable search application for ecommerce enhanced with Query Named Entity Recognition – an advanced query understanding capability. The thesis proposes a flexible three-phases QNER framework with an implementation on top of leading open-source search engine Elasticsearch. QNER enhances the search system by enabling the implementation of query scoping – a query transformation task. The resulting system is well engineered, and it satisfies the functional and operation objectives put forth. Before the implementation, the background researches in the field of query understanding were meticulously analyzed, especially the task of query transformation, query segmentation, and query named entity recognition. In addition, search query analysis and information retrieval evaluation methods were explained in detail.

The implemented search system has many rooms for improvements. For example, each of the three phases of the QNER process can be improved and tested with different implementations to increase the overall relevance of the search result. Besides, QNER opens the opportunities to implement advanced query understanding techniques such as query relaxation, query expansion. Most practical next step could be to build necessary search features and deploy the search system into the wild.

In summary, this thesis demonstrates how QNER can be applied to improve search. It emphasizes the role of query understanding component in search, which is ab indispensable feature of any ecommerce businesses.

## 7 References


[1] "eCommerce worldwide prediction," 2020. [Online]. Available: https://www.statista.com/outlook/243/100/ecommerce/worldwide. [Accessed 10 June 2020].

[2] Craig Silverstein, Michael Moricz, Hannes Marais, and Monika Henzinger, "Analysis of a very large web search engine query log," *SIGIR Forum,* p. 33(1):6–12, 1999.

[3] Daxin Jiang, Jian Pei, and Hang Li, "Mining search and browse logs for web search: A survey," p. 4(4):57:1–57:37, October 2013.

[4] Croft Michael Bendersky, W. Bruce, "Analysis of long queries in a large scale search log," in *Proceedings of the 2009 Workshop on Web Search Click Data, WSCD '09*, New York, USA, 2009.

[5] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Pla- chouras, and Fabrizio Silvestri, "The impact of caching on search engines," New York, USA, 2007.

[6] Amanda Spink, Dietmar Wolfram, Tefko Saracevic, and Bernard Jansen, "From e-sex to e-commerce: Web search changes," pp. 35(3):107 - 109, April 2002.

[7] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic, "Searching the web: The public and their queries," *J. Am. Soc. Inf. Sci. Technol.,* p. 52(3):226–234, February 2001.

[8] K. R. Milad Shokouhi, "Time-Sensitive Query Auto-Completion," in *35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'12)*, New York.

[9] Umut Ozertem, Olivier Chapelle, Pinar Donmez, and Emre Velipasaoglu, "Learning to Suggest: A Machine Learning Framework for Ranking Query Suggestions," in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'12)*, New York.

[10] A. Broder, "A taxonomy of web search," p. 36(2):3–10, September 2002.

[11] A. M. Alasiry, "Named entity recognition and classification in search queries," *PhD thesis, Birkbeck, University of London,* 2015.

[12] Dou Shen, Rong Pan, Jian-Tao Sun, Jerey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang, "Query enrichment for web-query classification," p. 24(3):320–352.

[13] Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang, "Robust classification of rare queries using web knowledge," New York, USA, 2007.

[14] Silviu Cucerzan, Eric Brill, "Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 2004.

[15] Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich,Vanja Josifovski, Donald Metzler, Lance Riedel, Jeffrey Yuan, "Online Expansion of Rare Queries for Sponsored Search," in *WWW*, 2009.

[16] Jingfang Xu, Gu Xu, "Learning Similarity Function for Rare Queries," in *WSDM '11: Proceedings of the fourth ACM international conference on Web search and data mining*, 2011.

[17] Doug Beeferman, Adam Berger, "Agglomerative clustering of a search engine query log," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, New York, 2000.

[18] Rosie Jones, Benjamin Rey and Omid Madani, and Wiley Greiner, "Generating Query Substitutions," in *Proceedings of the 15th international conference on World Wide Web (WWW'06)*, New York, 2006.

[19] Nish Parikh, Prasad Sriram, and Mohammad Al Hasan, "On segmentation of eCommerce Queries," 2013.

[20] Shane Bergsma, Qin Iris Wang, "Learning noun phrase query segmentation," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, 2007.

[21] Michael Bendersky, W. Bruce Croft, and David A. Smith, "Two-stage query segmentation for information retrieval," in *Proceedings of the 32Nd International ACM SIGIR Confer- ence on Research and Development in Information Retrieval, SIGIR '09*, New York, 2009.

[22] Bin Tan and Fuchun Peng, "Unsupervised query segmentation using generative language models and wikipedia," in *Proceedings of the 17th International Conference on World Wide Web, WWW '08*, New York, 2008.

[23] Matthias Hagen, Martin Potthast, Benno Stein, and Christof Braeutigam, "The power of naive query segmentation," in *Proceedings of the 33rd International ACM SIGIR Confer- ence on Research and Development in Information Retrieval*, New York, 2010.

[24] Satoshi Sekine and Chikashi Nobata, "Definition, dictionaries and tagger for extended named entity hierarchy," *LREC, European Language Resources Association,* 2004.

[25] Andrew McCallum and Wei Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, Stroudsburg, 2003.

[26] Andreas Eiselt, Alejandro Figueroa, "A two-step named entity recognizer for open-domain search queries," in *Proceedings of the Sixth International Joint Conference on BIBLIOGRAPHY 167 Natural Language Processing*, 2013.

[27] M. Pąsca, "Weakly-supervised discovery of named entities using web search queries," in *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, New York, 2007.

[28] Christopher D. Manning, Hinrich Schutze, "Foundations of Statistical Natural Language Processing," MIT Press, Cambridge, 1999.

[29] Jiafeng Guo, Gu Xu, Xueqi Cheng, Hang Li, "Named entity recognition in query," in *Proceedings of the 32rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, York, 2009.

[30] Alpa Jain, Marco Pennacchiotti, "Domain-independent entity extraction from web search query logs," in *Proceedings of the 20th International Conference Companion on World Wide Web*, New York, 2011.