Precomputing Search Features for Fast and Accurate Query Classification

Venkatesh Ganti
Microsoft Research
One Microsoft Way
Redmond, WA 98052
vganti@microsoft.com

Arnd Christian König
Microsoft Research
One Microsoft Way
Redmond, WA 98052
chrisko@microsoft.com

Xiao Li
Microsoft Research
One Microsoft Way
Redmond, WA 98052
xiaol@microsoft.com

ABSTRACT

Query intent classification is crucial for web search and advertising. It is known to be challenging because web queries contain less than three words on average, and so provide little signal to base classification decisions on. At the same time, the vocabulary used in search queries is vast: thus, classifiers based on word-occurrence have to deal with a very sparse feature space, and often require large amounts of training data. Prior efforts to address the issue of feature sparseness augmented the feature space using features computed from the results obtained by issuing the query to be classified against a web search engine. However, these approaches induce high latency, making them unacceptable in practice.

In this paper, we propose a new class of features that realizes the benefit of search-based features without high latency. These leverage co-occurrence between the query keywords and tags applied to documents in search results, resulting in a significant boost to web query classification accuracy. By pre-computing the tag incidence for a suitably chosen set of keyword-combinations, we are able to generate the features online with low latency and memory requirements. We evaluate the accuracy of our approach using a large corpus of real web queries in the context of commercial search.

Categories and Subject Descriptors

H.3.1 [**Information Search and Retrieval**]: Indexing Methods; H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Measurement, Performance, Experimentation

1. INTRODUCTION

Query intent classification is crucial for a number of applications, including computational advertising [7], vertical searches [19] or query reformulations [16]. One of the main challenges for accurate query-intent classification is that web queries contain – on average – less than three words of text, from a very large vocabulary of terms. Text classification techniques, that use the occurrence of specific words or word-combinations in a query as features thus have to contend with a very large and sparse feature space. The issue of feature sparseness is problematic as classifiers relying on these features may require very large amounts of training data to produce accurate classification. To illustrate this, consider the large number of labeled web queries required to cover a significant fraction of the vocabulary used in web search today. Given that manual labeling of queries is time-consuming and potentially expensive, this constraint can be crucial.

One approach to overcome feature sparseness has been to augment the feature space with additional data sources. It has been shown that approaches which augment word-based features by first issuing the query against a (web) search engine and subsequently extracting additional features from the search results can increase the accuracy of query classification significantly (e.g., [5, 7, 23, 6]). We refer to such features as *post-retrieval* features. However, these gains come at a significant cost: the retrieval of the result web pages and (when necessary) their subsequent processing can be very expensive and may not satisfy the latency-requirements of realistic search architectures. It is known that latency is critical to user satisfaction in web search (e.g., [8, 1]).

Ideally, we would like to realize the accuracy gains possible through post-retrieval features, yet with only small overhead/latency (and memory requirements) for the online query classification component. We address this challenge in this paper.

1.1 Our Approach

We consider a new type of post-retrieval features, which are based on the incidence of *tags* associated with the pages retrieved in response to a search query. Generally, these tags express properties of the pages, such as its topic or which types of entities it contains. The set of tags associated with search results is valuable for query classification – for example, knowing the distribution of topics retrieved in response to a query may allow us to determine the query's intent. As we will show, using various combinations of tags and corpora allows us to improve accuracy in real-life query classification tasks dramatically. Moreover, the number of tags is usually much smaller ($\leq$ 5K) than the number of all words in search queries (with 10^6–10^7 relevant terms), reducing the size and sparsity of the feature space. This allows us to formulate features that generalize better across queries and reduces the amount of training data required.

We use the incidence of each tag within the search results (i.e., the number of tag occurrences as a fraction of the result size) to compute features. This means that we need to compute the number of search results for a query q and – for each distinct tag t – the number of documents in the result tagged with t; we will refer to the ratio between tagged pages and all result pages as the *tag ratio* for a tag t and a query q. Using features based on tag ratios allows us to overcome the challenge of feature sparsity as the number of resulting features depends on the much smaller number of tags and not on the extremely large vocabulary of words used in search.

