Chapter 1

Aggregated Search

1.1 Introduction

Modern search portals such as Google, Bing, and Yahoo! provide access to wide range of search services in addition to Web search. These search services, commonly referred to as *verticals*, include highly-specialized search engines that focus on a particular type of media (e.g., *images*, *video*), search engines that focus on a particular type of search task (e.g., *news*, *local*), and applications that return a specific type of information (e.g., *weather*, *stock quotes*). In the most general sense, *aggregated search* is the task of providing integrated access to all these different vertical search services, and to the core Web search engine, within a common search interface—a single query box and a unified presentation of results.

Given a user's query, an aggregated search system must make a number of predictions. As one might expect, not every vertical is relevant to every query. If a user issues the query "nyc pizza restaurant locations", it is almost certain that the user wants local business information and not images. Thus, an important step for an aggregated search system is to present those verticals most likely to be relevant and to suppress those most likely to be non-relevant. Moreover, queries are often ambiguous in terms of the user's actual intent. For example, if a user issues the query "new york style pizza" (Figure 1.1), it is unclear whether the user wants to find a place to eat (in which case local business results might be relevant) or wants to learn to cook New York style pizza (in which case recipes, how-to videos, and/or images might be relevant). Thus, another important step of an aggregated search engine is to resolve contention between verticals and to present those more likely to be relevant in a more salient way.

While aggregated search might seem like a new technology, it has its roots in a fairly mature subfield of information retrieval called *federated search*, where the goal is to provide automated search across multiple distributed collections or search engines. And, like most methods for federated search, most aggregated search methods approach the task in two subsequent steps: (1) predicting *which* verticals to present (*vertical selection*) and (2) predicting *where* to present them (*vertical presentation*). Vertical selection refers to the problem of selecting a subset of relevant verticals given a query. One way to

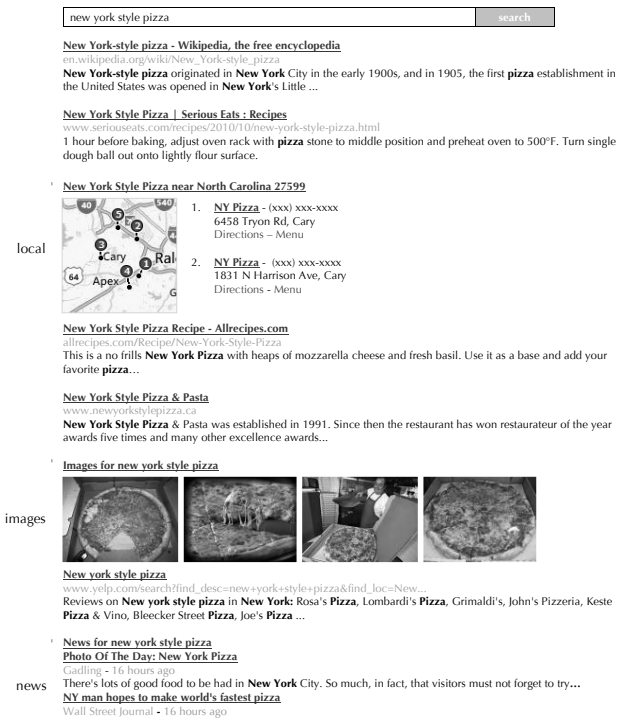


FIGURE 1.1: Given the query "new york style pizza", an aggregated search system decides to blend results from the *local*, *images*, and *news* into the core web results.

think about this is as the classification of verticals according to whether or not they should appear on the results page at all. Vertical presentation refers to the problem of interleaving this subset of verticals with the web results.[1]

In this chapter, we provide an overview of aggregated search techniques and methods for evaluation. As mentioned above, an aggregated search system must make predictions about which vertical(s) are more likely to be relevant to a user's query. State-of-the-art approaches combine multiple types of evidence to make these decisions. Section 1.2 provides an overview of features used in aggregated search and Section 1.3 provides an overview of machine

[1]This decomposition is primarily motivated by economics. Getting results from a vertical search engine oftentimes incurs some cost. This cost might be monetary if the aggregator is paying for access to a closed service. In the case of cooperative vertical search engines, this cost may be in computational if the vertical search engine cannot support portal traffic. Therefore, vertical selection can be a method for reducing the overhead of an aggregator.

learning methods for combining features in order to make aggregated search predictions. Section 1.4 focuses on methods for evaluation and covers methods for test collection evaluation and on-line evaluation . Finally, Section 1.5 covers special topics in aggregated search. Among these, we focus on methods for sharing training data among verticals and methods for eliciting and exploiting implicit user feedback to improve future aggregated search predictions.

1.2 Sources of Evidence

There are many ways in which an aggregated search system might be able to determine that a vertical is relevant to a query. Consider, for example, the task of predicting when to present the *news* vertical in response to a user's request. If the query contains the term "news", this is a strong indication that the *news* vertical is relevant. That said, not every newsworthy query will contain the term "news". Consider for example, the query "presidential debates". If many of the news articles currently in the news index contain these two query terms (or terms that are semantically related), this might also indicate *news*-vertical intent. In this case, the underlying assumption is that content supply (i.e., an abundance of query-related content in the underlying vertical collection) can help predict content demand. A system might also consider content demand directly. In many cases, users can navigate and issue queries directly to a particular vertical search engine. In a cooperative environment, the aggregated search system might have access to this vertical-specific query stream. Thus, another source of evidence is the number of similar queries issued to the vertical in the recent past. Finally, if a vertical is presented in response to a query, the aggregated search system can keep track of the user's actions. While implicit user feedback is subject to different types of bias, by averaging across many users, a system may be able to generate useful evidence for future impressions of the same query or similar queries.

Prior work shows that no single source of evidence can be used to predict that a particular vertical is relevant to a query [5, 3, 30]. Thus, state-of-the-art approaches to vertical selection and vertical presentation use machine learning to *combine* multiple types of predictive evidence as features. In this section, we provide an overview of different types of features used in prior work.

1.2.1 Types of Features

In learning about different types of features, it helps to be aware of their similarities and differences. Broadly speaking, features can be organized along two dimensions. The first dimension relates to whether the value of the fea-

ture depends only on the query (is the same for all verticals), only on the vertical (is the same for all queries), or is unique to the vertical-query pair. The second dimension relates to whether the feature value can be generated without issuing the query to the vertical; must be generated after issuing the query to the vertical, but before the vertical is presented to a user; or must be generated after the vertical is presented to a user.

With respect to the first dimension (i.e., the source of the feature value), there are three categories. *Query features* are generated from the query string and their values are independent of the candidate vertical. Examples may include whether the query contains a particular term (e.g., "news", "pics", or "weather") or a particular named entity type (e.g., a person, location, or organization). *Vertical features* are generated from the candidate vertical and their values are independent of the query. Examples may include the number of new documents added to the vertical collection or the number of queries issued directly to the vertical search engine in the recent past. Vertical features typically quantify bursts in vertical content supply or demand. Finally, *vertical-query features* are a function of both the query and the vertical in question. Examples may include the number of hits in the vertical collection or the similarity between the query the vertical's query stream.

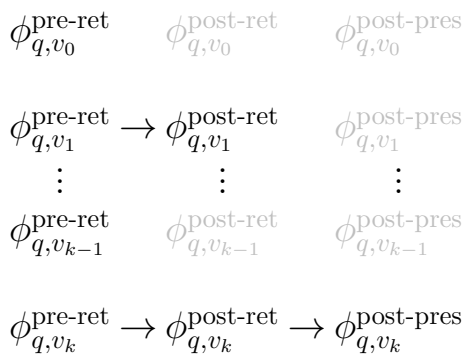


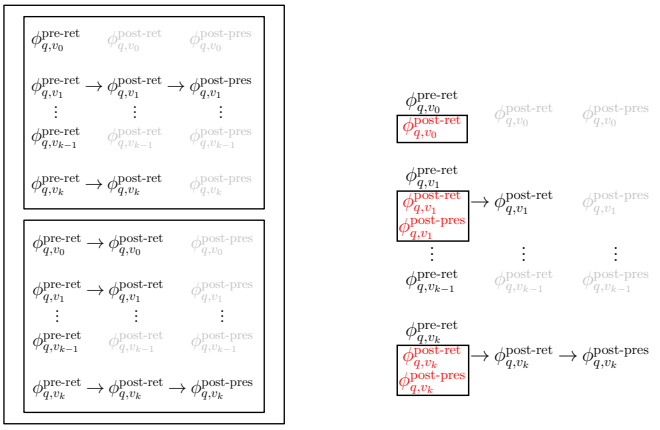
FIGURE 1.2: Feature availability during different stages of decision-making. Pre-retrieval features, $\phi^{\text{pre-ret}}_{q,v}$, are easy to compute, available to all verticals, and help with vertical selection decisions. Post-retrieval features, $\phi^{\text{post-ret}}_{q,v}$, are computed only for those verticals which we request results from and are useful for making final presentation decisions. Post-presentation features, $\phi^{\text{post-pres}}_{q,v}$, are available only after the presentation decision has been made and is useful for evaluating performance and predicting the values of post-presentation features for future issuances of q. Features in gray are not computed or logged during an individual search session because of upstream decisions.

With respect to the second dimension (i.e., the stage at which the feature value can be generated), there are also three categories. *Pre-retrieval features* can be generated without ever issuing the query to the vertical. Query features (generated from the query, independently from the vertical) and vertical features (generated from the vertical, independently from the query) tend to be pre-retrieval features. In most commercial environments, it is either impractical or impossible to issue every query to every vertical in order to decide which verticals to present and where. Thus, pre-retrieval features have received considerable attention in prior work. *Post-retrieval features* must be generated by issuing the query to the vertical. Examples may include the average recency of the vertical's top results or the average retrieval score of the vertical's top results. Post-retrieval features are motivated by the fact a vertical can be ultimately suppressed in light of poor post-retrieval evidence. For instance, if a vertical search engine retrieves an unusually low number of results, this may indicate that it is non-relevant. Finally, *post-presentation* features are observed after the vertical is presented to a user and are typically derived from actions taken by the user. Post-presentations features typically consist of implicit feedback signals such as clicks and skips on the vertical results, which are thought to be positively or negatively correlated with vertical relevance. Post-presentation features are *retrospective*. As such, they allow self-assessment of presentation decisions and can be used to inform predictions during future impressions of the same query or similar queries. Figure 1.2 depicts the availability of different features at different stages of decision-making.

Defining features as being pre-retrieval, post-retrieval, or post-presentation deserves additional clarification. One could imagine, for example, a system that caches post-retrieval and post-presentation features for future impressions of a query (Figure 1.3). Caching feature values is particularly useful for head queries, which are seen over and over. Caching post-retrieval and post-presentation feature values enables their use without having to issue the query to the vertical or having to present the vertical to the user. However, based on the terminology used in this chapter, cached features are still considered post-retrieval and post-presentation features. The distinction lies in whether it is necessary to issue the query to the vertical or present the vertical to the user in order to generate the *exact* feature value for the current query. Vertical collections, vertical query-streams, and vertical click behaviors are dynamic. Thus, while post-retrieval and post-presentation features can be cached for future impressions of a query, their values are likely to require periodic, and possibly online, updating.

