Identifying Web Queries with Question Intent

Gilad Tsur, Yuval Pinter, Idan Szpektor, David Carmel
Yahoo Labs, Haifa 31905, Israel
{giladt, yuvalp, idan, dcarmel}@yahoo-inc.com

ABSTRACT

Vertical selection is the task of predicting relevant verticals for a Web query so as to enrich the Web search results with complementary vertical results. We investigate a novel variant of this task, where the goal is to detect queries with a *question intent*. Specifically, we address queries for which the user would like an answer with a human touch. We call these CQA-intent queries, since answers to them are typically found in community question answering (CQA) sites.

A typical approach in vertical selection is using a vertical's specific language model of relevant queries and computing the query-likelihood for each vertical as a selective criterion. This works quite well for many domains like Shopping, Local and Travel. Yet, we claim that queries with CQA intent are harder to distinguish by modeling content alone, since they cover many different topics. We propose to also take the structure of queries into consideration, reasoning that queries with question intent have quite a different structure than other queries. We present a supervised classification scheme, random forest over word-clusters for variable length texts, which can model the query structure. Our experiments show that it substantially improves classification performance in the CQA-intent selection task compared to content-oriented based classification, especially as query length grows.

Keywords: Question intent, Vertical selection

1. INTRODUCTION

Modern Web search engines often combine search results of specific domains, such as News, Shopping or Travel, with general Web search results. The common approach is of using frames that are aggregated into the main results page, each containing search results from a single domain. Each such frame should be shown only when its domain results are highly relevant to the query. *Vertical Selection* [2] is the task of selecting the subset of the verticals (domains) most relevant to a given query.

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883058 .

Many domains, such as Travel, Shopping and Local, are "content-specific". For such domains, particular keywords in the query provide, often independently, useful signals about the domain's relevance to the query. Examples include '*flight*' and '*hotel*' for Travel, '*camera*' and '*price*' for Shopping and '*restaurant*' and '*address*' for Local. Therefore, much prior work has focused on constructing models that emphasize such domain-specific keywords for the vertical selection task [28, 2, 14, 27].

In this work, we target a domain of a different type, that of questions and answers, for answering queries with a *question intent*. Such question-intent queries constitute ∼10% of the Web queries issued to a search engine [29]. Specifically, we focus on queries for which the user wants an answer with a human touch, such as "*what to do with apples from tree*" and "*red bumps on healing burn*".

In general, such queries, with a complex information need, will not be satisfied by traditional Web search results. We call them queries with *CQA intent*, or simply *CQA queries*, since content for them is typically found on Community-based Question Answering (CQA) Web-sites (*e.g.* Yahoo Answers, StackExchange, Answers.com). These websites, in which users ask any type of question and receive answers from other users (experts or not), are very popular, overall attracting hundreds of millions of page views per month. A CQA vertical can handle CQA queries by providing relevant question-answer pairs extracted from CQA sites. An example of a CQA vertical search result (in the form of a related question and answer) presented on top of the results page for a query with CQA intent can be seen in Fig. 1.

We define our task to be the detection of queries with CQA intent. It is a variant of the vertical selection problem that is reduced to binary selection (CQA vs. non-CQA queries). Further refining the task, we are interested in correctly classifying queries prior to calling the vertical search, and not as a post-retrieval analysis. The reason is that pre-retrieval vertical selection can substantially reduce the load on the vertical search engine: instead of all queries hitting the general search engine being sent to it, only those that might truly benefit from the target domain are. Specifically, in the case of questions with CQA intent, according to the statistics above, optimal detection of such queries could result in 90% reduction in load on the CQA vertical with no performance drop.

The main difference between detecting queries with CQA intent and detecting queries with content-specific vertical intents lies in the type of input queries. As discussed above, queries that target many typical vertical domains contain

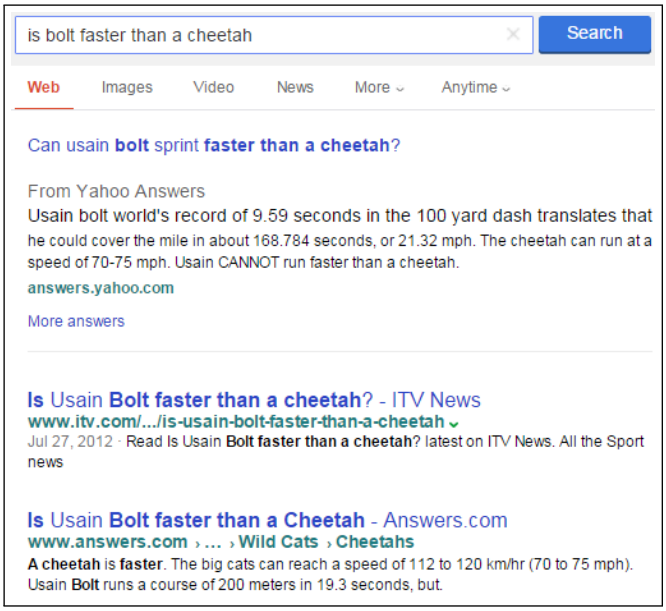

Figure 1: An example of a CQA vertical search result (a related question and human-written answer), presented on top of the results page for a query with question intent.

specific content words that hint towards the relevance of this domain. However, CQA queries address all kinds of content, including technology, health, gardening and style, to name a few. On the other hand, CQA queries have a rather unique structure compared to other Web queries. These queries express a specific question or situation, *e.g.* "*which is better DSLR entry level or semi professional cameras*" (as opposed to topical queries such as '*DSLR camera*'). As a result, they are longer (6-7 terms on average) than the average Web query [8] and they tend to contain complex syntactic structures. This property notwithstanding, only ∼30% of them are formulated as full natural language questions [29]. For example, from our analysis of thousands of CQA queries (*i.e.*, Web queries landing on CQA sites), only 40% of them start with a question word such as '*why*', '*how*', '*is*' or '*should*'.

Consequently, for the task of detecting queries with CQA intent, we propose to model the structure of a query in addition to modeling typical content for this domain. To this end, we introduce several ways to incorporate structure-oriented signals both for language models and for linear classifiers. These signals include the exact position of each word in the query, capturing rigid query structure, the part-of-speech of each word, representing the syntax of the query, and word-clusters, which provide high level semantics and structure for the query. Moreover, we present a random forest classifier over variable length texts, which makes use of word-clusters for representing the input text and also considers the output of a baseline linear classifier. We detail these models in Sections 3 and 4.

To show the difference between intent detection for CQA and other domains, we evaluated both our models on two datasets of Web queries, one for the CQA detection task and one for the Shopping domain detection task. On both datasets we compared our models to the language model approach that is typically used for vertical selection, as well as to a linear classifier over the query terms. Our results show that modeling query structure only marginally helps for the Shopping domain. Yet, in the CQA domain, the differences are vast. Specifically, our random forest model substantially outperforms all other models as query length increases.