We address the challenge of keeping the feature computation la-

WSDM'10, February 4–6, 2010, New York City, New York, USA.

tency low as follows. Instead of computing the tag ratios at query time, we pre-compute them for suitably chosen sets of query keywords; this set is sufficiently small so that we can then store and index the tag ratios in main memory.

When a new search query q is posed, we look up the tag ratios for keyword-combinations *related* to the query and use them to derive the final features. As we will show, this approach results in significant improvements in classification accuracy even when we have not pre-computed the tag ratios for the specific query to be classified. We will define the notion of relatedness we use in Section 3.3. Our overall approach is depicted in Figure 1.

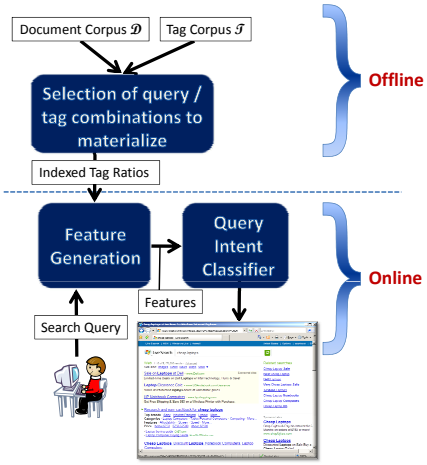

Figure 1: Our Architecture

1.2 Text Corpora and Tags

We now illustrate our approach using different text corpora.

Product Category Tags: An important query classification task in search is identifying queries with *product intent*. *Product-intent* means that the query refers to a specific product or a class of products and is intended to research, purchase or review these. A property that we observed for this scenario is that (the categories of) named entities, particularly commercial products, found in the pages returned in response to a query are very indicative of this query's intent. For example, queries that have product intent for a given category (e.g., *consumer electronics*) will commonly surface pages containing related product entities (e.g., *DVDs, music, TVs*). We can think of the occurrence of one or more entities from a specific category within a page as a page tag. Now each tag in $\mathcal{T}$ corresponds to a different product category; a page is tagged if it contains an entity in the category. We then use the occurrence-frequencies of such tags within the result documents as input features to a query classifier.

Wikipedia Page Category Tags: Consider the task of identifying queries with *entertainment* intent for which a search engine may want to display picture galleries or videos in addition to the search result. One approach here is to use a specific corpus (e.g., *Wikipedia*) for which a rich set of page categories are available. Then, the page categories (e.g., Wikipedia-categories such as *American Actor*, *Film by Genre: Romance*, *Dance*, etc.) are used as the tags and their relative frequency in search results as features.

The use of Wikipedia tags has the advantage that large classes of queries (e.g., names of famous actors) that have entertainment intent are reduced to a small number of tags. This in turns allows our technique to generalize much better across queries than classifiers based on the query text alone.

Ad Domain Tags: Finally, consider the task of identifying queries with retail intent, which is defined as product intent classification across the range of all retail products. Query log analysis shows that approximately 5%-7% of the distinct web search queries contain retail intent, although this number varies with the date and search-engine [19]. Obviously, we could use the tags introduced in the context of product intent classification here as well (using a larger range of product categories). A complementary approach is to use the corpus of advertising bids used in sponsored search. In sponsored search, each advertiser specifies which queries an ad is potentially shown for using a set of bid-phrases. These bid-phrases are matched against an incoming query to determine which ads to display.

Now, each advertiser is interested in specifying bid-phrases to capture those queries that have commercial intent for the products/services she is offering. Thus, the set of advertisers that have submitted a bid-phrase matching a specific query is a good signal to capture its retail intent. We can leverage this by treating the corpus of bid phrases as a set of documents, where each document is "tagged" with the advertiser who has submitted the bid.

Local vs. Web Corpora: The corpora we use to compute tag ratios could either be web-corpora, or smaller, manually curated and tagged text corpora with 10^6–10^7 documents. The advantage of using smaller corpora is threefold: first, if the set of tags is manually created and maintained (such as in Wikipedia), this typically results in more accurate tags than those generated automatically. Second, we have more control on the set of documents and are able to avoid issues such as spam. Finally, the pre-computation of the tag-ratios itself becomes much less expensive.