1.2.2 Query Features

Query features are generated from the query string and not from any resource associated with a candidate vertical. As described in more detail in Section 1.3, a predictive relationship between query features and the relevance



(a) cached feature values (b) runtime feature values

FIGURE 1.3: Feature caching. The left subfigure represents the logged features computed for previous issuances of q, including post-presentation features. The right subfigure represents the availability of cached feature values (red, in boxes) during runtime at the pre-retrieval stage. Note that these are approximations to downstream feature values and may be sensitive to corpus change or user differences.

of a particular vertical must be either hardwired manually or learned using supervised machine learning.

Prior work has experimented with different types of query features. These include features that indicate the presence of a particular term (e.g., "photo", "weather", "stocks") [5, 26], a particular named entity type (e.g., the name of a person, location, organization) [5], or a particular entry in a look-up table (e.g., a valid zip code or a valid ticker symbol) [5]. Such query features are typically binary-valued and implemented using rule-based triggers.

Query-term features such as those described above capitalize on the fact that certain keywords confidently predict vertical intent. The keyword "pics" suggests that the user wants images, the keyword "showtimes" suggests that the user wants movie times, and the keyword "directions" suggests that the user wants driving directions. Sometimes, however, a query may not have an explicit keyword. Thus, another type of query feature measures the degree of co-occurrence between the query terms and a set of manually-identified vertical-specific keywords [3]. Co-occurrence statistics can be derived from the aggregated search system's own query-log. So, for example, given the query "eiffel tower", the system might consider presenting the *images* vertical because "eiffel tower" tends to co-occur with terms such as "pics", "pictures", and "photos" in the system's query-log.

Among the most successful query features investigated in prior work are query-category features, which measure the query's affinity to a set of predefined semantic categories [5, 30]. Query-category features have been successful for several reasons. First, many verticals investigated in prior work have been topically focused, for example *travel*, *health*, *games*, *music*, *autos*, and *sports* [5]. Second, query-classification has been widely studied for purposes other than aggregated search (see, for example, [37, 38, 21]). Thus, aggregated search techniques can capitalize on these well-tested approaches for the purpose of feature generation. Finally, while general-purpose query-categorization may require making a binary decision about the query's membership to a category, for the goal of aggregated search, this is not required. Query-category features can be real-valued and be associated with the classifier's confidence values across the set of target categories. This gives the aggregated search model the flexibility of focusing on other types of features when query-category confidence values are *uniformly* low.

Query categorization is challenging because state-of-the-art classifiers tend to use a bag-of-words representation and queries are usually terse. However, a simple and effective solution is to categorize the query *indirectly*. That is, to issue the query to a search engine containing manually or even automatically labeled documents, and to categorize the query based on the categories assigned to the top-ranked results [37, 38]. Let $\mathcal{C}$ denote a set of predefined semantic categories (e.g., travel, health, sports, arts, movies, etc.). Given query q, the confidence value assigned to category $c \in \mathcal{C}$ can be computed based on the average confidence value associated with the top-N results ($\mathcal{R}_N$) weighted by their retrieval score,

$$P(c|q) = \frac{1}{\mathcal{Z}} \sum_{d \in \mathcal{R}_N} P(c|d) \times \text{score}(d, q) \tag{1.1}$$

where $P(c|d)$ is the confidence value that document d belongs to category c, $\text{score}(d, q)$ is the retrieval score given to d in response to q, and normalizer $\mathcal{Z} = \sum_{c \in \mathcal{C}} P(c|q)$.

1.2.3 Vertical Features

Vertical features are generated from the vertical, independent of the query. Therefore, their values are the same for all queries. There are two motivations for using vertical features in a model. First, some verticals are more popular than others either because they satisfy a wider range of information needs (*news* vs. *stock quotes*) or because they satisfy more frequently occurring information needs (*weather* vs. *calculator*). Second, user demand for the same vertical is dynamic. A popular news event may trigger a burst in demand for the *news* vertical, unusual weather may trigger a burst in demand for the *weather* vertical, and a viral video may trigger a bursts in demand for the *video* vertical.

As described in more detail later in Section 1.3, there are two ways that a model can learn to favor some verticals irrespective of the query. The first way is to learn different models for different verticals. Most machine learning algorithms harness information about the target class distribution in the training data. The second way is to learn a single model for all verticals, but to add a set of features that signal the identity of the vertical in question. This can be done, for example, by adding one binary feature per candidate vertical. Then, given a query and a particular candidate vertical, the binary feature corresponding to that candidate can be set to '1' and the features corresponding to the other verticals can be set of '0'.

In general, modeling bursts in vertical demand can be done in two ways. One way is to generate features directly correlated with vertical demand. To this end, one might generate features from the vertical's direct query traffic (provided, of course, that the vertical has direct search capabilities), or from clicks on recent presentations of the vertical (provided, of course, that the vertical is click-able). Detecting "bursty states" in a stream of events is a well-studied problem (see, for example, Kleinberg [22]) and such methods can be used to generate features that quantify the current demand for a particular vertical. The other method for modeling content demand is to model content supply, which may be correlated with demand. Such features might include the number of new results added to the vertical collection (e.g., *news*) or whether the vertical contains up-to-date information (e.g., *weather*).

1.2.4 Vertical-Query Features

Vertical-query features measure relationships between the vertical and the query and are therefore unique to the vertical-query pair. Vertical-query features can be classified into pre-retrival, post-retrieval, and post-presentation features.

1.2.4.1 Pre-retrieval Vertical-Query Features

Though it may seem counter intuitive, it is possible to generate vertical-query features without actually issuing the query to the vertical. One alternative is to generate vertical-query features from the vertical's query traffic. Such features consider the similarity between the query and those issued directly to the vertical by users. A simple similarity measure that has been effective in prior work is the query generation probability given the vertical's query-log language model [5, 16, 6]. Let θ_v^{qlog} denote a language model constructed from vertical v's query-log. The query generation probability is given by,

$$P(q|\theta_v^{\text{qlog}}) = \prod_{w \in q} P(w|\theta_v^{\text{qlog}}) \tag{1.2}$$

Because queries have different lengths, it becomes important to normalize this value by $\mathcal{Z} = \sum_{v \in \mathcal{V}} P(q|\theta_v^{\text{qlog}})$.

The above approach measures the similarity between the query and those issued to the vertical. A simpler alternative is to require an exact match. Diaz [15] used features that considered the proportion of vertical query-traffic that corresponded to query q.

Other pre-retrieval features consider the similarity between the query and content from the vertical. As previously mentioned, aggregated search is related to *federated search*, where the objective is to provide integrated search across multiple *text-based* collections. It is oftentimes impractical to issue the query to every collection. Thus, the goal of *resource selection*, which is analogous to *vertical selection*, is to predict the existence of query-related content in a particular collection without issuing the query to the collection. Many different resource selection methods have been proposed in prior work. Two highly successful ones, which have been used as features during aggregated search, are CORI [9] and ReDDE [40].

Both resource selection methods use sampled documents. In our case, if a particular vertical collection is accessible only via a search interface, query-based sampling [10] can be used. Query-based sampling is the iterative process of issuing a random query to the search engine, sampling documents from the results, selecting a new term from the current document sample to be used as new sampling query, and continuing until a large enough sample is collected. Usually, between 300-1000 documents are sampled from each collection.

Of the two methods, CORI is perhaps the simplest. Let v_s denote the set of documents sampled from vertical v. Given k candidate verticals, CORI first constructs an index of k large documents, where the large document associated with vertical v is a concatenation of all documents in v_s. Then, given query q, CORI scores each vertical v according to the retrieval score associated with its large document. So, for example, under the query-likelihood retrieval model, the CORI score given to v in response to q would be,

$$\phi_{q,v}^{\text{cori}} = \prod_{w \in q} P(w|\theta_{v_s}) \tag{1.3}$$

where θ_{v_s} is the language model associated with v_s.

CORI tends to favor collections with the greatest *proportion* of query-related documents (presumably reflected in the sampled set). However, what we really want are the verticals with the greatest *absolute* number of query-related documents. ReDDE directly estimates the absolute number of query-related documents in a collection. Instead of turning each set v_s into a large document, ReDDE constructs an index of all individual documents within all sets of samples. This is referred to as the *centralized sample index*. Then, given query q, ReDDE conducts a retrieval from this index and scores vertical

v according to,

$$\phi_{q,v}^{\text{redde}} = \frac{|v|}{|v_s|} \times \sum_{d \in \mathcal{R}_N} \mathcal{I}(d \in v_s) \tag{1.4}$$

where $\mathcal{R}_N$ denotes the top N results retrieved from the centralized sample index in response to q. ReDDE has an intuitive interpretation. Every document within $\mathcal{R}_N$ that originates from v, represents $\frac{|v|}{|v_s|}$ *unobserved* query-related documents in v. Thus, the summation corresponds the estimated total number of query-related documents in v.

1.2.4.2 Post-retrieval Vertical-Query Features

In contrast with CORI and ReDDE, which predict the existence of relevant content without issuing the query to the vertical, post-retrieval features focus on the quality of the *actual* vertical results. Within IR, *retrieval effectiveness prediction* is the task of assessing the quality of a retrieval without human intervention, for example, based on observable properties of the top-ranked documents. Several methods for retrieval effectiveness prediction have been proposed. Such methods can be used to produce features that predict the quality of the vertical results. Based on these features, if the vertical results are predicted to be bad, a system might decide to suppress.

Prior work on aggregated search investigated a retrieval effectiveness measure known as Clarity [13]. Clarity makes the assumption that in an effective retrieval, the top results should look different from the "average" document within the collection. Based on this assumption, Clarity [13] uses the Kullback-Leibler divergence to measure the difference between the language of the top N documents and the language of the entire vertical,

$$\phi_{q,v}^{\text{clarity}} = \sum_{w \in q} P(w|\theta_q) \log \left(\frac{P(w|\theta_q)}{P(w|\theta v)} \right), \tag{1.5}$$

where θ_q and θ_v are the query and vertical language models, respectively. The query language model can be estimated using the top N vertical results,

$$P(w|\theta_q) = \frac{1}{\mathcal{Z}} \sum_{d \in \mathcal{R}_N} P(w|\theta_d) P(q|\theta_d), \tag{1.6}$$

where $P(q|\theta_d)$ is the query likelihood score given d and $\mathcal{Z} = \sum_{d \in \mathcal{R}_N} P(q|\theta_d)$. Arguello *et al.* [5] used Clarity features for vertical selection. Different from the description above, however, the retrieval from the vertical was produced locally using sampled documents.

Other, perhaps simpler features that attempt to capture the quality of the vertical results are possible. A simple heuristic is to include the hit count (i.e., the number of results retrieved from the vertical). Notice that this is not necessarily equal to the number of results blended into the web results if

the vertical is presented. A vertical with an unusually low number of results is probably not relevant. The hit can be particularly informative when the vertical collection is highly dynamic. In prior work, Diaz [15] use the hit count as a feature in predicting *news* vertical relevance.