Our key contributions are:

1. A novel variant of the vertical selection task: detecting queries with CQA intent;
2. Methods for utilization of query structure signals within language models and within supervised classification;
3. A supervised classifier based on random forest over variable length texts, using word-clusters for input text representation.

2. RELATED WORK

Identifying the search intent of a user query is a long-standing research goal in IR that is generally treated as a classification scheme. Some popular approaches predict the search intent by mapping the Web query into a predefined taxonomy of observed search intents [6, 26, 15]. Others map the search intent into specific domains, or verticals. The search intent is determined based on the subset of verticals which will most likely satisfy the user's need [28, 2, 1, 3, 11].

Typical intent classification methods for vertical selection rely mostly on evidence found in the vertical content (*e.g.* [27, 11]), or the query string, from which features are derived [2, 1, 12]. Other works focused on intent classification using click-through data. Li et al. [19] increased the amount of training data obtained for semi-supervised learning by click graphs. Specifically, they inferred class membership of unlabeled queries from those of labeled ones based on their proximity in the click graph. Hu et al. [14] leveraged Wikipedia for the intent classification task. The intent of any input query is identified through mapping the query into the Wikipedia representation space, spanned by Wikipedia articles and categories.

Arguello et al. [2, 1], expanded the classification scheme for vertical selection by combining signals from the vertical content, the query string, and queries previously issued to the vertical (the vertical's query log [2], or general Web queries that resulted in a click on the vertical pages[1]). The click-through data is used to construct a descriptive language model for each vertical's related queries, and a query's likelihood is computed and used to select the most relevant verticals. Arguello et al. [3] proposed to reuse training data from a set of operating verticals, which already have significant traffic, in order to learn a predictive intent model for new unrecognized verticals with no landing traffic. Similarly to our work, Diaz [12] predicted the query intent for News articles. He trained a binary classifier that distinguishes between newsworthy and non-newsworthy queries, using queries annotated manually to be deserving or not deserving of a news display.

Zhou et al. [31] presented a formal analysis and a set of extensive user studies to show that in addition to the vertical's retrieved content, vertical relevance heavily depends on the user having prior intent for content from a given vertical, or how oriented each vertical is to the user's information need. Similarly, our work also focuses on identifying

the user's search intent based on prior analysis of the user query, without post-analysis of retrieved content from the vertical.

White et al. [29] analyzed the intent of Web queries. Any query beginning with a question indicator or ending with a question mark was considered to be a question query. They showed that about 10.3% of queries issued to a search engine have question intent. However, only 3.2% are formulated as natural language questions.

Several previous works analyzed CQA queries as part of their study [21, 22, 8]. Liu et al. [21] defined a CQA query as a query that results in a visit to a CQA site. They predicted the satisfaction of searchers who submit a CQA query by analyzing a large number of such searches and identifying unique characteristics of searcher satisfaction, namely, the effects of query clarity, query-to-question match, and answer quality. In successive work, Liu et al. [22] analyzed the characteristics of Web queries that precede posting a question on a CQA cite. Lately, Carmel et al. [8] analyzed a large set of CQA queries that landed on Yahoo Answers pages. By tagging the title of each clicked page for parts of speech and syntactically parsing it, they noticed significant differences in the probability of a title term to appear in the landing query given the term's part-of-speech tag, and similarly given its syntactic role in the title's dependency parse tree. This analysis was used for improving query term weighting for retrieval. We differ from their work by analyzing the structure of the CQA query itself, with the goal being question intent classification rather than term weighting.

There is also a line of research that analyzes query structure using linguistic tools, mostly for query analysis [4, 18] and for improving retrieval [17, 30]. To the best of our knowledge, there is no prior work that linguistically analyzes Web queries for question intent detection.

3. INTENT MODELS

We next turn to methods for addressing query structure when modeling domains in the vertical selection task. Prior works on vertical selection typically modeled a domain by looking at terms that are common in queries that hit Web pages from the target domain. We first describe two such models, which we refer to as *baselines*: language models and supervised classification (Sec. 3.1). We then discuss ways to revise and extend these baseline models to incorporate signals for query structure as well (Sec. 3.2). In Sec. 4, we will present a classification scheme that makes a different use of such signals.

3.1 Baselines

3.1.1 Language Model

A common approach for modeling a vertical domain is to construct a language model over the queries that result in clicks on Web sites from this domain, extracted from a search-engine query log [19]. For example, the Shopping domain can be modeled with queries that are followed by clicks on links from *Amazon* and *eBay*.

Following, as a first baseline, we trained a trigram language model with Kneser-Ney smoothing [16] for a target domain D, denoted LM_D, using the BerkeleyLM[1] package. Input queries for the construction of the language model were extracted from the query log of a commercial search engine. The queries were selected by randomly sampling from all queries that led to clicks on a manually constructed list of websites that are good representatives of the domain (see Sec. 5.1 for a detailed description of our data-set construction). In addition, we constructed a similar trigram language model for general Web queries, denoted LM_G, by randomly sampling queries from the overall query log as a training set.

For the classification task of whether an incoming query q has intent for domain D, we compute the score

$$s_{LM}(q, D) = LM_D(q) - LM_G(q),$$

where $LM_X(t)$ represents the log-probability that the language model for domain X estimates for a text t. The higher the score, the more confident we are that the query has intent for D. We note that since the raw language model scores are log-probabilities, score subtraction is akin to computing the probability ratio for the query between the two language models.

The procedure described above for generating the training set is used in all the models that are presented in this paper. We note that this procedure is noisy. Landing queries for websites that are strongly related to D are taken as positive examples for the vertical selection task. Still, anecdotal landing queries may be irrelevant, for example due to user exploration. Similarly, randomly sampling queries from the query log as negative examples for a target domain would obviously contain some queries that are relevant to this domain. However, most vertical domains are targeted by only a small percentage of the queries in the log. Therefore, the vast majority of sampled queries are true negative examples. On the other hand, since our models are statistical in nature, and are learned over hundreds of thousands of examples, we expect that the relatively small amount of noise in the training set will not hurt their performance significantly considering the large training set size.

3.1.2 Linear Classifier

Another common approach for vertical selection in prior work is learning a supervised classifier based on click-through data [19, 1]. To this end, we trained a linear classifier for the task. The input to the classifier is a (sparse) feature vector representing the query as a bag-of-words. Specifically, each query term is a feature (dimension) in the vector and the value of each feature is 1 if the term appeared in the input query, or 0 otherwise. For example, the query "*aloe lotion where buy*" induces the sparse feature vector $\{\text{`}aloe\text{'}:1, \text{`}buy\text{'}:1, \text{`}lotion\text{'}:1, \text{`}where\text{'}:1\}$.

For training, we utilized the AROW online training procedure [10]. It is an efficient algorithm that can handle millions of training examples and millions of features; its performance was shown to be similar to SVM also for IR tasks [8].