1.3 Contributions

Overall, our paper makes the following salient contributions:
(1) *Accuracy:* We define a novel class of post-retrieval features based on the incidences of tags in the documents containing search terms. We show that such features can be derived from a variety of corpora and result in a significant boost in accuracy for various query classification tasks.
(2) *Efficiency:* Moreover, we show that these features are amenable to off-line pre-computation; this allows us avoid latency problems by pre-computing the tag ratios required for feature computation and indexing them in main memory. Because the main memory space is limited, we will define a subset of all tag ratios that results in sufficiently small main memory requirements without significantly affecting classification accuracy.
(3) *Scalability:* We develop efficient algorithms that pre-compute the relevant keyword-combinations and the associated tag ratios from large text corpora.
(4) *Evaluation:* Finally, we demonstrate the applicability of our approach to various query classification problems by showing the accuracy improvements for three different real query classification tasks. We also evaluate the scalability of the preprocessing steps and the ability of our features to generalize.

2. RELATED WORK

The problem of feature sparseness described in Section 1 has received significant attention recently. One related approach has been the enrichment of query features using search engine results. Here, the KDD 2005 Cup inspired the use of the World Wide Web and in particular search results for feature enrichment. The winning solution [23] leveraged search engine results, including page titles, result snippets and the page text to enrich the feature space. However, the latency of this approach was not discussed in [23]; it may

not be efficient, as it requires the classifier to wait for the search results to be computed and the result documents to be processed.

Other approaches that leverage similar techniques have been described in the context of web search advertising [7, 6]. In both these papers, web queries are first classified using the top K search result documents returned to generate additional features. The system described in [6] is based on an experimental system that crawls and analyzes web search results at query time; the changes required to make this approach efficient for a real-life system are only outlined briefly, without any evaluation of the resulting latency. In [7], only small numbers of search results are taken into account (the paper reports optimal precision for $K = 40$ results); again, there is no evaluation of the resulting latency. In contrast, our solution extends to much larger result sizes, allowing for more robust estimates of the incidence of tags in the result documents.

A different approach to address the issue of feature (and training data) sparsity has been the use of semi-supervised techniques that leverage both labeled and unlabeled training data (e.g., [5]) or approaches that – based on an initial seed set of labeled queries – enrich the training corpus by leveraging the web click graph [19]. These techniques are orthogonal to our approach, which only expands the feature set, and can be combined with it. The same holds for techniques that leverage NLP techniques to automatically extract training data from web sources [14]. As we will show in our experiments, our approach results in significant increases in classification accuracy even in cases where very large amounts of training data ($\approx 10^6$ labeled examples) are available.

A related approach is also used in [11]; here the authors study the use of post-retrieval features for the prediction of query difficulty and expansion risk. Here, the authors also point out the costs associated with post-retrieval processing and propose pre-computing a low-dimensional summary of the documents to address this.

Vertical Selection, which can be thought of as a different, specific query classification task, is studied in [2]. Here, the authors use evidence from external resources, including corpora representative of the vertical content. The main focus of this paper is the integration and modeling of different types of features derived from the corpora; the performance of feature generation is not evaluated.

3. FEATURES

Notation: Throughout the paper we will use the following notation: Let $\mathcal{D} = \{d_1, \ldots, d_j\}$ denote the document corpus. As illustrated in the earlier examples, this corpus may correspond to a set of Wikipedia documents, advertisement bids or a web crawl. Given a corpus $\mathcal{D}$, we use $\mathcal{V}$ to denote the set of all distinct words in $\mathcal{D}$ and $2^{\mathcal{V}}$ to denote the power-set of $\mathcal{V}$. We denote the frequency of a keyword $v \in \mathcal{V}$ as $frq(v) := |\{d \in \mathcal{D} | v \in d\}|$. Similarly, we use the notation $frq(q)$ to denote the number of documents in $\mathcal{D}$ that contain a set of keywords $q \subseteq \mathcal{V}$. We define the set of all tags that may be associated with documents in $\mathcal{D}$ as $\mathcal{T} = \{t_1, \ldots t_p\}$. If a document d is tagged with a tag t, we also use the shorthand $t \in d$; we denote the set of all documents tagged with t as $\mathcal{D}^t \subseteq \mathcal{D}$.