Alternatively, we can also include features that are highly vertical-specific and relate to the kinds of things users expect from good results. For instance, *news* results should be recent and from a reliable news source, *local* results should be about business that are nearby, *image* results should have a high picture quality, *video* results should have lots of views, *stock quote* and *weather* results should be up-to-date, and, finally, *shopping* results should be priced attractively. The possibilities are here are vast. For example, any feature used by a vertical-specific ranker could potentially be included as a post-retrieval feature. These feature characterize the quality of individual results. Thus, in cases where a vertical retrieves more than one result (e.g., not *weather* nor *finance*), these features can be compute by taking the average, minimum, and maximum value from the vertical's top N results.

Finally, for users to interact with vertical results, they must be perceived as relevant. Thus, prior work has also considered features generated from the surrogate representation of the vertical results. Examples include number of query terms appearing in the surrogate title and the summary snippet or any other textual element of the surrogate representation [3]. Again, these features require combining the different values from the multiple results in the blended display. One possibility is to use the average, maximum and minimum value.

1.2.4.3 Post-presentation Features

Post-presentation features are derived from user actions on previous presentations of the vertical. These can be derived either from previous presentations of the vertical for the *same query* (potentially from other users), previous presentations of the vertical for *similar queries* (potentially from other users), and from previous presentations of the vertical for *same-session queries*.

The most commonly used post-presentation feature is the vertical-query *click-through rate* [31, 29]. Let C_q^v denote the number of times vertical v was presented for query q and the user clicked on it, and let S_q^v denote the number of times v was presented for q and the user did not click on it. The vertical-query click-through rate is given by,

$$\phi_{q,v}^{\text{click}} = \frac{C_q^v}{C_q^v + S_q^v} \tag{1.7}$$

The main limitation with the above formulation of click-through rate is that it requires an exact match between the current query and a previously observed query. Suppose a user issues a *previously unseen* query "ny style pizza" and the system needs to decide whether to present *local* results. While the current query has no click-through rate, if it did , it would probably have a click-through rate similar to the query "new york style pizza". For the purpose

of feature generation, there are two ways to exploit click-through information from similar queries. One way is to take the average click-through rate from all previous queries weighted by the query-query similarity,

$$\phi_{q,v}^{\text{sim-click}} = \frac{1}{\mathcal{Z}} \sum_{q'} \text{sim}(q, q') \times \phi_{q,v}^{\text{click}} \tag{1.8}$$

where q' denotes a previously seen query and $\mathcal{Z} = \sum_{q'} \text{sim}(q, q')$.

A second way of harnessing click-through evidence from similar queries is to build a language model from all previous queries associated with a click on the vertical (allowing duplicates queries) and then to compute the query-generation probability given this language model [3],

$$\phi_{q,v}^{\text{click-lm}} = \prod_{w \in q} P(w|\theta_v^{\text{click}}) \tag{1.9}$$

where θ_v^{click} denotes the language model associated with the clicks on vertical v. Again, because queries have different lengths, it becomes important to normalize this value by $\mathcal{Z} = \sum_{v \in \mathcal{V}} \prod_{w \in q} P(w|\theta_v^{\text{click}})$.

Beyond click-through information, mouse movement can also be used as a source of post-presentation evidence. One challenge behind interpreting click-through data is that non-clicks do not necessarily indicate non-relevance. It may be that the user did not notice the vertical. Maybe the vertical was presented below the fold or maybe the user was satisfied by a result presented above it. Previous work shows that mouse hovers are correlated with eye gaze [19]. To date, mouse movement features have not been used in published work on vertical selection or presentation. However, one could imagine features that measure the number of times users hovered over previous presentations of the vertical, but did not click on it. If effective, this type of evidence could be also extended to similar queries as described above.

1.2.5 Implementation Details

At this point, it is important to take notice of two nuances associated aggregated search features. First, not every feature will be available for every vertical. Consider for example, ReDDE features, which use sampled vertical content to predict the quantity of query-related content in a vertical collection. A ReDDE feature does not make sense for a vertical that is not associated with a document collection (e.g.,*calculator*). Likewise, some verticals do not have direct search capabilities (e.g., *weather*). Thus, these verticals would not have features derived from the vertical query-stream. As discussed in the next section, methods for combining evidence must accommodate the fact that different verticals will be associated with difference sets of features. Second, some features (particularly query features) are likely to be correlated with relevance differently for different verticals. Consider, for example, a feature

that indicates whether the query contains the term "news". This feature is likely to be positively predictive of the *news* vertical, but negatively predictive for the *recipes* vertical. Thus, methods for combining evidence are likely to be more effective if they can learn a vertical-specific relation between features and vertical relevance.

Aggregated search requires combining *many* different types of features. In this section, we considered features derived from the query string, from the vertical query-log, from sampled vertical content, from the actual vertical results, and from previous presentations of the vertical. In the next section, we present an overview of methods for combining features for making predictions.

1.3 Combination of Evidence

In Section 1.2, we described several signals believed to be correlated with vertical relevance. In this section, we will describe how to combine those signals to make vertical selection and vertical presentation decisions.

1.3.1 Vertical Selection

The vertical selection task refers to picking those verticals likely to appear on a good aggregated search results page. We can rephrase the task as, given a query and a vertical, predict whether should be included on the search results page. Formally, let $\mathcal{V}$ be the set of k candidate verticals. We would like to learn a function that maps a query-vertical pair to relevance, $f : \mathcal{Q} \times \mathcal{V} \rightarrow \mathbf{R}$. Throughout this section, in order to maintain a general perspective, we will remain agnostic about what we mean by functions, queries, and relevance. Nevertheless, when necessary, we adopt certain definitions which admit experimentally-supported approaches.

1.3.1.1 Training Data

All of our algorithms require *training data* encoding examples of relevant and non-relevant query-vertical pairs, (q, v). There are two general methods for gathering training data. In a laboratory setting, assessors are provided with detailed relevance guidelines and then asked to judge the relevance of individual (q, v) pairs. The assessment might be query-driven (e.g. 'for each query, request relevance judgments for all verticals') or vertical-driven (e.g. 'for each vertical, request relevance judgments for many queries'). In query-driven assessment, editors are explicitly more aware of competing user intents and may make judgments different than with vertical-driven assessment. In either case, query sampling is crucial to training a model which can compute

a reliable relevance grade for individual verticals and a calibrated relevance grade for comparison across verticals. In order to provide quality control, (q, v) pairs are usually judged by multiple assessors. Editorial assessment requires some overhead in terms of time (e.g. recruiting assessors, drafting guidelines, verifying labels) and money (e.g. paying assessors). Although crowd-sourcing systems address some of these costs, editorial assessment also suffers because judgments are requested outside of a search context, resulting in the relevance judgments only as good as the assessor's guess of the user intent. For example, local intent queries almost always require an assessor with some familiarity about the hypothetical user's location; a naïve crowd-sourced system may ask for a judgment from an assessor in a different city or country.

If we have access to a production aggregated search system, we can gather labels through *implicit feedback*. For example, if we have access to log data that includes vertical result presentation, we can hypothesize that a user click on a vertical result implies relevance. This allows us to collect a large set of tuples with no editorial cost. Data from implicit feedback carries its own issues. First, the labels are subject to bias effects from the layout constraints (i.e. not every vertical can be displayed for every query to the system), position (i.e. items higher in the page receive more clicks) or presentation (i.e. presentations such as images attract more clicks than other presentations). Furthermore, a click does not necessarily imply relevance; a user may have clicked accidentally or not found what they were looking for from the click. This effect can be mitigated by the large amount of data gathered in production systems.

Regardless of the source of training data, throughout this section, we use the notation $\mathcal{D}$ to refer to the multiset of judged query-vertical pairs. Let the multisets $\mathcal{D}^+$ and $\mathcal{D}^-$ partition $\mathcal{D}$ into relevant and nonrelevant instances respectively.

1.3.1.2 Basic Models

There are several ways we might construct the function f using $\mathcal{D}$. Perhaps the most straightforward way would be to manually enumerate which query-vertical pairs are likely to be relevant,

$$f_{\text{MLE}}(q, v) = \frac{\left|\mathcal{D}^+_{(q,v)}\right|}{\left|\mathcal{D}_{(q,v)}\right|} \tag{1.10}$$

where $\mathcal{D}_{(q,v)}$ is the subset of $\mathcal{D}$ containing (q, v). In other words, we are counting the fraction of presentations for that query which resulted in a relevant judgment with v. We will refer to this as the *maximum likelihood estimate* of relevance for a query-vertical pair. Unfortunately, because we can never judge *every* query—even in a production setting—most of the queries we want to be able to make predictions on will not be in $\mathcal{D}$. Even if we gather a large

amount of data in a production setting, the skewed query frequency distribution suggests that we will observe only a handful of impressions for most queries, resulting in a poor estimation of the relevance.

We can improve the coverage of the maximum likelihood estimate by exploiting the topical similarity between queries. Because queries can be represented as very sparse term vectors, any of the standard term-based similarity measures can be used [34]. Queries specifically have been studied in the literature and admit unique similarity measures [28, 33, 47]. Given such a similarity measure, we can compare a new query to those queries we have previously judged. For example, we can define a function,

$$f_\kappa(q, v) = \sum_{(q',v)\in\mathcal{D}} \kappa(q, q') f_{\text{MLE}}(q', v) \tag{1.11}$$

where $\kappa(q, q')$ is the similarity between two queries. In other words, we are smoothing the estimates from the maximum likelihood estimate using similar queries [14]. We refer to this as the *kernel estimate* of relevance for a query-vertical pair. In this case, our coverage increases to include similar queries. However, depending on the robustness of our similarity measure, this may not be sufficient. For example, assume that we have hired editors to manually generate a set of relevant queries for an image vertical. If our image vertical contains a large number of cat pictures but editors did not manage to label 'cat' as a relevant query for the image vertical, our feline-inclined users will not discover these fascinating pictures. Therefore, it is important to understand that the effectiveness of f_κ is the topical nature of the query similarity. Nevertheless, techniques based on query similarity have been found to be a strong baseline [26].

We can address the issues with query similarity by using the signals described in Section 1.2. Specifically, our approach will be to learn f as a function from features of the query-vertical pair to relevance. That is,

$$f_\theta(q, v) = g(\phi_{q,v}, \theta_v) \tag{1.12}$$

where $\phi_{q,v}$ is the $m \times 1$ vector of m feature values described earlier and θ_v is a vector model parameters for that specific vertical. We have a set of model parameters for each vertical because the relationship between a feature and relevance is likely to be dependent of the vertical (e.g. the feature 'query contains *picture*' is likely to be positively correlated with the image vertical but negatively with other verticals). We suspect that similar queries will have similar feature vectors; therefore, the behavior of f_κ is preserved. In addition, though, we can generalize to those queries with similar feature vectors but with potentially different topics. The function g has two arguments: features and model parameters. Features are often encoded as a vector of scalar and binary values according to the definitions in Section 1.2. We might use a subset of features, for example only the pre-retrieval features, or all features, depending on the cost or availability. The model parameters represent the knobs used

to tune the model, given some training data. The precise definition of θ_v depends on the functional form of g. If we are using logistic regression for g, then θ_v is a length m vector of feature weights and our functional form is the inner product between $\phi_{q,v}$ and θ_v,

$$g_{\log}(\phi_{q,v}, \theta_v) = \frac{\exp\left(\phi_{q,v}^{\mathsf{T}} \theta_v\right)}{1 + \exp\left(\phi_{q,v}^{\mathsf{T}} \theta_v\right)} \tag{1.13}$$

Given a functional form, we tune θ_v such that classification error on our training data is minimized. In the case of logistic regression, we minimize the logistic loss using an iterative technique such as the Newton-Raphson method. Many other functional forms exist in the machine learning literature. A full review is beyond the scope of this chapter.