3.2 Modeling Structure

Our hypothesis in this paper states that the structure of queries with question intent is distinguishable from that of other queries. We next describe several text representations that are meant to capture the query's structure, and how we incorporate them within the various models for vertical selection.

[1] `https://github.com/adampauls/berkeleylm`

3.2.1 Word position in query

The absolute position of each word in the query may hint at the type of query. A common example in CQA queries is those that start with a WH word, *e.g.* "*where can i sell mercury*" and "*how to get a boxer body*". If structure is an important facet of CQA queries, words of certain classes may tend to be in relatively stationary locations, while in general Web queries the word order is more arbitrary and thus these signals will not be of much use. We especially expect that position information may help in recognizing CQA queries when attached to function words such as WH words, prepositions, articles, etc.

To this end, on top of the word features described above for the linear classifier, we also induce more fine-grained features containing each word associated with its absolute position in the query. For example, for the query "*buy aloe lotion where*" we also generate the binary features {'*buy_1*':1, '*aloe_2*':1, '*lotion_3*':1, '*where_4*':1}.

3.2.2 Part-of-speech tags

Part-of-speech (POS) tagging is a method where each word in a given text is assigned a POS tag from a set of roughly 40 possible tags [24]. Each POS tag represents a category of words that have similar grammatical properties. For example, the text "*how to hide camera in a bag*" will be tagged as "*WRB TO VB NN IN DT NN*", marking '*how*' as a wh-adverb (WRB), '*hide*' as a base verb form (VB), '*camera*' and '*bag*' as common nouns (NN) and '*in*' as preposition (IN). This text representation is structure-oriented: the semantically unrelated sentence "*how to access ip phone over the internet*" will receive a very similar tag sequence, "*WRB TO VB NN NN IN DT NN*". Conversely, the word '*hide*' which acts as a verb in the first query, will receive the noun tag in the query "*quality hide jacket*", demonstrating the representation's sensitivity to structural context. We therefore extract POS information from the inferred tag sequence per input query to enhance our baseline models, expecting this information to be leveraged to capture the query structure.

To this end, we used the OpenNLP[2] toolkit to train a specialized POS tagger model for Web queries, following the method described in [13]. For the supervised linear classification model, we provide each POS tag as an additional feature, whose value is the number of times the specific tag appears in the query. For example, for "*how to hide camera in a bag*", we induce the features {'*WRB*':1, '*TO*':1, '*VB*':1, '*IN*':1, '*DT*':1, '*NN*':2}. In addition, we also generate the composite features combining the POS tag and its position in the query. For the same example query, the additional (binary) features are {'*WRB_1*':1, '*TO_2*':1, '*VB_3*':1, '*NN_4*':1, '*IN_5*':1, '*DT_6*':1, '*NN_7*':1}. This feature representation enables generalization of word-position signals. For example, all queries that start with a WH word would be represented by the feature '*WRB_1*'.

In the case of language models, which do not use a bag-of-word vector representation but rather learn probabilities over sequences, we replace each word with its POS tag and feed the transformed query to the model. Therefore, the resulting language model is trained to recognize POS tag sequences of the target domain. However, preliminary analysis showed that the granularity obtained by this method is too coarse, implying that content words cannot be removed altogether from the model. Consequently, we introduced a backoff to the language model generation. Specifically, for each domain we created a joined list of ∼200 most-frequent words in the domain and in general queries, based on pre-analysis of the training queries. If a query word appears on the list it is not replaced by its POS tag and is kept as is. The transformed queries in each training set were used to train two POS-tag language models, PLM_D and PLM_{G^D}, in a similar manner to the baseline language model. Here too the deciding score for an incoming query was computed by subtracting the general model score from the domain model score.

We emphasize that for training and testing each language model (of a specific domain), the transformed input query, following the backoff procedure, may be different for the same original input query. For example, the query "*buy aloe lotion where*", POS tagged as "*VB NN NN WH*", is transformed for the CQA domain with "*VB NN NN where*", while under the Shopping domain it is transformed to "*buy NN NN where*". This is due to the fact that while '*where*' is a frequent word in both domains, and '*aloe*' and '*lotion*' are rare in both domains, '*buy*' is frequent in the Shopping domain but rare in the CQA domain.

Cluster	Example words
C3	login, online, mail, home, school, bank
C4	how, happened, to, this, yourselves
C6	google, yahoo, download, gmail, tv, hotmail
C10	facebook, twitter, chris, michael, jennifer
C22	constricted, favoritism, diet, tuberculosis
C28	2015, 2014, news, game, vs, nfl, club

Table 1: Example word clusters

3.2.3 Word clusters over embedded representations

Part-of-speech tagging is one way to group words into clusters that capture some notion of structure in texts. However, there are many other word clustering algorithms which may further assist in differentiating queries of a target domain from general Web queries. Quite a few clustering algorithms for words were proposed over the years, *e.g.* Brown Clusters [7] to name one. Lately, word embedding approaches showed good performance in different Natural Language Processing tasks compared to prior word representations [9, 23, 25]. These models embed words into low-space vector representations that still preserve some aspects of the word semantics such as its closeness in embedded space to related words. We therefore experimented with word clusters that are induced from embedded word vectors.

Specifically, in this work we employ the SkipGram algorithm [25] which learns word embedding in an unsupervised way by optimizing the vector similarity of each word to context words in a small window around its occurrences in a large corpus. We learned[3] the mapping of 300,000 words to a 100-dimension embedded space over a corpus consisting of 7.5 million Web queries, sampled randomly from a query log. We then clustered the learned word vectors into 30 clusters using the K-means algorithm. We experimented with other amounts of clusters as well on a held-out dataset, and found that number of clusters at the range of several dozens (30-

[2]https://opennlp.apache.org/

[3]We used the word2vec implementation in `https://code.google.com/p/word2vec/`

50) performed the best in out task. It is interesting to note that this range is similar to the number of POS tags. We chose the smallest amount that provided best performance (30).

Inspecting the various clusters, we found that the semantic cue in the clusters is coarse and captures words of different classes together: socially-oriented commercial names like *facebook* share a cluster with people's names like *chris*; other technologically-oriented commercial names like *google* are in a cluster with other technological concepts like *download* and *tv*; another cluster includes words from the reporting domain, such as event-related concepts like years and *game*, as well as sports concepts like *nfl* and *club*. On the other hand, some clusters are more focused on function words, grouping WH words, prepositions and determiners. Overall, the clusters combine words based on semantic relatedness with structural resemblance. Table 1 illustrates a few of the generated clusters.