Query Semantics: To define the result of a keyword-search for a query q, we will define $result(q) \subseteq \mathcal{D}$ as the set of all documents retrieved as a response to a query q. In general, we can incorporate varying search semantics, depending on the query and the corpus. In this paper, we will primarily use containment-semantics, i.e., we model a query as a set of keywords $q = \{w_1, \ldots, w_l\}$ and $result_{\mathcal{D}}(q) = \{d \in \mathcal{D} \mid d \text{ contains all } w_i \in q\}$ – these semantics have the advantage of decoupling our approach from any specific search engine or algorithm. Note that under these semantics, q corresponds to an unordered set of words. We denote the number of words in a query q as $|q|$.

Definition 1. Tag Ratios: given a set of documents $D \subseteq \mathcal{D}$ and a tag $t \in \mathcal{T}$, we define the *tag-ratio* of D with respect to t as $ratio_D(t) = |\{d \in D | d \text{ is tagged with } t\}|/|D|$. If D corresponds to the result of a query q, we also use the notation $ratio(q, t) := ratio_{result(q)}(t)$. In the special case that a query result is empty, we define $ratio_\emptyset(t) = 0$.

3.1 Using Tag Ratios in Classification

A straight-forward way to use tag ratios in query classification would be to associate each query q with the feature vector formed by q's tag ratios: $F = [ratio(q, t_1), \ldots, ratio(q, t_p)]$. However, there are two important reasons to extend this approach to also consider tag ratios associated with suitable queries $q' \neq q$ when classifying a query q.

First, it is often possible to improve classification accuracy for a query q further when not only considering the tag ratios for q, but also those of queries $q' \subset q$. The main reason for this is that longer queries may result in small (or even empty) result sets, thereby making it hard to assess the correlation between the individual tags and the words in q. By also considering the ratios for subsets q' we are often able to obtain additional estimates of tag incidence, which – as we will show experimentally – result in improved classification accuracy. For example, consider the query $q = \{$*Canon Camera SD2*$\}$ This query is likely surface an empty (or very small) result set, as '*SD2*' is not a valid Canon camera model (as opposed to SD5 or SD7). However, if we consider the tag ratios surfaced by the query $q' = \{$*Canon Camera*$\}$ we are still likely able to infer that q has commercial intent.

Second, as we have stated before, our approach relies on pre-computing the tag ratios for queries and indexing these in memory to derive features at run-time. For the indexing and retrieval of tag ratios we use efficient and compact main-memory data structures designed for retrieval of advertisements in sponsored search [18].

It is neither possible to pre-compute these tag ratios for all possible queries nor to store them, due to constraints on pre-processing time and storage space. We therefore compute the ratios for an appropriately chosen subset of all keyword combinations and tags. This means that we will have to in some cases generate features for queries q where the values of $ratio(q, t)$ are not stored. We now describe a more general scheme that derives features for a given query q from the pre-computed tag ratios for subsets of q.

3.2 "Backoff" ratios and their benefits

The use of the query results for queries containing subsets of the words in the query q to be classified is conceptually related to "*back-off models*" used in language modeling ([20], p. 219). In these models, the probability of a text n-gram is modeled "backing off" to models with smaller histories under certain conditions, i.e., the conditional probability of a n-gram $(w_1, \ldots, w_k)$ is not modeled as $Prob(w_1 | w_2 \ldots w_k)$ but rather using a smaller history $w_{2+i} \ldots w_k$. Our idea is similar in that we – instead of using the tag ratios for query q – "back off" to a combination of the tag ratios from queries $q' \subset q$. We refer to these tag ratios as *back-off ratios* from now on.