Moving forward, we will adopt probabilistic versions of f. That is, we will adopt the notation,

$$p^{\beta}_{(q,v)} = f(q, v) \tag{1.14}$$

to indicate that a score for a query-vertical pair (q, v) can be interpreted as a probability of relevance. More concretely, the probability p^{MLE} is the maximum likelihood probability of relevance, p^{κ} is the kernel density estimate of relevance, and p^{θ} is a parametric model of relevance. In principle, the techniques in remainder of the chapter can be extended to non-probabilistic paradigms.

1.3.1.3 Advanced Models

Each of the models presented in the previous section has advantages and disadvantages: p^{MLE} may be effective for judged query-vertical pairs but the coverage is low, p^{κ} does not generalize to topics outside of $\mathcal{D}$, and p^{θ} may not capture topical similarity. In this section, we present one way of combining these approaches in a single framework [15].

Although p^{MLE} suffers from low coverage, for those queries where we have reliable judgements, performance is strong; Equation 1.10 becomes more accurate with increasing data. On the other hand, our machine learned estimate p^{θ} provides an effective method when there is no data but does not change observed judgments (e.g. if the system presents that vertical to real users). One way to combine these two method is with Bayesian updating. The intuition with Bayesian updating is that, in the absence of (enough) data for a specific query, we can use p^{θ} to estimate the relevance; after observing judgments from editors or from query logs, we can adjust this estimate. So, instead of modeling the precise probability of relevance, we are estimating a *distribution* over the value of the probability of relevance. Let p^{β} be this distribution. If we assume that the parametric form of this distribution is the Beta distribution, then we can define it as,

$$p^{\beta}_{(q,v)} \sim \text{Beta}(a, b) \tag{1.15}$$

where we set the Beta parameters such that,

$$a = \mu p^{\theta}_{(q,v)} \qquad b = \mu \left(1 - p^{\theta}_{(q,v)}\right) \tag{1.16}$$

where μ is a hyperparameter of our model. That is, absent any data, the distribution over the probability of relevance is strictly a function of the $p^{\theta}_{(q,v)}$ and μ.

Assume we have observed some positive and negative feedback for a query-vertical pair. Then, the posterior, given this data is also a Beta distribution,

$$p^{\beta}_{(q,v)}|\mathcal{D} \sim \text{Beta}\left(a + \left|\mathcal{D}^{+}_{(q,v)}\right|, b + \left|\mathcal{D}^{-}_{(q,v)}\right|\right) \tag{1.17}$$

And the posterior mean,

$$\hat{p}^{\beta}_{(q,v)} = \frac{|\mathcal{D}^{+}_{(q,v)}| + \mu p^{\theta}_{(q,v)}}{|\mathcal{D}_{(q,v)}| + \mu} \tag{1.18}$$

Note here that we can gain an intuition for μ. For small values of μ, the model will be very sensitive to early feedback from the user. For large values, the model will rely on $p^{\theta}_{(q,v)}$ more than feedback.

The hypothesis underlying the kernel density estimate was that a query's probability of being relevant is related to the relevance of topically related queries. We can incorporate information from related queries as pseudo-judgments on the candidate query. Specifically, we can define the aggregated information for a query as,

$$\tilde{\mathcal{D}}^{+}_{(q,v)} = \mathcal{D}^{+}_{(q,v)} + \sum_{q'} \kappa(q, q')\mathcal{D}^{+}_{(q',v)} \tag{1.19}$$

$$\tilde{\mathcal{D}}^{-}_{(q,v)} = \mathcal{D}^{-}_{(q,v)} + \sum_{q'} \kappa(q, q')\mathcal{D}^{-}_{(q',v)} \tag{1.20}$$

We can use these modified counts in the same way we used the original counts in Equation 1.18.

1.3.2 Vertical Presentation

Vertical presentation refers to deciding precisely where to place relevant verticals on the search results page. For simplicity, we constrain our discussion to a ranked list of web documents and interleaved with vertical results.

1.3.2.1 Pointwise Interleaving

According to Robertson, any ranked list of results to a user query should be ordered according each item's probability of relevance [32]. Since an aggregator interleaves vertical result blocks with web documents, we can assert that the presentation problem reduces to estimating the probability of vertical

relevance (Section 1.3.1.3). Let $p_{(q,d)}$ be the probability that web document d is relevant to query q. If this probability is defined for all documents and we use a probabilistic vertical selection algorithm, then we can interleave items by the probabilities. The rank of v in the interleaved list is,

$$\text{rank}_{(q,v)} = \left|\{d \in \mathcal{W} : p_{(q,d)} > p_{(q,v)}\}\right| + \left|\{v' \in \mathcal{V} : p_{(q,v')} > p_{(q,v)}\}\right| \quad (1.21)$$

where $\mathcal{W}$ is our collection of web documents. Because we are interested in estimating the probability of relevance of each vertical and document, we refer to this as *pointwise interleaving*.

Unfortunately, the assumptions underlying probabilistic interleaving are rarely satisfied. The scores returned by web rankers are not guaranteed to be probabilities. For example, many modern 'learning to rank' approaches rank documents by an arbitrary scalar value output by the ranking model. Even if web document scores are probabilities, the values are unlikely to be well-calibrated with the vertical selection scores, resulting in poor interleaving [18].

One strategy for comparing $p_{(q,v)}$ to a document retrieval score is to transform the original document score output by the web search engine to a probability. If there are relevance labels available for (q, d) pairs, then we can learn a statistical model to do this transformation [1]. Specifically, we can learn a logistic regression model in the same way as in Equation 1.13,

$$h_{\log}(\phi_{q,d}, \theta) = \frac{\exp\left(\phi_{q,d}^{\mathsf{T}}\theta\right)}{1 + \exp\left(\phi_{q,d}^{\mathsf{T}}\theta\right)} \quad (1.22)$$

In this case, the features would, at a minimum, include the document score. Although we can add other features of the document, there is the temptation to begin adding large numbers of features to this model. There are two reasons to be cautious with feature addition. First, we want to avoid rebuilding a web ranking algorithm; we only need to calibrate a score which, conceivably, has been independently and rigorously tested to predict relevance, albeit on a different scale. Second, feature computation, especially for web documents can be expensive if, for example, the value is stored in an index on a separate server. Furthermore, there is bandwidth overhead for the web ranker to expose the entire feature vector used for core ranking.

In cases where interleaving decisions need to be made without knowledge of the web document ranking scores, we can use position-based models to make interleaving decisions [30]. A position-based interleaving model is similar to the score transformation described in Equation 1.22 except our only feature is the position of the document. As a result, we predict a fixed probability of relevance for each position in the ranking, regardless of the documents or their scores. Given probabilistic vertical selection scores, the vertical presenter can

position each vertical according to,

$$\text{rank}_{(q,v)} = \sum_{i=0}^{|\mathcal{W}|} I(p_i > p_{(q,v)}) + \left|\{v' \in \mathcal{V} : p_{(q,v')} > p_{(q,v)}\}\right| \tag{1.23}$$

where p_i is the predicted probability of relevance of rank position i.

1.3.2.2 Pairwise Interleaving

Given an accurate and calibrated probability of relevance for each document and vertical, pointwise interleaving should provide an optimal ranking. In reality, estimating the true probability of relevance is very difficult. In fact, estimating the probability of relevance may be an inappropriate target if we are only interested in the relative relevance of verticals and documents. The user is never exposed to the absolute probability of relevance and is only concerned that more relevant items are places above less relevant items. Therefore, we may want to focus our modeling effort on predicting the preference users have between pairs of items given a query. Modeling the order of items instead of the absolute relevance is precisely the goal of *pairwise interleaving*.

Although the optimization target for pairwise interleaving is different than for pointwise interleaving, many of the fundamentals are the same. Specifically, we are still interested in learning a function f_{LTR} with two differences. First, the domain includes the 'web result' argument for determining the preference between a vertical and a web result. This is similar to the task of estimating $p_{(q,d)}$ in the previous section. Second, for pairwise models, our range is the set of the reals. That is, we are only interested in a function that outputs a real value, unconstrained by modeling the exact probability of relevance. Our objective is to find f_{LTR} such that, given a query, the values for more relevant verticals or documents are larger than the values for less relevant verticals. The precise training of such models is studied in the field known as 'learning to rank' (LTR) [27].

Casting pairwise interleaving as a learning to rank problem requires training a *single* model f_{LTR} to predict a item's rank irrespective of its type (e.g. image, local, web result). In our situation, this is problematic because different item-types are associated with different features (i.e., some features may be specific to a handful of types and some may be unique to a particular one). In addition, it is problematic because those features that are common to *multiple* types (e.g., whether the query contains a city name) may be predictive for some types more than others, or even predictive for different types in the opposite direction. Next, we propose three LTR variants which address these challenges in different ways. Each variant makes a different assumption about how features may be correlated with item relevance across item-types.

1.3.2.2.1 Equally Correlated Features One alternative is to assume that each feature is equally predictive of item relevance (in the same direction)

independent of the item-type.

$$\phi_{q,v}^{\text{shared}} = \left[\begin{matrix} \phi_{q,v}^{\text{pre-ret}} \\ \phi_{q,v}^{\text{post-ret}} \end{matrix}\right] \Big\} \text{shared features} \tag{1.24}$$

where $\phi_{q,v}^{\text{pre-ret}}$ is a column vector of pre-retrieval features and $\phi_{q,v}^{\text{post-ret}}$ is a column vector of post-retrieval features. The feature representation is as follows. Pre-retrieval features, $\phi_{q,v}^{\text{pre-ret}}$, are independent of the item. This model uses a *single* copy of each pre-retrieval feature. The values of post-retrieval features are item-specific (i.e., they are generated directly from the item or the item's search engine results). As with pre-retrieval features, this approach *also* uses a *single* copy of each post-retrieval feature in the sub-vector $\phi_{q,v}^{\text{post-ret}}$. If a item is not associated with a particular post-retrieval feature, then the feature is zeroed-out in that instance. Consider, for example, our post-retrieval features which determine the text-similarity between the query and the summary snippets presented in the item. These features may only be associated with *news* and web result items. Therefore, if the item is not one of these types, all of these features are zeroed-out. This approach assumes that features are equally correlated with relevance irrespective of the item-type. Once trained, model f_{LTR} will apply the *same* parameters to a feature independent of the instance's item-type.