We utilize the clusters in the same way we utilize POS tags. For the linear classification model, we provide each word's cluster ID as an additional feature, whose value is the number of times the cluster appears in the query. For example, for "*how to hide camera in a bag*", we induce the features {'*C4*':5, '*C8*':1, '*C18*':1}. We also generate the composite features referring to cluster IDs with word positions, *e.g.* {'*C4_1*':1, '*C4_2*':1, '*C4_3*':1, '*C8_4*':1, '*C4_5*':1, '*C4_6*':1, '*C18_7*':1} for this example. For the language model representation, each word is replaced by its cluster ID and the resulting sequence is fed into the model. Consequently, such language model models the sequence of cluster ids instead of the sequence of words.

4. RANDOM FOREST FOR MODELING QUERY STRUCTURE

We expect that the structure of CQA queries would include relationships between the query words that imply particular question semantics. To better model the complex structure of CQA queries we employ a variant of the Random Forest algorithm [5], which is a non-linear classifier that can represent relationships between different input elements. Our random forest variant follows the overall architecture of random forests - each tree in the forest is constructed on a subset of the input, and each tree uses randomness in the process of selecting features to split leaves on. However, the techniques we use in building the trees, in particular the choice of variables and values used to split nodes of the tree, are fairly distinct.

We next present our random forest model. First, we describe its overall structure (Sec. 4.1), its input and output. Then, we detail our forest learning algorithm (Sec. 4.2). Finally, we discuss some intuitions on what our model learns to represent (Sec. 4.3).

4.1 Random Forest Architecture

The input to our random forest is all categorical, and is given as key-value pairs. In particular, each example is represented by two types of inputs. The first type is a single Boolean value – the decision of the baseline AROW linear classifier, as described in Sec 3.1.2. This input is based on information about each word independently of each other. The second type of input is a list of pairs, each has the word position in the query as the key, and the word-cluster ID of the word in the corresponding position as the value (cluster IDs are generated as described in Sec. 3.2.3). For example, for the input query "*how to hide camera in a bag*", the input to the random forest is $\{\langle AROW : false\rangle, \langle 1 : C4\rangle, \langle 2 : C4\rangle, \langle 3 : C4\rangle, \langle 4 : C8\rangle, \langle 5 : C4\rangle, \langle 6 : C4\rangle, \langle 7 : C18\rangle\}$. Therefore, this classifier relies on a basic modeling of the contents of the query (captured by the Boolean signal from the linear classifier) and focuses on modeling the structure of the query by analyzing the word clusters in the various query positions.

The output of the model is a probability distribution over the possible output classes. An example for our CQA intent classification task may be $\{G : 0.3, CQA : 0.7\}$, which means that the forest assessment of an input query is that it is a general Web query (G) with 30% probability, and a CQA query (CQA) with 70% probability.

Our random forest is composed of binary trees and a weight associated with each tree. Each tree is composed of internal nodes and leaves. Each leaf contains a probability distribution over the possible output classes, which may serve as the model's output. Each decision rule in an internal node contains a subset of input key-value pairs, which are evaluated as a disjunction. When classifying an input instance, the model accepts the rule if at least one of the keys of the input matches an associated value in the decision rule. An example rule is $\left(AROW : \text{'}false\text{'} \vee 1 : \text{'}C4'\right)$, stating that this rule should be accepted only if at least one of the two statements is true: (a) the AROW classifier decision for the input query is '*false*'; (b) the first word in the query is mapped to cluster 4.

When classifying an input query using the forest we first evaluate the input on each tree by beginning at the root and traversing the tree until a leaf is reached, following the edges according to its conformance with each rule encountered. The output of the tree is probability distribution over classes in the leaf that was reached. The final output of the forest is a weighted sum of the class distributions output by all of the trees in the forest.

Our selected encoding of the input query as pairs of word-positions and their respective cluster id values allows us to employ the random forest architecture over variable length input. Together with the nature of our decision rules, which are not limited to a single variable, this model enjoys a fairly high level of expressiveness. For example, since at each tree node specific text positions are inspected, a rule can express a convolutional operator, *e.g.* '*one of the first three words is in C15*'. Missing positions, which may occur due to short texts, are handled by considering the inspected position/value pair in the mode as a mismatch. For example, if a part of a decision rule is '*5:C8*' (the fifth text position contains a word mapped to cluster 8), this part of the rule does not hold for queries of length less than 5.

We note that during our research we also trained our random forest using the query words directly, instead of their mapped clusters. However, this resulted in severe overfitting. We suspect that this is due to the large word vocabulary and the sparse nature of the examples, for which linear classification is more robust.

4.2 Random Forest Learning

The input to the training process of the forest is a set of labeled examples S and the number of target trees in the

Algorithm 1 Reweigh Trees

1: **function** REWEIGH-TREES
Require: a set of labeled examples S_i; trees learned so far $\{T_j\}_1^{i-1}$ and their weights $\{w_j\}_1^{i-1}$; re-weighting constants α_s, α_ℓ
2: **for all** $s \in S_i$ **do**
3: **if** s is correctly labeled by forest **then**
4: $\alpha \leftarrow \alpha_s$
5: **else**
6: $\alpha \leftarrow \alpha_\ell$
7: **for all** $1 \leq j < i$ **do**
8: **if** T_j classified s correctly **then**
9: $w_j \leftarrow w_j + \alpha$
10: **else**
11: $w_j \leftarrow w_j - \alpha$

forest N_{RF}. The output is a set of trees and, for each tree, a weight associated with it.

Our learning algorithm starts by randomly splitting the training set S into $N_{RF}+1$ training subsets $\{S_i\}_1^{N_{RF}+1}$. Each subset is sampled while keeping a balanced distribution of positive and negative examples, without repetitions ($\forall_{i \neq j} S_i \cap S_j = \emptyset$). The training algorithm iterates once over all sampled subsets. For each subset, it first tunes the weights of all currently learned trees in the forest. Then, for each subset except the last one, the algorithm generates a new tree that attempts to optimize the classification for this specific subset. The weight of the new tree is initially set to 0[4]. We next describe the algorithm parts for re-weighting existing trees and for learning a new tree given a training subset S_i.

The procedure that re-weighs existing trees in the forest is described in Algorithm 1. It increases the weight of trees that have correctly classified an example, and decreases the weight of trees that misclassify an example. To this end, it uses two different increment constants: α_s is used when the forest as a whole has correctly classified an example, and α_ℓ is used when the forest has misclassified an example. One can think of α_s as a small change in weight reflecting the model's increased confidence in the correctness of the tree, while α_ℓ is larger and helps to correct a mistake in classification.

For each training subset a new tree is learned and added to the forest. The tree starts with a single leaf node, containing the class distribution in the training subset S_i. Then, the tree structure is constructed in an iterative manner by splitting leaves of sufficient size. This process is summarized in Algorithm 2. In Step 5 of the algorithm, all leaves that are big enough to be considered for splitting are selected. In Step 6, a decision rule for splitting is chosen per selected node (see Algorithm 3), and the algorithm picks the one that results in highest information gain. This leaf is split according to the rule and the resulting new leaves are added to the tree.