To illustrate the significance of using these ratios, we consider the following experiment. Note that this is just a motivating example; we later give a more comprehensive explanation of how we generate features from tag ratios. Consider the product intent classification task described in Section 1.1. We use a corpus $\mathcal{D}$ of 7.2 Million Wikipedia documents and define the set of tags $\mathcal{T}$ using a list of 759K products from the area of consumer electronics; each of these products has been categorized into one of 157 higher-level categories such as *CDs*, *cameras*, and *MP3-players*, etc. We use

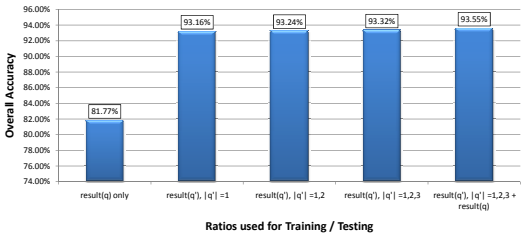

Figure 2: Backoff ratios improve classification accuracy significantly.

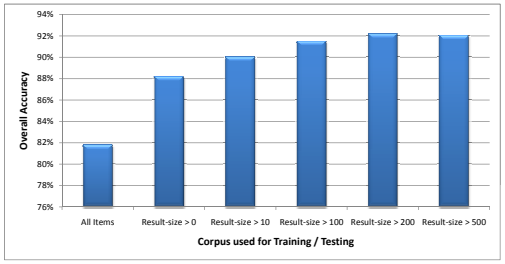

Figure 3: Accuracy for queries with different result-sizes.

this list of categories as $\mathcal{T}$ and tag each document with $t \in \mathcal{T}$ if it contains a product belonging to the corresponding category.

To evaluate the impact of different feature sets, we use 30K real-life search queries which have been manually labeled as either having product-intent or not for *consumer electronics* products. This data is separated into 20K training and 10K test examples, with the baseline probability of label "*no product intent*" being 67.2%.

We compare the classification accuracy we obtain by using the following feature sets. First, as a baseline feature set F_{base}, we use the tag ratios for all tags in $\mathcal{T}$ as the feature vector for a query q. Second, we define additional feature sets in which we use the tag ratios for all subsets q' of q of sizes $s = 1, 2, 3$; here, each feature set F_i contains the average tag ratios for all subqueries q' of q for $|q'| \leq i$, grouped by t and $|q'|$. In addition, we use the union of F_{base} and F_3 as the final feature set.

We have plotted the resulting classification accuracies in Figure 2. The baseline feature set results in a classification accuracy of 81.8%, which is significantly better than the baseline probability, but not on par with the accuracy of a classifier based on the query text itself (see Section 6.1). In contrast, all feature-sets that incorporate back-off ratios result in significantly better accuracy; the accuracy increases with the number of back-off features used. In fact, using the back-off ratios for single keywords only already results in significantly higher accuracy than the base feature set F_{base}.

The main reason for these improvements is that using back-off ratios allows us to more accurately assess the correlation between query keywords and tags for queries with small result sizes. In particular, a number of queries with product intent contain long combinations of rare terms (such as model numbers) which do not all co-occur in a single document within Wikipedia, in turn resulting in a completely empty feature-vector if we only use tag ratios for the original query. To illustrate these effects, we ran a similar experiment using only the F_{base} feature set, but included only queries in the test and training data for which the corresponding result-sizes surpassed a threshold Δ; here, we varied Δ between 0 and 500. The accuracy for these subsets of the test data is plotted in Figure 3.

As we can see, the accuracy we can achieve using F_{base} only for queries with large result sizes is more than 10% higher than the accuracy over all queries. In fact, for the feature set where we eliminated queries which have an empty result only, we already see a classification accuracy of 88%, as well as higher gains for queries with larger result sizes; however, the feature sets that incorporate backoff ratios still perform better.

3.3 Features based on Backoff Ratios

As illustrated above, the set of backoff ratios is – on average – a more robust representation of a query q's intent than the tag ratios for q itself. We can now define features that characterize the distribution of tag ratios in this set. One thing we have to account for is the fact that the number of (backoff) ratios for a given tag/query combination can vary depending on the number of keywords in the query; at the same time, we require a constant number of features in the classifier. To address this, we group the ratios associated with a query into groups and compute features by aggregating over the ratios in each group.