1.3.2.2.2 Uniquely Correlated Features We can also assume that every feature—whether it is a pre- or post-retrieval feature—is uniquely correlated with relevance across different item-types. The feature representation is as follows. We make a separate, item-type-specific copy of each feature. So, for example, given $|\mathcal{V}| + 1$ item-types, we make $|\mathcal{V}| + 1$ copies of each pre-retrieval feature (one per item-type). Given an instance, all copies are zeroed-out except for those corresponding to the instance's item-type. For post-retrieval features, we make one copy per item-type for which the feature is available. That is,

$$\phi_{q,v}^{\text{disjoint}} = \left[\begin{matrix} 0 \\ 0 \\ \vdots \\ \phi_{q,v}^{\text{pre-ret}} \\ \phi_{q,v}^{\text{post-ret}} \\ \vdots \\ 0 \\ 0 \end{matrix}\right] \begin{matrix} \Big\} \text{non-}v \text{ features} \\ \\ \\ \Big\} v \text{ features} \\ \\ \\ \Big\} \text{non-}v \text{ features} \end{matrix} \tag{1.25}$$

Consider, for example, our temporal features, which are available for items from *blogs*, *community Q&A*, *news*, and *twitter*. We make 4 copies of each temporal feature.

This approach assumes that features are correlated differently with relevance depending on the item-type. Once trained, model f_{LTR} will apply a *different* θ subset depending on the instance's item-type. While this added flexibility may be advantageous, the increased number of features may introduce predictive noise and result in overfitting. Thus, this LTR variant may require more training data than the model described in the previous section.

1.3.2.2.3 Equally and Uniquely Correlated Features The previous two approaches make opposite assumptions: features are either equally correlated or uniquely correlated with relevance for different item-types. A third alternative is to make neither assumption *a priori*, but to give the algorithm the freedom to exploit both types of relationships using training data.

For this approach, we maintain a single copy of each pre- and post-retrieval feature which is shared across all item-types. As before, if an instance's item-type is not associated with a shared feature, the feature is zeroed-out for that instance. In addition to these shared features, we make one item-type-specific copy of each pre- and post-retrieval feature. Given an instance, all copies corresponding to types other than the instance's item-type are zeroed-out. That is,

$$\phi^{\text{combined}}_{q,v} = \begin{bmatrix} \phi^{\text{pre-ret}}_{q,v} \\ \phi^{\text{post-ret}}_{q,v} \\ \vdots \\ 0 \\ 0 \\ \vdots \\ \phi^{\text{pre-ret}}_{q,v} \\ \phi^{\text{post-ret}}_{q,v} \\ \vdots \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{shared features} \\ \\ \left.\vphantom{\begin{matrix} a \\ b \end{matrix}}\right\} \text{non-}v \text{ features} \\ \\ \left.\vphantom{\begin{matrix} a \\ b \end{matrix}}\right\} v \text{ features} \\ \\ \left.\vphantom{\begin{matrix} a \\ b \end{matrix}}\right\} \text{non-}v \text{ features} \end{matrix} \tag{1.26}$$

The canonical feature representation for this approach is the union of features used by the previous two approaches.

This approach makes no assumption about how a feature is correlated with relevance across item-types. If a feature is equally correlated across item-types, then the algorithm can assign a large (positive or negative) weight to the copy of the feature which is shared across types. Alternatively, if a feature is correlated differently for different item-types, then the algorithm can assign a large positive weight to some copies of the feature and a large negative weight with to others. Of all three LTR variants, this one has the largest number of features and may therefore need the most training data to avoid overfitting.

1.4 Evaluation

Evaluation is essential to all subfields of information retrieval, and the same is true for aggregated search. In general, the goal of evaluation is to facilitate the objective comparison between different algorithms, different features, and different parameter settings. As previously mentioned, aggregated search is viewed as a two step process: predicting which verticals to present (*vertical selection*) and predicting where in the web results to present them (*vertical presentation*). In some situations, it is desirable to evaluate the vertical selection component in isolation. Given a query, the goal for the vertical selection component to select those verticals that are relevant and suppress those verticals that are not relevant. In other situations, the goal is to evaluate the end-to-end aggregated search solution. In this case, the goal for the system is not only to select the relevant verticals, but to present those verticals that are more likely to be relevant in a more salient way. In practice, this means presenting the most relevant verticals higher in the aggregated results. In the following sections, we present an overview of methods for aggregated search evaluation. First, we focus on vertical selection evaluation and then we focus on end-to-end evaluation.

1.4.1 Vertical Selection Evaluation

Vertical selection is the task of deciding which verticals to present along with the core web results for a given query. From the perspective of evaluation, the best vertical selection system is the one that selects the relevant verticals and avoids selecting the non-relevant ones. In this respect, vertical selection can be evaluated as any other multiclass classification problem, using metrics such as accuracy, which summarizes performance for all verticals, or precision and recall, which summarize performance for each vertical independently.

Let $\mathcal{Q}$ denote the set of evaluation queries and $\mathcal{V}$ denote the set of candidate verticals. Vertical selection evaluation requires knowing which verticals are *truly* relevant to each evaluation query $q \in \mathcal{Q}$. Let $\mathcal{V}_q$ denote the set of verticals that are *truly* relevant to query q and $\tilde{\mathcal{V}}_q$ denote the set of verticals that are *predicted* relevant to q. A commonly-used evaluation metric for vertical selection evaluation is *accuracy*. Given a query q, a vertical selection component must make $|\mathcal{V}|$ predictions. That is, for each vertical v, it must decide whether to present the vertical or to suppress it. Accuracy measures the percentage of correct predictions and is computed as,

$$\mathcal{A} = \frac{1}{|\mathcal{V}| \times |\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \sum_{v \in \mathcal{V}} \Big(\mathcal{I}(v \in \tilde{\mathcal{V}}_q \wedge v \in \mathcal{V}_q) \vee \mathcal{I}(v \notin \tilde{\mathcal{V}}_q \wedge v \notin \mathcal{V}_q) \Big) \qquad (1.27)$$

where $\mathcal{I}$ denotes the indicator function (equals 1 if the its argument is true and 0 otherwise).

Other measures used in multiclass classification are also possible. For example, one can also calculate precision, recall, and f-measure with respect to each vertical, and then possibly average across candidate verticals to obtain a single measure. Let $\mathcal{Q}_v$ denote the set of queries for which vertical v is *truly* relevant and $\tilde{\mathcal{Q}}_v$ denote the set of queries for which vertical v is *predicted* relevant. Precision, recall, and f-measure with respect to vertical v are given by,

$$\mathcal{P}_v = \frac{1}{|\tilde{\mathcal{Q}}_v|} \sum_{q \in \tilde{\mathcal{Q}}_v} \mathcal{I}(q \in \mathcal{Q}_v) \tag{1.28}$$

$$\mathcal{R}_v = \frac{1}{|\mathcal{Q}_v|} \sum_{q \in \mathcal{Q}_v} \mathcal{I}(q \in \tilde{\mathcal{Q}}_v) \tag{1.29}$$

$$\mathcal{F}_v = \frac{2 \times \mathcal{P}_v \times \mathcal{R}_v}{\mathcal{P}_v + \mathcal{R}_v} \tag{1.30}$$

In prior vertical selection evaluations, Arguello *et al.* [5] addressed the task of *single* vertical selection (a more simplified version of the full vertical selection task). During single vertical selection, the goal is to predict a single relevant vertical, if one exists, or to predict that no vertical is relevant. Let $\tilde{v}_q$ denote a system's single vertical prediction and $\tilde{v}_q = \emptyset$ denote the prediction that no vertical is relevant. In this work, accuracy was measured according to,

$$\mathcal{A} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} (\mathcal{I}(\tilde{v}_q \in \mathcal{V}_q \wedge \mathcal{V}_q \neq \emptyset) \vee \mathcal{I}(\tilde{v}_q = \emptyset \wedge \mathcal{V}_q = \emptyset)) \tag{1.31}$$

Li *et al.* [26] compared approaches to vertical selection by constructing interpolated precision-recall (PR) curves for each candidate vertical. Precision and recall for a vertical can be computed as described above and different precision and recall operating points can be derived by sweeping the classifier's prediction confidence threshold. Usually, the higher the threshold, the higher the precision and the lower the recall. This evaluation method has the advantage of providing a more complete picture of the trade off between precision and recall for different approaches.

As previously mentioned, vertical selection evaluation requires knowing which verticals are relevant to each query. Arguello *et al.* [5] and Li *et al.* [26] used trained assessors. Assessors with expert knowledge about the different candidate verticals were given a set of queries (sampled from a commercial query-log) and were asked to determine which verticals, if any, were likely to be relevant to each query. This method of assessment has two potential drawbacks. First, queries are oftentimes ambiguous. Therefore, it may be difficult for an assessor to determine the user's actual intent. Second, the assessments

do not consider the vertical's relevance within the context of the core web results.

Diaz [15] and König *et al.* [23] evaluated vertical selection for the *news* vertical in a production environment. The gold standard data was collected using a commercial system, where a small fraction of query traffic was always presented the *news* vertical above the core web results. Relevance judgements were derived from click-through data, and because the vertical was always presented, all *news* clicks and skips were observed. In other words, retrospectively, the data collection included queries for which the *news* vertical should not be selected based on observed skips. Features were generated and cached to allow repeated experimentation after the data collection phase. Evaluation was conducted using *accuracy*, defined here as the percentage of correctly predicted clicks and skips.

1.4.2 End-to-End Evaluation

Vertical selection evaluation is more clearly defined than end-to-end aggregated search evaluation. Consider, for example, the aggregated results shown in Figure 1.1. The basic end-to-end evaluation questions is: how good are these results? If the system presented *video* results instead of *image* results, would the presentation be better? And, what about more subtle changes? For instance, what if the *local* vertical was presented above the first Web result? Would this be better? Would it really make a difference?

End-to-end aggregated search evaluation falls under three broad categories: test collection evaluation, on-line evaluation, and user study evaluation. Test collection evaluation builds upon the Cranfield IR evaluation paradigm [12]. A test collection typically includes a set of queries, a set of systems (i.e., a set of verticals and core content providers) with responses to those queries, and a set of relevance judgements on the vertical and web results. Given these three components, evaluation measures can be computed on any given aggregation of results. On-line evaluation focuses on implicit user feedback from real users in an operational setting. Implicit feedback measures typically focus on clicks and skips, where clicks are usually treated as positive feedback and skips are usually treated as negative feedback. Finally, user study evaluation involves having users perform search tasks with different aggregated search systems in a controlled environment. Evaluation measures are derived from outcome measures thought to be correlated with a positive user experience or from responses to questions given to study participants.

All three evaluation methodologies have advantages and disadvantages and can be used to answer different types of research questions. Once a test collection is built, evaluation is basically free and results are reproducible. This makes test collection evaluation an attractive alternative for fine-tuning parameters. However, test collection evaluation assumes that collections are static and that relevance judgments from assessors are consistent with those made by real users. On-line evaluation methods use real users in real sit-

uations. However, because evaluation measures are computed using implicit feedback signals, precision can be estimated more accurately than recall. That is, the operational system can observe false-positive mistakes, but not false-negative mistakes. Finally, user study evaluation gives the researcher more control than on-line evaluation. For example, the researcher can learn about participants' backgrounds, can manipulate the higher-level search task, and can manipulate the search context. Such variables cannot be controlled and cannot be easily identified in an on-line setting. On the other hand, user studies are expensive and time consuming. For all these reasons, all three evaluation methodologies are important and all three are necessary to measure improvement and understand user behavior.