The process by which a candidate decision rule is created for a node is summarized in Algorithm 3. The rule is built in a greedy manner, beginning with an empty rule. The algorithm randomly samples a single example e from all the examples in S_i that reach the target node at inference time

[4]The last subset is only used to tune the weights, otherwise the weight of the last learned tree would remain 0.

Algorithm 2 Learn Tree

1: **function** LEARN-TREE
Require: A set of labeled examples S_i; maximum number of nodes *max-nodes*; minimal node size for split *split-size*
2: $root \leftarrow \{S_i\}$
3: $N \leftarrow \{root\}$
4: **while** $|N| \leq$ *max-nodes* **do**
5: $BL \leftarrow \{l \in leaves(N)\ s.t.\ |l| \geq split\text{-}size\}$
6: $best\text{-}split \leftarrow argmax_{l \in BL} IG(l, Descision\text{-}Rule(l))$
7: $best\text{-}split.split(Descision\text{-}Rule(best\text{-}split))$
8: $N \leftarrow N \cup best\text{-}split.children()$

Algorithm 3 Get Decision Rule for Leaf

1: **function** DECISION-RULE
Require: A leaf l containing a set of labeled examples E_l; max number of unsuccessful attempts to add to rule *max-tries*; minimum information gain for adding to rule *min-ig*
2: $rule \leftarrow \emptyset$
3: $failed\text{-}attempts \leftarrow 0$
4: **while** $failed\text{-}attempts < max\text{-}tries$ **do**
5: $e \leftarrow Uniformly\text{-}Sample(E_l)$
6: $\langle k : v \rangle \leftarrow Uniformly\text{-}Sample(e)$
7: **if** $IG(l, rule \cup \langle k : v \rangle) > IG(l, rule) + min\text{-}ig$ **then**
8: $rule \leftarrow rule \cup \langle k : v \rangle$
9: **else**
10: $failed\text{-}attempts \leftarrow failed\text{-}attempts + 1$
return *rule*

(Step 5). It next randomly samples a single key-value pair $\langle k : v \rangle$ from e (Step 6) and checks if adding $\langle k : v \rangle$ to the currently constructed rule would improve its splitting performance, increasing information gain in a non-negligible manner. If performance is improved, $\langle k : v \rangle$ is added to the rule. This procedure is repeated as long as performance is increased or up to a limit of failed attempts.

To illustrate the process of building a decision rule in a node, consider a node that contains the following six labeled examples:
· $G : \{\langle AROW : false \rangle, \langle 1 : C1 \rangle, \langle 2 : C5 \rangle\}$
· $G : \{\langle AROW : true \rangle, \langle 1 : C3 \rangle\}$
· $G : \{\langle AROW : false \rangle, \langle 1 : C3 \rangle, \langle 2 : C4 \rangle, \langle 3 : C5 \rangle\}$
· $CQA : \{\langle AROW : true \rangle, \langle 1 : C2 \rangle, \langle 2 : C4 \rangle, \langle 3 : C5 \rangle\}$
· $CQA : \{\langle AROW : true \rangle, \langle 1 : C3 \rangle\}$
· $CQA : \{\langle AROW : true \rangle, \langle 1 : C5 \rangle, \langle 2 : C6 \rangle, \langle 3 : C1 \rangle, \langle 4 : C2 \rangle\}$
For simplicity we disregard node size. Let's say the minimal information gain we want is 0.3. We begin with the empty rule and an information gain of 0. First the key-value pair $\langle 1 : C3 \rangle$ from the second example is randomly picked. This rule splits the node into two sets, with two general query (G) examples going together with one CQA example, say to the left subset. This split results in information gain greater than 0.3, so this key-value pair is added to the rule. Next, $\langle AROW : false \rangle$ is randomly picked from the first example, which moves the remaining G example to the left. As this also improves the information gain by more than 0.3 over the previous best gain, the rule now becomes $\big(1{:}\text{`}C3\text{'} \vee$

Model	CQA	Shopping
PosLM	0.84	0.85
BasicLM	0.87	0.88
ClusterLM	0.87	0.88
RandomForest	0.91	0.90
BasicAROW	0.91	0.92
FullAROW	0.94	0.94

Table 2: AUC scores for each model on each domain.

AROW:'*false*'). After several additional iterations in which no improvement to the rule is found, the procedure ends.

4.3 What the Model Actually Learns

It is often hard to interpret the results of a non linear classifier that contains many complex elements, which is our case with a forest containing complicated decision rules in its trees' nodes. In an effort to understand some of what is learned, we examined the splitting rule in the root of each of the trees in the learned forest.

The classification signal extracted using the linear model naturally served as a strong predictor. In each of the tree roots the result of the linear classification was one of the variables in the rule. However, it rarely constituted the entire splitting rule. The following examples are splitting rules in the roots of trees, as observed.

(1) (1:'*C13*' ∨ AROW:'*false*') — The most frequent words in cluster *C*13 are *www.google.com, Amazon, www.friv.com, HomeDepot, ebay.com, expedia, .co.uk, bt, hulu, hosting.* Site names and brands at the beginning of a query (position 1) may indicate navigational intent.

(2) (1:'*C4*' ∨ 2:'*C4*' ∨ 2:'*C6*' ∨ 3:'*C4*' ∨ 4:'*C4*' ∨ 6:'*C4*' ∨ 7:'*C10*' ∨ AROW:'*true*') — The most frequent words in cluster 4 are function words, like *in, for, to, the, and, a, undefined, how, on, is.* The model here recognizes that words from cluster 4 in most locations indicate a full natural language question and favor a true result. This is not a very structural result, but rather shows how the rule can also capture location irrelevant information.

(3) (1:'*C4*' ∨ 4:'*C22*' ∨ 7:'*C4*' ∨ AROW:'*true*') — The most common words in cluster *C*22 are *symptoms, pain, treatment, skin, cancer, natural, surgery, diet, breast, effects.* These words are indicative of health-related queries, which are common for CQA website search. Word position information may be relevant here, since health-related words at the beginning of the query may indicate a more topical query without a specific question intent.

5. EXPERIMENTS

5.1 Experimental Settings

Data Sets.

As a training set, we obtained a log of all queries issued to a general search engine over the period of June 1-21, 2015. We marked CQA queries as those landed on three leading CQA sites: *Yahoo Answers*, *Answers.com*, and *Quora*, and Shopping queries as those landed on two leading Shopping sites: *eBay* and *Amazon*. We randomly selected 300,000 CQA queries, 300,000 Shopping queries, and 300,000 Web queries, uniformly sampled from the query log. A held out

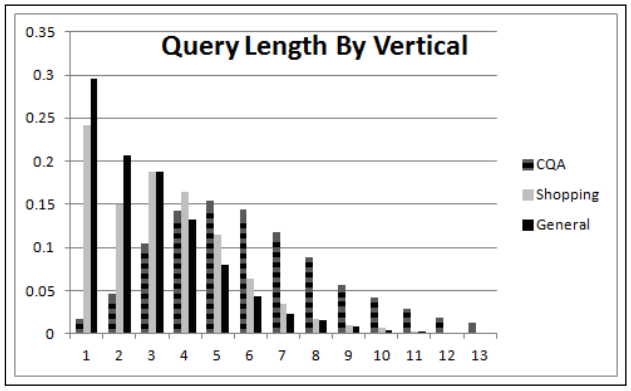

Figure 2: Query length distribution for the CQA domain, the Shopping domain, and the Web domain in our test sets.

set of 1,000 queries was randomly extracted from each training set for parameter tuning. We then extracted a test set using the same procedure out of the query log for August 1-7, 2015, sampling 11,000 queries per domain.