A simple example would be to – similarly to what the experiment in the previous section – simply compute an aggregate (e.g., the average) over all ratios and use these aggregates for the different $t \in \mathcal{T}$ as features. The one limitation of computing such aggregates over *all* backoff ratios is that it does not differentiate between the keyword combinations associated with the backoff ratios. However, a keyword combination q' that shares 3 words with the query q be classified is more likely to result in tag ratios characterizing the intent of q than a combination q'' that shares only 1 word. To leverage this, we first partition the backoff ratios into groups depending on the "similarity" between the keyword combination q' and the query to be classified q. There are several possible techniques to perform this partitioning: due to space-constraints, we only use a simple grouping scheme in this paper that partitions keyword combinations q' by the number of words they have in common with the query q to be classified. However, our approach is also applicable to much more sophisticated partitioning schemes, such as grouping the tag ratios based on different notions of query similarity (e.g., [21, 26]).

For our simple partitioning scheme, we define a grouping operator π that – for an input query q and a vocabulary $\mathcal{V}$ groups subsets q' of q into groups Q_s by the number of words they have in common with q:

$$\pi(q) = \{\underbrace{\{q' \in 2^{\mathcal{V}} | q' \subseteq q \text{ and } |q' \cap q| = s\}}_{Q_s} | s = 1, \ldots |q|\}.$$

In practice, we generally limit the length of the combinations q' we consider here to a small constant.

For each of the groups Q_s created in this manner, we define features that characterize the distribution of tag ratios in each group, such as their *average, sum, standard deviation, min, max* etc. For example, for the case of average we generate the features

$$F_{avg}(Q_s) := [\frac{\sum_{q' \in Q_s} ratio(q', t)}{|Q_s|} | t \in \mathcal{T}].$$

Moreover, we create one additional feature vector that contains the (average) result sizes of the queries in each Q_s:

$$Count_{avg}(Q_s) := [\frac{\sum_{q' \in Q_s} |result(q')|}{|Q_s|} | t \in \mathcal{T}].$$

These last set features allow the classifier to distinguish between queries that do not co-occur with a specific tag and ones that have empty result sets.

Our approach results in a feature set that (a) is much smaller than the word-occurrence features commonly used in text classification (as its dimensionality depends on $|\mathcal{T}|$ and not $|\mathcal{V}|$) and (b) has a constant number of features even for varying number of backoff ratios.

3.4 Classification Model

The classification model we use is based on *Multiple Additive Regression-Trees* (*MART*). MART is based on the *Stochastic Gradient Boosting* paradigm described in [13] which performs gradient descent optimization in the functional space. In our experiments, we used the log-likelihood as the loss function (optimization criterion), used the steepest-decent (gradient descent) as the optimization technique, and used binary decision trees as the fitting function - a "non-parametric" approach that applies numerical optimization in functional space. The details of the algorithm are described in [25].

One reason behind the use of this technique is that – unlike the sparse feature spaces common in text classification – the classification tasks resulting from the feature sets defined above typically have more training examples than distinct features; in addition to MART, we also tested different classification models (*SVM-models, logistic regression*), none of which produced comparable accuracy. We omit the details due to space considerations.

4. SELECTING TAG RATIOS

It is known that the size of vocabularies grows steadily with the size of the underlying text corpus [3], in turn implying vocabulary-sizes in excess of 10^7 (and often of 10^8) entries for the types of corpora (10^6-10^7 documents) we are considering. Since we index the pre-computed tag ratios in (a limited amount of) main memory, we are not able to store the tag ratios for all possible word-combinations and tags, even when we restrict ourselves to short queries. Therefore, the challenge is to select a subset of tag ratios that we can index in main memory, but at the same time gives as high classification accuracy as possible. We restrict the set of tag ratios we pre-compute and store for a given tag $t \in \mathcal{T}$ as follows:

Short queries: As illustrated in Figure 2, it is often possible to infer the correct category for a query q using the tag ratios containing small subsets of the words in q only. As a consequence, we limit the number of words in the queries q' for which we pre-compute tag ratios to a small constant w_{max}.

Informative Tag Ratios: Ideally, we want to preserve the tag ratios that are of high value for classifying queries; specifically, we want to retain keyword combinations whose distribution exhibits significant correlation (either positive or negative) to the tag. To characterize the importance of different tag ratios, we model the distribution of tags as a random process that assigns a tag to page with the probability $p = |\mathcal{D}^t|/|\mathcal{D}|$. Now, if the incidences of a tag are not correlated with the occurrence of a set of words q, we can expect $ratio(q,t) \approx p$. In contrast, if the words in q have significant positive/negative correlation, the corresponding ratio will be significantly larger/smaller than p.