1.4.2.1 Test Collection Evaluation

Test collection evaluation follows the Cranfield evaluation paradigm [12] and has the following components: a static collection of retrievable items, a set of queries with topic descriptions that define what should and should not be considered relevant, a set of relevance judgements for all query-document pairs, and a suite of evaluation measures that operate on a ranking of known relevant/non-relevant items. Usually, because collections are large and because most documents are not relevant, it is unnecessary (and often prohibitively expensive in terms of time and effort) for assessors to judge all documents for all queries. Instead, assessors typically judge only those documents most likely to be relevant. These can be determined using a method known as *pooling*. The basic idea is to take the union of top results from a wide range of systems. Documents within this set are judged and documents outside of this set are automatically assumed to be non-relevant.

With regards to aggregated search, test collection evaluation can deviate from the general Cranfield method in one respect. Depending on the formulation of the task, pooling results from the different systems being aggregated (i.e., from the set of candidate verticals and from the core web search engine) may not be necessary. Most approaches to vertical selection and presentation assume that the top results from each vertical (those that would be presented if the vertical were selected) are fixed.

Thus far, two test collection evaluation methodologies have been proposed for aggregated search. The first was proposed by Arguello *et al.* [4] and the second was proposed by Zhou *et al.* [48]. Both methods have things in common. First, both impose constraints on the end-to-end aggregation task. For example, vertical results must be blended into the core web results from top to bottom (i.e., verticals cannot be stacked horizontally) and, if a vertical is presented, then its results must be displayed in sequence. Second, both methods propose ways of collecting assessor relevance judgements on vertical results and on the core web results for a given query. Finally, both methods propose evaluation measures that take as input a set of relevant judgements and can determine the quality of any possible aggregation of results (subject

to the imposed constraints).

The approach from Arguello *et al.* [4] casts the aggregation task as *block ranking*. The first step is to impose a set of layout constraints. Most importantly, if a vertical is presented, then all its results must be presented together and they must be presented within a set of predefined slots, for example, above the first web result, between web results three and four, or below the last web result. Given such constraints, aggregated search can be viewed as a block ranking task, where a block is defined as either a sequence of same-vertical results or a sequence of web results that cannot be split.

Let $\mathcal{B}_q$ denote the set of blocks associated with query q. $\mathcal{B}_q$ includes one block for every vertical $v \in \mathcal{V}$ that retrieves a minimum number of results and one block for every sequence of web results that cannot be split. Following the example constraints from above, Web results 1-3 would form one block and Web results 4-10 would form another block. Additionally, $\mathcal{B}_q$ includes an imaginary "end of SERP" block. Thus, the end-to-end goal for the system is to produce a ranking of all elements in $\mathcal{B}_q$, where the blocks ranked below the imaginary "end of SERP" block are suppressed and effectively tied. Given this formulation of the aggregated search task, the method proposed Arguello *et al.* [4] is to derive an ideal or *reference* block-ranking for each evaluation query q and to evaluate alternative block-rankings for q based on their similarity or distance to the reference.

Given query q, let σ_q^* denote the ideal block-ranking and σ_q denote a predicted block-ranking. Two components are still missing. First, how do we derive σ_q^* and, second, how do we measure the similarity between σ_q and σ_q^*? In response the the first question, Arguello *et al.* [4] collected preference judgements on all block pairs in $\mathcal{B}_q$ and derived the reference presentation σ_q^* by applying the Schulze Voting Method [35] to these preference judgements. In response to the second question, Arguello *et al.* [4] used a variant of Kendall's τ [24]. While these were the particular solutions used in this work, other ways to constructing σ_q^* and other ways of calculating the similarity between σ_q and σ_q^* are also possible.

The evaluation method proposed in Zhou *et al.* [48] takes a different approach. The main evaluation metric is defined as *utility*. Let σ_q still denote the output predicted by the system in response to query q. The output can still be viewed as a ranking of web and vertical blocks. Zhou *et al.* define *utility* as the user's expected *gain* obtained from reading σ_q divided by the expected *effort* expended in reading σ_q,

$$\mathcal{U}(\sigma_q) = \frac{\sum_{b \in \sigma_q} E(b) \times G(b)}{\sum_{b \in \sigma_q} E(b) \times F(b)} \tag{1.32}$$

Now, let us consider the different components of the above equation. Let $b \in \sigma_q$ denote a web or vertical block presented in σ_q. The first component,$E(b)$, is defined as the probability that a user will examine block b. $E(b)$ can be derived in different ways, however, it should be a function of the block's position in

the ranking as well as its visual salience. So, for example, all else being equal, a block of results from the *images* vertical should have a higher examination probability than a block of results from the *news* vertical. Image results are more salient than news results and are therefore more likely to be noticed and examined.

The second component, $G(b)$, is defined as the user's *gain* obtained from examining block b. One important distinction between this approach and the one proposed by Arguello *et al.* [4] is that this approach decouples the relevance of a system (i.e., the relevance of a vertical or the relevance of the web search engine) from the relevance of the results retrieved by that system. Given a query, assessors are first asked judge the relevance of a particular system independent of any results. This is referred to as the query's *orientation*. Then, assessors are asked to judged the items retrieved by each system based on topical relevance. Topically relevance is judged independent of the query's orientation. Finally, the gain associated with block b is given by the product of the total topical relevance of items within b (the sum of topical relevance grades associated the items in b) and the query's orientation with respect to the system that produced b.

The third and final component, $F(b)$, is defined as the amount of *effort* required to assess the relevance of b. The idea here is that different types of results require different amounts of effort. For example, *images* require little effort because the surrogate is a faithful representation of the underlying result. On the other hand, *news* results require more effort because the user must read the summary snippet and possibly even navigate to the article. Finally, *video* results take a significant amount of effort because the user may need to view the entire video to assess its relevance. Thus, blocks from different systems can be assigned different weights based on different heuristics or assumptions.

The approach from Zhou *et al.* [48] is fairly general. One can imagine different browsing models for estimating the examination probability associated with a block ($E(b)$), different ways of measuring gain ($G(b)$), and different assumptions for modeling assessment effort ($F(b)$).

1.4.2.2 On-line Evaluation

On-line evaluation involves testing a system in an operational setting based on implicit user feedback. The basic idea is to have different subsets of users exposed to different systems and to compare between systems based on observable signals that are thought to be correlated with user satisfaction. To date, most on-line evaluations have focused on clicks and skips.

On-line evaluation has two main advantages. First, the evaluation is conducted using real users in real situations. This facilitates the evaluation of methods that provide personalized results to different individuals and methods that harness evidence from the user's context, for example, their location. Second, the evaluation can be conducted using *lots* of people, ensuring that

results generalize across users and situations.

That said, on-line evaluation also has some challenges. First, and most importantly, implicit user feedback is noisy. Users tend to click on results that ranked higher and results that are visually salient, for example, results that show images. Moreover, when users do click, they do so based on *perceived* relevance. The underlying web or vertical result may actually turn out to be non-relevant. Skips are noisy as well. Users may satisfy their information need from the summary snippet and, in fact, some verticals (e.g., *weather*) may not even be click-able. The second challenge is that on-line experiments are not repeatable. The users, the collections, and queries will be different. That is not to say that on-line evaluation results cannot be trusted. The same comparison between approaches can be conducted multiple times to verify that the best system is still the best system. However, the exact outcome measures will be different. The dynamic nature of on-line evaluations make it difficult to do debugging and error analysis, which often require changing one thing at a time.

One approach to on-line evaluation is to measure the *click-through* rate for each candidate vertical independently. Click-through rate answers the following question: of the times the vertical was presented, how often was it clicked? The metric assumes that, if a user did not click on a vertical that was presented, the vertical should not have been presented, or should have been presented in a different location. While click-through rate is an intuitive measure, it paints an incomplete picture. It measures precision, but not recall. Suppose a system only predicts a vertical relevant when it is *very* confident that it relevant. Such a system would probably observe a very high click-through rate. However, what about the false-negatives? In other words, how often did the system suppress a vertical that should have been presented?

To address this limitation, an evaluation based on click-through rate should also consider another measure known as *coverage*. Coverage measures the percentage of queries for which the vertical was presented. Used in conjunction, a greater coverage and a higher click-through rate can be seen as an improvement. It means that the system made fewer false-negative *and* fewer false-positive mistakes. While coverage is not equal recall, the measures are related. High coverage probably means high recall, but low coverage *does not* necessarily mean low recall.

Ponnuswami *et al.* [30] used click-through rate and coverage in conjunction to evaluate an end-to-end system, where verticals could be slotted into one of three positions: above the web results, below the web results, and in the middle. Click-through rate and coverage were measured independently for each vertical-slot pair.

In addition to the challenges mentioned above, on-line evaluation can also be time consuming and expensive. If the goal is to fine-tune an existing system, it may not be possible to conduct an on-line evaluation for every combination of parameter values. To address this limitation, a few recent studies have investigated methods for collecting on-line user-interaction data once and us-

ing this data to perform *multiple* rounds of off-line testing [29, 25]. These methods have some of the benefits of test collection evaluation. Namely, once the data is collected, evaluation is inexpensive and results are reproducible. The basic idea is to collect the user interaction data in a completely unbiased fashion, where every system output is *equally* probable. Then, given a particular model (with a particular parameter configuration), evaluation can be done by considering only the interaction data (e.g., the clicks and skips) associated those outputs that are identical to what the system would have produced given the same query. Results show that metrics computed in this off-line fashion closely approximate those computed in an on-line setting using the same experimental system [29, 25].

1.4.2.3 User Study Evaluation

User study evaluation involves exposing study participants to different systems and measuring their level of task success or satisfaction. Compared to test collection evaluation and on-line evaluation, user study evaluation has a few advantages. First, participants can be asked questions that directly measure their level of satisfaction with the system. The evaluation does not rely on metrics that may be only weakly correlated with user satisfaction. Second, user studies can target specific user populations. For example, one can study differences in search behavior between experienced and inexperienced searchers. Third, because the experiment is conducted in a controlled setting, user studies can manipulate the higher-level task or the search context. Within aggregated search, user studies have been conducted to answer two basic questions: "What do users want?" and "What are the factors that affect their preferences or their behaviors?".

Both test collection approaches discussed in Section 1.4.2.1 were validated by conducting user studies [4, 48]. The goal was to compare the value of the proposed evaluation metric with user preferences between alternative presentations of aggregated results. The basic assumption is that a 'good' metric should be consistent with user preferences. Both user studies consisted of showing participants pairs of aggregated results and comparing the metric score with the stated preference. Several common trends were observed. First, agreement between study participants was low. The Fleiss' Kappa agreement [17], which corrects for the expected agreement due to random chance, was about 20%. Given this low level of agreement between people, it would be unreasonable to expect any metric to agree with a user 100% of the time. Second, the agreement between the metric and the assessors was about 65%, Notice that the expected agreement due to random chance is 50%. Finally, agreement between the metric and the assessors was greater (about 80%) on those presentation pairs where the value of the metric was drastically different. In other words, the proposed metrics were better at distinguishing between good and bad presentations than between pairs of good and pairs of bad presentations. Taken together, these three trends tell us that while

perfect agreement between any metric and users' preferences is unlikely, there is room for improvement.