All queries in our experiments were cleaned by lowercasing and removing foreign characters and punctuation characters. The queries were then split into word tokens by whitespace, after which tokens beginning with the '*site:*' prefix were also removed.

Tested Models.

We trained and evaluated the models presented in Sec. 3.1 on each domain. We tested the language model and the linear classifier baselines (which use only the words in the query), denoted by *BasicLM* and *BasicAROW* respectively. For AROW training, the R parameter was set to 100, and 10 iterations were performed on the input.

We also tested models that address the query structure. Specifically, we constructed a language model over POS tags, denoted by *PosLM*, and a language model over cluster words, denoted by *ClusterLM* (see Sec. 3.2). We also evaluated an AROW classifier, denoted by *FullAROW*, which gets as input both word features and the structure-oriented features introduced in Sec. 3.2. The same AROW parameters of the baseline model were used.

Finally, we assessed our random forest model (Sec. 4), denoted by *RandomForest*. For each domain, the size of the training subsets S_i was set to $10,000$, the maximum number of nodes per tree was set to 150 and the minimal node size for split was set to 40. The minimal information gain for selection was set to 5. The number of trees in the forest (N_{RF}) was set to 15, the update constants were set to $\alpha_s = 0.01$, $\alpha_\ell = 0.05$.

5.2 Results

We evaluated each tested model on the vertical selection task both for the CQA domain and for the Shopping domain. We measured the performance of each classifier over every test-set in our experiments using Area under the ROC Curve (AUC) [20], which avoids the subjectivity of threshold selection and reflects the main objective of query type identification, *i.e.* filtering before sending to a vertical search engine, better than either precision or recall measures. We com-

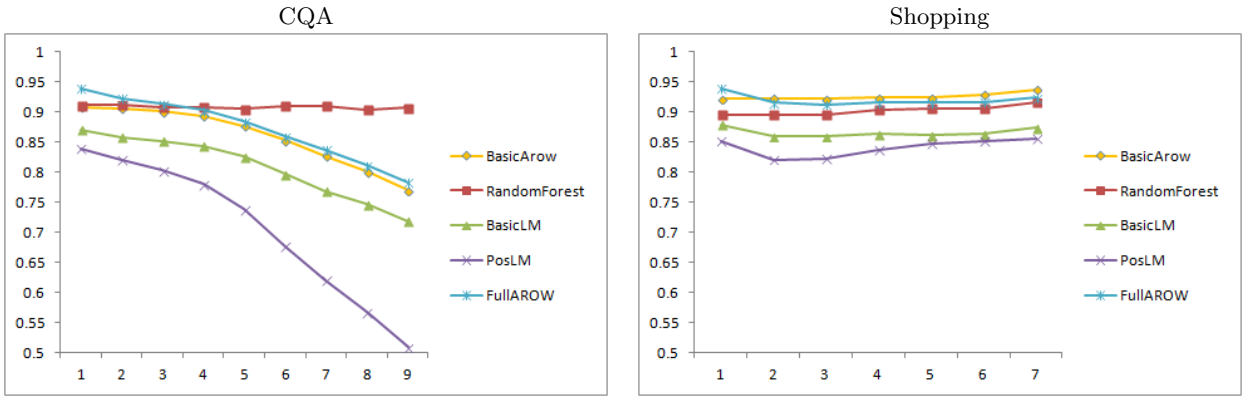

Figure 3: AUC per domain and model. Each point represents the AUC value (Y-axis) over all queries of a given length (X-axis) and above.

Table 3: Number of test queries of length N or more in each domain

Domain	length ≥ 1	length ≥ 3	length ≥ 5	length ≥ 7	length ≥ 9
CQA	11000	10313	7585	4317	2047
Shopping	11000	6702	2821	849	280
General	11000	5473	1963	611	195

puted statistical significance between model performance using the following procedure: a) the test set was split into 11 distinct subsets; b) AUC was measured for each model on each subset; c) the AUC results of the two models for the subsets were compared using the Mann-Whitney test.

The overall results for our datasets are shown in Table 2. These results show that all models perform well for both domains. As expected, the three supervised discriminative models, *BasicAROW*, *FullAROW* and *RandomForest*, outperform the language models *BasicLM* and *PosLM*. We note that the language model over clusters *ClusterLM* performed on par with the baseline language model *BasicLM*. We therefore do not present its results in the more detailed analyses. All supervised models improve over the language models with statistical significance at $p < 0.01$.

In both domains, *FullAROW* significantly outperforms all other models ($p < 0.01$), which may hint towards the benefit of using our structure-oriented features. The overall results, however, only depict part of the story. Many recurring general Web queries are simple navigational queries, such as '*facebook*' or '*citibank*', which may turn the classification task into an easy one. Yet, looking at query length distribution for general Web queries as well as for each of the domains in our test sets (Figure 2) reveals some of the differences between the two domains. Indeed, while general Web query length distribution behaves like a dropping long-tail curve, as expected, and the Shopping domain shows strong tendency towards short queries as well, most queries in the CQA domain are much longer. As a consequence, if we use a simple classifier based only on query length, the AUC for the Shopping domain is 0.57 but for the CQA domain it is 0.83. It is therefore important to look at the performance of the models for queries of different lengths. This is especially important for the CQA domain, since most of its associated queries (6-7 terms on average) are longer than average Web queries, and while the fraction of long general Web queries is small, their absolute volume in the overall query log still exceeds that of CQA queries. Therefore it is crucial for vertical selection models to do well also on long queries.

Accordingly, we analyzed the models' performance for different query lengths. Figure 3 presents the AUC score per domain and model by query length, where each query length N refers to all queries in the test set that have $\geq N$ words (Table 2 corresponds to the results for $N = 1$). The test set sizes per query length cutoff and domain appear in Table 3. We note that while some test sets are unbalanced, the AUC metric depends on the rate of acceptance on domain examples vs. rate of acceptance on general examples, therefore it is robust for this kind of imbalance.

We first look at the AUC scores for the Shopping domain (Figure 3(b)). For this domain the length of the query appears to have no effect on model performance (the performance of each model for the different query subsets is virtually the same). Specifically, the top performers are the two linear classifiers, *BasicAROW* and *FullAROW*, which perform comparably, followed closely by the random forest model. These results indicate that the Shopping domain is modeled well by content keywords and that query structure, if present at all, is quite loose in this domain. The gain of the linear classifiers over *RandomForest* is statistically significant at $p < 0.01$ for all length-specific test sets.