Therefore, we retain tag ratios that differ significantly from p:

$$\boxed{ratio(q,t) \leq \theta_{\text{low}} \frac{|\mathcal{D}^t|}{|\mathcal{D}|}, \text{ or } ratio(q,t) \geq \theta_{\text{high}} \frac{|\mathcal{D}^t|}{|\mathcal{D}|}|} \quad \text{[Correlation Condition]}$$

for appropriately chosen $\theta_{\text{low}} < 1$, $\theta_{\text{high}} > 1$. In the following, we will refer to the required ratios of occurrences for a tag t as $\beta^t_{low} = \theta_{\text{low}} \frac{|\mathcal{D}^t|}{|\mathcal{D}|}$ and $\beta^t_{high} = \theta_{\text{high}} \frac{|\mathcal{D}^t|}{|\mathcal{D}|}$.

No small results: For the tag ratios themselves to be statistically meaningful, we need to also ensure that the underlying size of $result(q)$ is sufficiently large to allow us to robustly assess the correlation between the occurrence of keywords and tags:

$$\boxed{|result_{\mathcal{D}}(q)| \geq \alpha} \quad \text{[Support Condition]}$$

We refer to keyword combinations q that satisfy all three conditions specified above for a tag t as *indicative* for t, since they are indicative of correlation between the keywords and the tag.

Practical considerations: In practice, it often makes sense to set different values for the parameters α, β^t_{low}, and β^t_{high} for different sizes of the corresponding keyword combinations. In particular, given the results of the experiments in Section 3.2, it makes sense to materialize the ratios for all singe keywords in $\mathcal{V}$ and use the above pruning for combinations of multiple keywords only. While current servers can easily hold a vocabulary $\mathcal{V}$ of size $|\mathcal{V}| = 10^7$ words and all single-word ratios (for moderately-sized tag-corpora, e.g., $|\mathcal{T}| = 200$) in main memory, the number of multi-word combinations grows rapidly as the number of words increases (e.g., see [10]), in turn requiring more stringent pruning.

4.1 Properties of Indicative Combinations

In order to justify the pruning conditions, our approach has to satisfy two properties: (i) Using only indicative keyword combinations does not reduce the resulting classification accuracy significantly. We will experimentally evaluate this in Section 6. (ii) Materializing only indicative keyword combinations leads to a large reduction in the overall storage space required. In the following, we will give a formal justification of this property and illustrate it with examples from real-life data sets. In the experimental evaluation we will then quantify the reduction in space empirically.

Intuitively, we will show that the support and correlation conditions can be expected to prune away a large fraction of word combinations: because the distribution of words and word-combinations in natural language documents can be expected to follow a power-law, the support condition already removes the vast majority of keyword combinations from consideration. Now, if you consider the subset of the remaining word combinations and assume (most of) these were uniformly distributed in $\mathcal{D}$, then we expect the corresponding tag ratios to be concentrated around their expectation, in turn meaning that most of them will be pruned by the correlation condition. In other words, if we think of the keywords co-occurring with a tag t as partitioned into two groups – one group containing keywords that have a semantic relationship to t and hence exhibit correlation and the second group containing a large number of keywords distributed at random, independently of t, then it is unlikely that many keyword-combinations from the second set will satisfy both the support and correlation conditions. We will now formalize this argument.

Preliminaries: It is well-known that the frequency-distribution of single keywords in natural language text follows a power-law [3]; similarly, our experiments regarding the frequency-distribution for keyword-combinations indicate that this is true for the frequency-distribution of sets of keywords in a natural-language document corpus as well (see [10] (Figure 3) and [18] (Figure 2)). To describe this distribution, we will use the following notation. Let N be the total number of all pairs of words $w \in \mathcal{V}$ and documents $d \in \mathcal{D}$ for which $w \in d$, and $V = |\mathcal{V}|$ be the number of distinct words in

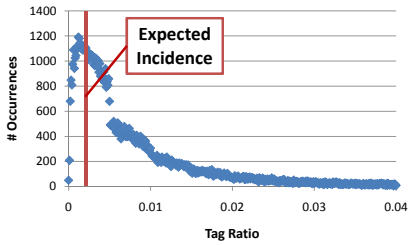

Figure 4: Distribution of tag ratios for tag '*English-language Films*'.