Preference behavior was also studied by Zhu and Carterette [49]. Again, participants were shown pairs of presentations for a given query and asked to state a preference. Different from the preference-based studies described above, however, this work only looked at the blending of the *image* vertical in different slots. The study found a strong preference for the *image* vertical ranked high for queries likely to have image intent.

Studies have also considered preference behavior over the course of an entire search session, not only on a query-by-query basis. Sushmita *et al.* [43] conducted a task-oriented user study with two types of interfaces: a tabbed interface, where users could only access different verticals using tabs, and an aggregated interface, where the top results from every vertical were blended into the core web results. Participants were given a search task and asked to compile as much relevant content as possible. Two important trends were observed. First, the aggregated interface was associated with more clicks. Second, the amount of cross-vertical content compiled by participants was greater for the aggregated interface. Taken together, these two trends suggest that user engagement with vertical content is greater when the verticals are showcased in the main results.

The aggregated interface used in Sushmita *et al.* [43] was static—verticals were blended in fixed positions. In a later study, Sushmita *et al.* [44] investigated search behavior with a *dynamic* aggregated search interface. This study found two main results. First, users click more on verticals that are relevant to the task *and* verticals that are shown higher in ranking. In other words, aggregated search is not immune to positional bias. Users click more on verticals ranked higher, either because they scan results from top-to-bottom or because they trust that the top results are more relevant. Second, users click more on verticals that are more visually salient (in the case of this study, the *video* vertical). Thus, positional bias is not the only bias that affects clicks on vertical results.

Thus far, we have focused on user studies that investigate the question "What do users want?". Users want a system that combines results from different sources in the main results and a system that makes the relevant verticals more salient. Next, we focus on user studies that investigate factors that might affect user preference and search behavior.

As previously mentioned, an important advantage of user study evaluation is the ability to manipulate properties of the user's higher-level goal. Prior work shows that task complexity influences search behavior [20].[2] Among the differences in search behavior, is the demand for *diverse* content. That is, during more complex tasks, users exhibit a greater demand for content that

[2]Here, task complexity refers to *cognitive* complexity, which relates to the amount of learning required for the user to complete the task.

is more diverse (content from different sources or different types of media). Motivated by this finding, Arguello *et al.* [7] investigated the effect of task complexity on users' demand for vertical content, operationalized using clicks on vertical results. The study looked at the interaction between two experimental variables: *task complexity* and *vertical aggregation*. Participants were given search tasks of varying degrees of complexity and used two different interfaces: a tabbed interface in which vertical content was only *indirectly* accessible and an aggregated interface in which vertical results were also blended into the web results. Results showed a greater number of vertical clicks for more complex tasks, but only for the aggregated interface. However, the effect was only *marginally* significant. This result suggests that properties of the higher-level task may influence a user's demand for vertical results. Thus, session-level evidence, which can provide better hints about the higher-level task, may need to be considered in order to improve aggregated search.

During vertical selection, a false-negative mistake means that a non-relevant vertical was presented alongside the core web results. All evaluation methods for vertical selection and end-to-end aggregated search assume that all false-negative mistakes are *equally* bad. In other words, all instances where the system presents a non-relevant vertical equally hurt the user's search experience. However, consider a user that issues the ambiguous query "jaguar" because they are looking for a list of a places in the world where jaguar can found in the wild. In this particular scenario, displaying the *images* vertical can be viewed as a false-negative mistake. The information need is more likely to be satisfied with web results instead of image results. However, is the user experience equally affected if the images are all pictures of "jaguar" the automobile vs. pictures of "jaguar" the animal? Arguello and Capra [2] studied the effect of the query-senses represented in the blended image results on user interaction with the web results. They found that given an ambiguous query (e.g., "jaguar"), user interaction with the web results, operationalized using web clicks, is greater when the query-senses represented in the image results (e.g., "jaguar" the animal, or "jaguar" the car) are consistent with the intended query-sense (e.g., "jaguar" the animal). This result suggests that in certain situations, a "spill over" effect can occur. In other words, depending on the vertical results, certain false-negative vertical predictions may have a stronger negative effect on the user's perception of *other* results presented on the SERP.

Such interaction effects between different components of a SERP, which include vertical results, but also extend to components like query suggestions and advertisements, motivate work on *whole-page* evaluation. The idea behind whole-page evaluation is the relevance of a component may depend on *other* components presented on the SERP. For example, a relevant web result presented at rank five may be less relevant if it contains information that is redundant with a web result presented at rank one. Likewise, the quality of the SERP as a whole may be inferior if different components in the SERP (e.g, the web results, the vertical results, and/or the query-suggestions) focus

on different query-senses. Bailey *et al.* [8] proposed a whole-page evaluation methodology referred to as Student Assignment Satisfaction Index (SASI). The evaluation methodology focuses on eliciting judgments from assessors on parts of the SERP within the context of the whole SERP. Bailey *et al.* [8] show that the SASI-style judgments on the whole page can be done surprisingly fast. A whole SERP can be evaluated in the time it takes an assessor to make two document-level relevance judgements in a test collection evaluation method. However, the main disadvantage of this evaluation method is that the judgments are purely retrospective. In other words, the judgements are based on the system's output and are therefore not reusable for future evaluations.

1.5 Special Topics

1.5.1 Dealing with New Verticals

Just as pages are constantly being added to and removed from the web, verticals themselves appear as new subcollections are curated. Unfortunately, a new vertical requires training a new vertical selection model and requires new training data. As mentioned earlier, training data can be expensive to gather so it would be attractive for a system to be able to exploit training data from existing verticals when training a new vertical selection model.

Let $y_v(q)$ denote the true relevance label of vertical v with respect to query q.[3] In the general vertical selection setting, the goal is to learn a function f that approximates y. In this section, we focus on the following scenario. Assume we have a set, $\mathcal{S}$, of source verticals each with labeled queries. Then, suppose we are given a new (target) vertical t with no labeled data. The objective is to learn a function f that approximates y_t using *only* source-vertical training data. The quality of an approximation will be measured by some metric that compares the predicted and true query labels. We use notation,

$$\mu(f, y_t, \mathcal{Q})$$

to refer to the evaluation of function f on query set $\mathcal{Q}$. This metric could be any of those discussed in Section 1.4.

A *portable* vertical selection model is defined as one that can make vertical relevance predictions with respect to any arbitrary vertical. In other words, a portable model is not specific to a particular vertical, but rather

[3] This true label can be derived by thresholding of the training data such as $\frac{\left|\mathcal{D}^{+}_{(q,v)}\right|}{\left|\mathcal{D}_{(q,v)}\right|} > \lambda$.

agnostic of the candidate vertical being questioned for relevance. For this reason, throughout this section, we adopt the shared feature representation ϕ^{shared}(Equation 1.24).

Let us examine the distinction between a portable and non-portable vertical selection model with an example. Consider a single-evidence model that predicts a vertical relevant based on the number of times the query was issued previously to the vertical by users. This type of evidence is likely to be positively correlated with the relevance of the vertical in question. In fact, it is likely to be positively correlated with vertical relevance irrespective of the particular candidate vertical. On the other hand, consider a model that predicts a vertical relevant if the query is classified as related to the *travel* domain. This model may be effective at predicting the relevance of a vertical that serves travel-related content. However, it is probably not effective on a vertical that focuses on a different domain. This model is less portable.

Most existing single-evidence resource selection models can be considered portable [41, 40, 39, 36, 46]. For example, ReDDE [40] prioritizes resources for selection based on the estimated number of relevant documents in the collection. This expectation is a function of the number of documents sampled from the collection that are predicted relevant and the estimated size of the original collection. The greater the expectation the greater the relevance irrespective of the particular resource.

1.5.1.1 Basic Model

The objective of a portable vertical selection model, $f_\star$, is to maximize the average performance across source verticals. Our assumption is that if $f_\star$ performs consistently well across $\mathcal{S}$, then $f_\star$ will perform well on a new (target) vertical t. In general, the portability of a model is defined by a metric that quantifies performance for a vertical $s \in \mathcal{S}$ and a function that aggregates performance across verticals in $\mathcal{S}$.

For example, the portability, π, which uses the arithmetic mean of the metric is defined by,

$$\pi^{\text{avg}}(f_\star, y_{\mathcal{S}}, \mathcal{Q}_{\mathcal{S}}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mu(f_\star, y_s, \mathcal{Q}_s) \tag{1.33}$$

where $\mathcal{Q}_s$ is the set of training queries for source s and $\mathcal{Q}_{\mathcal{S}}$ is the set of those sets; similarly, y_s provides labels for vertical s and $y_{\mathcal{S}}$ is the set of these functions. We refer to the model which optimizes π^{avg} as the basic model.

1.5.1.2 Vertical Balancing

In the basic model's training set, positive instances correspond to relevant query-vertical pairs from *all* source verticals. For this reason, we expect the basic model to focus on evidence that is consistently predictive of relevance across source verticals, and hence predictive of the target vertical. In other

words, vertical-specific evidence that is conflicting with respect to the positive class should be ignored. The challenge, however, is that the positive instances in the basic model's training pool may be skewed towards the more popular source verticals. This is problematic if these verticals are reliably predicted relevant using vertical-specific evidence, not likely to be predictive of the new target vertical. To compensate for this, we consider a *weighted* average of metrics across verticals. Specifically,

$$\pi^{\text{wavg}}(f_\star, y_{\mathcal{S}}, \mathcal{Q}_{\mathcal{S}}) = \frac{1}{\mathcal{Z}} \sum_{s \in \mathcal{S}} w_s \mu(f_\star, y_s, \mathcal{Q}_s) \tag{1.34}$$

where $\mathcal{Z} = \sum_{s \in \mathcal{S}} w_s$. We use the simple heuristic of weighting a vertical with the inverse of its prior,

$$w_s = \frac{1}{p_s}$$

where p_s is the prior probability of observing a query with relevant vertical s. This value is approximated with the training data,

$$p_s \approx \frac{\sum_{q \in \mathcal{Q}_s} y_s(q)}{|\mathcal{Q}_s|}$$

The goal is to make training instances from minority verticals more influential and those from majority verticals less.