Observing the most important positive features, out of the $500,000$ features used in the *FullAROW* classifier, strengthens this conclusion: while some relate to navigational hints for sites like *Amazon* or *eBay*, many are related to products, *e.g.* '*book*', '*dvd*', '*toys*' and '*albums*'. Similarly, important negative features contain hints of navigational queries, such as '*twitter*' or '*tumblr*', but also strong indication of domains other than Shopping, such as '*installation*' and '*insurance*'. On the other hand, structure-oriented features are not abundant in the top features.

Table 4: Example queries of different lengths for each domain

Length	General Queries	CQA Queries	Shopping Queries
1	· facebook · lebron	· quroa · monocytes	· ebay · vaz-2106
3	· greenhill humane society · george michael horrible	· mentalist vs sherlock · divorce rate asian	· paradice pool vacume · used golf clubs
5	· cars with bad hail damage · david foster wallace suicide note	· how long do mosquitoes live · great niece or grand niece	· ebay motors 2001 head lights · furniture for my first dollhouse
7	· Emergency Garage Door Repair in Jacksonville, FL · song with lyrics you shouldn't have promised	· will denatured alcohol damage wood floors flooring · rash upper body going to the head	· duct work for Broan PM250 exhaust fan · Reebok Men's DMX Max Mania Walking Shoes
9	· jon stewart daily show says goodbye to fox news · bank of america online sign in to online banking	· how can i tell if my tattoo is infected · difference between board of nursing and professional nursing organizations	· window tiers 17 l x 20 w for bathrooms · Legends of the diamond Limited Edition Forever Colectibles Guerrero

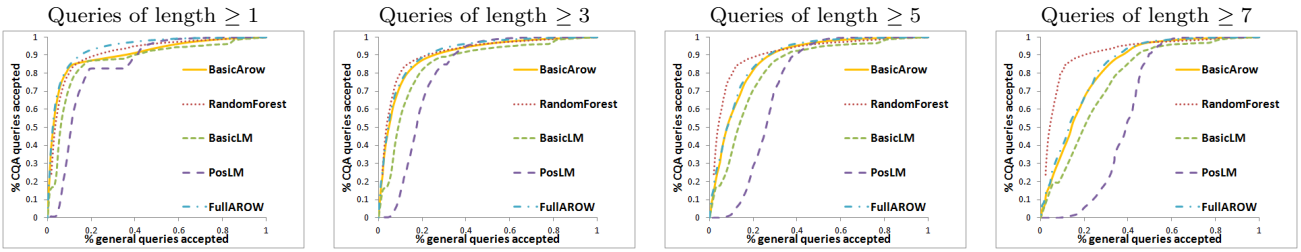

Figure 4: ROC for CQA queries by length and model. The X-value represents the proportion of general queries which were accepted by the model. The Y-value represents the proportion of the vertical queries that were accepted.

Next, we analyze the model performance for the CQA domain (Figure 3(a)). The results reveal a very different model behavior. For test sets of query lengths ≥ 1 and ≥ 2 the linear classifiers significantly outperform *RandomForest* at $p < 0.01$. Yet, once the test sets are confined to long queries, the performance of both the linear classifiers and the language models drops significantly. The only model that maintains a high level of classification is the *RandomForest* model. The gap keeps increasing as query length grows, and is evident also in the ROC curves for specific query lengths, presented in Figure 4. At test sets of query lengths $\geq 5...9$[5], *RandomForest* improvement over all other models is statistically significant at $p < 0.01$. As shown in Figure 2, roughly 66% of CQA queries are of length 5 or more. This means that for a majority of the full query set, content-oriented features alone are not enough for modeling the difference between CQA queries and other long queries. On the other hand, *PosLM*, which models only structure, performs the worst, showing that a combination of content and structure bearing signals is necessary. Only our proposed *RandomForest* model manages to learn the discriminating features of long queries as well as those of short ones, and successfully differentiates between CQA queries and other queries even at queries of length 9 and above.

Table 4 presents examples for queries of different length in each domain, which illustrate the differences between the tested domains. For general queries and the Shopping domain, queries are more topical in nature, and even long queries are usually not strongly structured, and may be viewed as a bag of keywords. On the other hand, CQA queries resemble complete questions, and the relationships between words are stronger.

To gain a more fine-grained understanding of the model performance on the different domains, we also regarded each specific website, which was used for extracting queries either for the CQA or the Shopping domain, as a specialized "*domain*". This experiment further tests the robustness of the different models to idiosyncrasies of various websites. As before, we constructed a training set and a test set where positive examples were queries landing on the target website, while negative examples were sampled uniformly from the query log. Overall, we had 5 different specialized domains, and each model was trained and tested on each website's data. We found that the results for the specialized domains strongly adhere to the trend shown in the main results, namely that once query length grows above 3, the *RandomForest* model substantially outperforms all other models for the three CQA websites, while for the Shopping websites all models performed well for all query lengths, with the linear classifiers at the lead followed closely by the *RandomForest* model.

Our experiments show the importance of designing vertical selection models with the capability to handle queries of different lengths. In this work, we investigated single classifiers, designed for all queries, that are robust to query length. A different approach would be to train a classifier per query length. This may show some advantage over a single classifier, but will cost more in terms of space constraints (M models in memory per M query lengths instead of a single model). This deserves further investigation which we leave for future work. Overall, the presented results indicate that a single classifier can handle all queries of different length

[5]For test sets ≥ 3 and ≥ 4 the difference between the three supervised models is not statistically significant.

Table 5: Error analysis: types of errors, example queries and the error's frequency.

Type of Error	Examples	Frequency (%)
False Positives		
Explicit CQA question format	· how to dress when you are obese and 50s and look pregnant but not · if you get married does your credit combine	80%
A specific scenario - implicit CQA question intent	· most powerful air rifle on the market · i have white spots on my toe nails	15%
Others	· she liked books more than anything else burnett · lyrics to until we meet again hymn	5%
False Negatives		
Non-whitespace separated queries	· should.you.drink.coffee before.running · whatsideofyourbodyisyourhearton	35%
Non-standard navigational	· QOURA · qura.cpm	5%
Rare words	· hypocalcemiaqt · erienerphobia	5%
Uncommon names, locations, non-English words	· abvimas manali · Bir Atrium Makati Map	15%
Non-CQA, suggesting user exploration	· po300 · QUANTITY: Size Chart	40%

robustly and with high performance when constructing an appropriate classification model.

5.3 Error Analysis

We examined the 100 general Web queries that got the highest scores by the *RandomForest* model, and the 100 CQA queries that got the lowest scores. These queries constitute the top false positive and false negative errors of this model, respectively. The various types of errors we found, as well as example queries per type, are presented in Table 5.