1.5.1.3 Feature Weighting

An alternative to optimizing for a portable model is to find portable features and to train a model using only those. A portable feature is defined as a feature which is highly correlated with relevance across all verticals. Recall that, across verticals, all features are identically indexed. Let ϕ^i be a predictor based only on the value of feature i. In previous work, the effectiveness of features across verticals was shown to be very dependent on the vertical being considered. In order to address the expected instability of feature predictiveness across verticals, we adopt a harmonic average for our aggregation method.

$$\pi^{\text{havg}}(\phi^i, y_{\mathcal{S}}, \mathcal{Q}_{\mathcal{S}}) = \frac{|\mathcal{S}|}{\sum_{s \in \mathcal{S}} \frac{1}{\mu(\phi^i_s, y_s, \mathcal{Q}_s)}} \tag{1.35}$$

Additionally, features, on their own, are not scaled to the label range, making the use of logistic loss difficult. Instead of constructing a mapping from a feature value to the appropriate range, we adopt a rank-based metric. In other words, for each feature, we rank queries by feature value and compute the harmonic mean average precision across verticals. Having computed the portability of each feature, we build a portable model by restricting our training to the most portable features.

1.5.1.4 Adaptability

Above, we focus on ways of improving the portability of a model by influencing the model to ignore evidence that is vertical-specific. The argument is that a model that focuses heavily on vertical-specific evidence will not generalize well to a new target vertical.

Given access to target-vertical training data, previous work reveals two meaningful trends [5]. First, given a wide-range of input features, most features contribute significantly to performance. In Arguello *et al.* [5], no small subset of features was solely responsible for effective vertical prediction. Second, the features that contributed the most to performance, which characterize the domain of the query, seem to be vertical-specific (assuming that verticals focus on different domains). Based on these observations, while ignoring vertical-specific evidence seems necessary to improve a model's portability, a model customized to a particular vertical is likely to benefit from it.

In the context of adaptation for web search, Chen *et al.* [11] propose several ways to adapt an already-tuned model given data in a new domain. Their approach, Tree-based Domain Adaptation (TRADA), essentially consists of continuing the training process on labeled data from the target domain. Arguello *et al.* apply this technique to adapt predictions from a portable vertical selection model to a new vertical [6].

1.5.2 Explore/Exploit

The gathering training data in a production environment is limited by production constraints. We cannot show every vertical for every query and expect to gather training data, much less retain users. At the same time, if a production system only gathers data from existing presentation decisions, judgments on suppressed verticals will not be gathered. This is precisely the problem of balancing exploration (i.e. gathering training data with good coverage) and exploitation (i.e. providing users with a satisfying experience). That said, we may be able to gather a small amount of feedback without devastating system performance. Specifically, we would like to present a vertical display for a query even though it is not predicted to be the display with the highest probability. As a result, our production system combines exploration (i.e. the random gathering of training data) with exploitation (i.e. the application of the trained model). Section 1.3 covers methods for exploitation. In this section, we will discussion two methods for exploration.

Naïve exploration suggests randomly altering vertical selection decisions as queries are submitted to the aggregated search system. Specifically, we can define a system sampling rate, ϵ, which determines if the system decision will be perturbed. This approach is referred to as the ϵ-greedy approach [45].

The ϵ-greedy approach while providing a control on the amount of sampling, does not incorporate any information from a query's feedback history. We may want to explore only when we have presented few displays. That is, we

might make ϵ a function of $\mathcal{D}_{(q,v)}$. To achieve this, we can exploit the fact that Equation 1.17 defines a distribution from which we can sample $p_{(q,v)}$. Using a random sample instead of the posterior mean results in a more data-driven policy than ϵ-greedy exploration. If we have seen few or no samples, the variance of the posterior will be high, resulting in samples unlike the posterior mean. As we accumulate samples, this variance falls, ensuring that samples will converge to the posterior mean. This process is similar to approaches used in reinforcement learning [42].

1.6 Conclusion

Although aggregated search is relatively new as a research topic, its foundation in distributed information retrieval has allowed rapid development of sophisticated models. In this chapter, we have outlined the fundamentals of developing an aggregated search system. These include features based on the distributed information retrieval literature as well as newer methods for training vertical selection and presentation systems.

Aggregated search will continue to develop as the online information landscape develops. New verticals will almost certainly necessitate revisiting feature development. At the same time, training algorithms will increasingly exploit more sophisticated user data to refine presentation decisions. Finally, as interfaces move away from traditional ranked lists, vertical presentation and interleaving frameworks will have to be modified.

References

[1] Deepak Agarwal, Evgeniy Gabrilovich, Robert Hall, Vanja Josifovski, and Rajiv Khanna. Translating relevance scores to probabilities for contextual advertising. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1899–1902, New York, NY, USA, 2009. ACM.

[2] Jaime Arguello and Robert Capra. The effect of aggregated search coherence on search behavior. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1293–1302, New York, NY, USA, 2012. ACM.

[3] Jaime Arguello, Fernando Diaz, and Jamie Callan. Learning to aggregate vertical results into web search results. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 201–210, New York, NY, USA, 2011. ACM.

[4] Jaime Arguello, Fernando Diaz, Jamie Callan, and Ben Carterette. A methodology for evaluating aggregated search results. In *Proceedings of the 33rd European conference on Advances in information retrieval*, ECIR'11, pages 141–152, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Jaime Arguello, Fernando Diaz, Jamie Callan, and Jean-François Crespo. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322, 2009.

[6] Jaime Arguello, Fernando Diaz, and Jean-François Paiement. Vertical selection in the presence of unlabeled verticals. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 691–698, New York, NY, USA, 2010. ACM.

[7] Jaime Arguello, Wan-Ching Wu, Diane Kelly, and Ashlee Edwards. Task complexity, vertical display and user interaction in aggregated search. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 435–444, New York, NY, USA, 2012. ACM.

[8] Peter Bailey, Nick Craswell, Ryen W. White, Liwei Chen, Ashwin Satyanarayana, and S.M.M. Tahaghoghi. Evaluating whole-page relevance.

In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 767–768, New York, NY, USA, 2010. ACM.

[9] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '95, pages 21–28, New York, NY, USA, 1995. ACM.

[10] Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Trans. Inf. Syst.*, 19(2):97–130, 2001.

[11] Keke Chen, Rongqing Lu, C. K. Wong, Gordon Sun, Larry Heck, and Belle Tseng. Trada: tree based ranking function adaptation. In *CIKM 2008*, pages 1143–1152. ACM, 2008.

[12] Cyril W. Cleverdon. The aslib cranfield research project on the comparative efficiency of indexing systems. *Aslib Proceedings*, 12(12):421–431, 1960.

[13] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 299–306, New York, NY, USA, 2002.

[14] Fernando Diaz. Regularizing ad hoc retrieval scores. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 672–679, New York, NY, USA, 2005. ACM Press.

[15] Fernando Diaz. Integration of news content into web results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 182–191. ACM, 2009.

[16] Fernando Diaz and Jaime Arguello. Adaptation of offline vertical selection predictions in the presence of user feedback. In *SIGIR 2009*, 2009.

[17] J.L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

[18] Michael D. Gordon and Peter J. Lenk. When is the probability ranking principle suboptimal? *Journal of the American Society for Information Science*, 43(1):1–14, January 1992.

[19] Jeff Huang, Ryen White, and Georg Buscher. User see, user point: gaze and cursor alignment in web search. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 1341–1350, New York, NY, USA, 2012. ACM.

[20] Bernard J. Jansen, Danielle Booth, and Brian Smith. Using the taxonomy of cognitive learning to model online searching. *Information Processing and Management*, 45(6):643–663, 2009.

[21] In-Ho Kang and GilChang Kim. Query type classification for web document retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 64–71, New York, NY, USA, 2003. ACM.

[22] Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 91–101, New York, NY, USA, 2002. ACM.

[23] Arnd Christian König, Michael Gamon, and Qiang Wu. Click-through prediction for news queries. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 347–354, New York, NY, USA, 2009. ACM.

[24] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 571–580, New York, NY, USA, 2010. ACM.

[25] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 297–306, New York, NY, USA, 2011. ACM.

[26] Xiao Li, Ye-Yi Wang, and Alex Acero. Learning query intent from regularized click graphs. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 339–346, New York, NY, USA, 2008. ACM.

[27] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[28] Donald Metzler, Susan T. Dumais, and Christopher Meek. Similarity measures for short segments of text. In *ECIR*, pages 16–27, 2007.

[29] Ashok Kumar Ponnuswami, Kumaresh Pattabiraman, Desmond Brand, and Tapas Kanungo. Model characterization curves for federated search using click-logs: predicting user engagement metrics for the span of feasible operating points. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 67–76, New York, NY, USA, 2011. ACM.

[30] Ashok Kumar Ponnuswami, Kumaresh Pattabiraman, Qiang Wu, Ran Gilad-Bachrach, and Tapas Kanungo. On composition of a federated web search result page: using online users to provide pairwise preference for

heterogeneous verticals. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 715–724, New York, NY, USA, 2011. ACM.

[31] Ashok Kumar Ponnuswami, Kumaresh Pattabiraman, Qiang Wu, Ran Gilad-Bachrach, and Tapas Kanungo. On composition of a federated web search result page: using online users to provide pairwise preference for heterogeneous verticals. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 715–724, New York, NY, USA, 2011. ACM.

[32] Stephen Robertson. The probability ranking principle. *Journal of Documentation*, 1977.

[33] Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 377–386, New York, NY, USA, 2006. ACM.

[34] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[35] Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011.

[36] Jangwon Seo and Bruce W. Croft. Blog site search using resource selection. In *CIKM 2008*, pages 1053–1062. ACM, 2008.

[37] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Q2C@UST: Our winning solution to query classification in KDDCUP 2005. *ACM SIGKDD Exploration Newsletter*, 7:100–110, December 2005.

[38] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 131–138, New York, NY, USA, 2006. ACM.

[39] Milad Shokouhi. Central rank based collection selection in uncooperative distributed information retrieval. In *ECIR 2007*, pages 160–172, 2007.

[40] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *SIGIR 2003*, pages 298–305. ACM, 2003.

[41] Luo Si, Rong Jin, Jamie Callan, and Paul Ogilvie. A language modeling framework for resource selection and results merging. In *CIKM 2002*, pages 391–397. ACM, 2002.

[42] Malcolm J. A. Strens. A bayesian framework for reinforcement learning. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 943–950, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[43] Shanu Sushmita, Hideo Joho, and Mounia Lalmas. A task-based evaluation of an aggregated search interface. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, SPIRE '09, pages 322–333, Berlin, Heidelberg, 2009. Springer-Verlag.

[44] Shanu Sushmita, Hideo Joho, Mounia Lalmas, and Robert Villa. Factors affecting click-through behavior in aggregated search interfaces. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 519–528, New York, NY, USA, 2010. ACM.

[45] Richard Sutton and Andrew Barto. *Reinforcement Learning*. MIT Press, 1998.

[46] Paul Thomas and Milad Shokouhi. Sushi: Scoring scaled samples for server selection. In *SIGIR 2009*. ACM, 2009.

[47] Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. Clustering user queries of a search engine. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 162–168, New York, NY, USA, 2001. ACM.

[48] Ke Zhou, Ronan Cummins, Mounia Lalmas, and Joemon M. Jose. Evaluating aggregated search pages. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 115–124, New York, NY, USA, 2012. ACM.

[49] Dongqing Zhu and Ben Carterette. An analysis of assessor behavior in crowdsourced preference judgements. In *SIGIR Workshop on Crowdsourcing for Search Evaluation*, pages 21–26, New York, NY, USA, 2010. ACM.