From the table it is fairly clear that the vast majority of general queries that got a high score (false positives) are simply CQA queries sampled from the general query log, and as expected these queries are ranked high. We already discussed the noisy nature of our data-set (see last paragraph in Sec 3.1.1). It does imply, however, that our model's true performance should be somewhat higher than estimated.

On the other hand, false negative errors (CQA queries that received low scores) are more diverse. Still, they can largely be categorized into two types. The first type includes queries that require better text processing, such as spelling correction and considering '.' as a word separator (a common error in mobile usage). The second type of errors concerns rare words that do not appear in our word-to-cluster mapping, and therefore are treated in a default, non-optimized manner. The issue of rare terms is known for word embedding in general, and in the future we would like to address it for this task but also in a more generic setting.

6. CONCLUSIONS

We introduced the novel task of detecting queries with question intent as a variant of the vertical selection problem. Typically, queries with question intent can be successfully answered by a CQA vertical, hence we focus on CQA queries, *i.e.* those that landed on CQA sites. Since the structure of CQA queries is quite different from the structure of general Web queries, we proposed to model the structure of a query, on top of its content, for this task. We introduced two classification schemes that take query structure into consideration. In the first approach, we induce features from the query structure as an input to supervised linear classification. In the second approach, word clusters and their positions in the query are used as input to a random forest classifier to identify discriminative structural elements in the query.

We evaluated our proposed classifiers against the popular language model approach for vertical selection, as well as a linear classifier over the query terms. To show the difference between the CQA domain and other domains, we assessed the classification models on two datasets, one for the CQA detection task and one for the Shopping domain detection task. Our results show that while keyword modeling seems enough for the Shopping domain, for the CQA domain structure modeling substantially boosts classification performance as query length increases, differentiating between non-CQA long queries, such as a name of a long movie or the location of a university, and CQA queries, which pose a specific question or scenario.

Since CQA query structure is complex, we want to test in future work whether additional advanced text processing techniques, such as Dependency Parsing and Abstract Meaning Representation, can help in detecting such queries. We would also like to identify other domains where query structure plays an important role, and where our structure-based classification schemes can bring value. Finally, our best-performing model for long queries, random forest over variable length text, is a general classification scheme. We would like to find other tasks in which it may be found beneficial.

7. REFERENCES

[1] J. Arguello, J. Callan, and F. Diaz. Classification-based resource selection. In *Proceedings of CIKM'09*, pages 1277–1286. ACM, 2009.

[2] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of evidence for vertical selection. In *Proceedings of SIGIR'09*, pages 315–322. ACM, 2009.

[3] J. Arguello, F. Diaz, and J.-F. Paiement. Vertical selection in the presence of unlabeled verticals. In *Proceedings of SIGIR'10*, pages 691–698. ACM, 2010.

[4] C. Barr, R. Jones, and M. Regelson. The linguistic structure of English Web-search Queries. In

Proceedings of EMNLP'08, pages 1021–1030. ACL, 2008.

[5] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, Sept. 2002.

[7] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[8] D. Carmel, A. Mejer, Y. Pinter, and I. Szpektor. Improving term weighting for community question answering search using syntactic analysis. In *Proceedings of CIKM'14*, pages 351–360. ACM, 2014.

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[10] K. Crammer, A. Kulesza, and M. Dredze. Adaptive regularization of weight vectors. *Machine Learning*, 91(2):155–187, 2013.

[11] T. Demeester, D. Trieschnigg, D. Nguyen, K. Zhou, and D. Hiemstra. Overview of the Trec 2014 federated web search track. 2014.

[12] F. Diaz. Integration of news content into web results. In *Proceedings of WSDM'09*, pages 182–191. ACM, 2009.

[13] K. Ganchev, K. Hall, R. McDonald, and S. Petrov. Using search-logs to improve query tagging. In *Proceedings of ACL'12*, 2012.

[14] J. Hu, G. Wang, F. Lochovsky, J.-t. Sun, and Z. Chen. Understanding user's query intent with wikipedia. In *Proceedings of WWW'09*, pages 471–480. ACM, 2009.

[15] V. Jethava, L. Calderón-Benavides, R. Baeza-Yates, C. Bhattacharyya, and D. Dubhashi. Scalable multi-dimensional user intent identification using tree structured distributions. In *Proceedings of SIGIR'11*, pages 395–404. ACM, 2011.

[16] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[17] C.-J. Lee, R.-C. Chen, S.-H. Kao, and P.-J. Cheng. A term dependency-based approach for query terms ranking. In *Proceedings of CIKM'09*, pages 1267–1276. ACM, 2009.

[18] X. Li. Understanding the semantic structure of noun phrase queries. In *Proceedings of ACL'10*, pages 1337–1345. ACL, 2010.

[19] X. Li, Y.-Y. Wang, and A. Acero. Learning query intent from regularized click graphs. In *Proceedings of SIGIR'08*, pages 339–346. ACM, 2008.

[20] C. X. Ling, J. Huang, and H. Zhang. Auc: a statistically consistent and more discriminating measure than accuracy. In *Proceedings of IJCAI'03*, volume 3, pages 519–524, 2003.

[21] Q. Liu, E. Agichtein, G. Dror, E. Gabrilovich, Y. Maarek, D. Pelleg, and I. Szpektor. Predicting web searcher satisfaction with existing community-based answers. In *Proceedings of SIGIR'11*, pages 415–424. ACM, 2011.

[22] Q. Liu, E. Agichtein, G. Dror, Y. Maarek, and I. Szpektor. When web search fails, searchers become askers: Understanding the transition. In *Proceedings of SIGIR'12*, pages 801–810. ACM, 2012.

[23] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of HLT 1*, pages 142–150. ACL, 2011.

[24] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: Annotating predicate argument structure. In *Proceedings of HLT '94*, pages 114–119. ACL, 1994.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *NIPS 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[26] D. E. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of WWW'04*, pages 13–19. ACM, 2004.

[27] M. Shokouhi and L. Si. Federated search. *Found. Trends Inf. Retr.*, 5(1):1–102, Jan. 2011.

[28] L. Si, R. Jin, J. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of CIKM'02*, pages 391–397. ACM, 2002.

[29] R. W. White, M. Richardson, and W.-t. Yih. Questions vs. queries in informational search tasks. In *Proceedings of WWW'15 Companion*, pages 135–136. WWW, 2015.

[30] W. Zhang, Z. Ming, Y. Zhang, L. Nie, T. Liu, and T.-S. Chua. The use of dependency relation graph to enhance the term weighting in question retrieval. In *Proceedings of Coling'12*, pages 3105–3120, 2012.

[31] K. Zhou, R. Cummins, M. Lalmas, and J. M. Jose. Which vertical search engines are relevant? In *Proceedings of WWW'13*, pages 1557–1568. WWW, 2013.