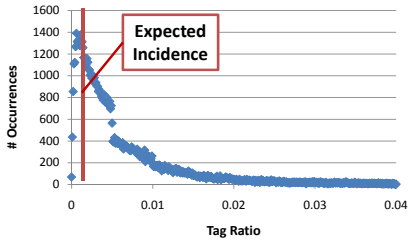

Figure 6: Distribution of tag ratios for tag '*American Film Actors*'.

$\mathcal{D}$. Due to the power-law, the frequency $frq(w)$ of a word of rank r can be modeled as

$$f(r) = \frac{\zeta}{r^a} N$$

where ζ is a normalizing constant smaller than 1 ensuring that $\sum_{r=1}^{V} f(r) = N$ and a is a fitting parameter modeling the skew of the distribution. For ease of exposition we set a equal to unity, resulting in the *standard harmonic* probability distribution over words.

Single indicative keywords: First, consider the case of all single-keyword queries q and a specific tag t. The *support condition* thus eliminates roughly $|\mathcal{V}| - \frac{\zeta \cdot N}{\alpha}$ (and thus the majority of) keywords from consideration.

For the remaining keywords, we can use the following argument: For a keyword $v \in \mathcal{V}$ of rank r_v, we know that it occurs in $f(r_v)$ documents in total. Now, consider the subset of keywords for which the occurrence of t is independent of the occurrence of the keyword (i.e., they have no semantic relationship).

For each of these $f(r_v)$ documents, the probability that it is tagged with t is $p = |\mathcal{D}^t|/|\mathcal{D}|$. In order for the keyword v to be indexed, either more than $\theta_{high} \cdot p \cdot f(r_v) = \beta^t_{high} \cdot f(r_v)$ or less than $\theta_{low} \cdot p \cdot f(r_v)$ documents among the $f(r_v)$ documents v occurs in need to be tagged with t. In the following, we will only consider the case of more than $\beta^t_{high} \cdot f(r_v)$ occurrences of t, the argument for the other case is analogous. Let the random variable X denote the number of documents tagged with t in which v occurs. For a keyword v which occurs independently of t, the probability that this is the case corresponds to the probability of flipping $f(r_v)$ biased coins that show head with probability p and obtaining more than $\beta^t_{high} \cdot f(r_v)$ heads with these flips. Hence, the probability $Pr(X \geq \beta^t_{high} \cdot f(r_v))$

$$= \quad 1 - \sum_{j=0}^{\lfloor \beta^t_{high} \cdot f(r_v) \rfloor} \binom{f(r_v)}{j} (p)^j \cdot (1-p)^{f(r_v)-j}.$$

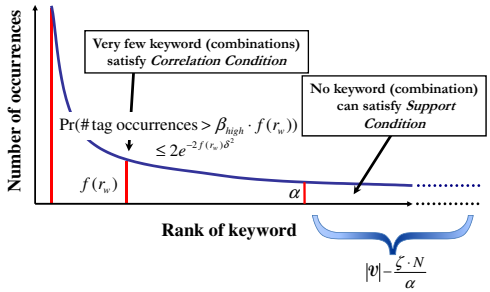

Figure 5: Tight "concentration bounds" make it unlikely for combinations of unrelated keywords to satisfy both conditions by chance.

Let δ denote this difference in ratio: $\delta = \beta^t_{high} - p$. It is known that the distribution of the random variable X as defined above is tightly concentrated around its expectations as the number of coin flips becomes larger. Using *Chernoff's bounds* we obtain

$$Pr\big(X - p \cdot f(r_v) \geq \delta \cdot f(r_v)\big) \leq 2e^{-2 \cdot f(r_v) \cdot \delta^2}.$$

Here, the number of documents appears in the (negative) exponent, illustrating the rapid decline in probability of a large deviation from the expectation as $f(r_v)$ increases. Now, due to the support-condition, all keywords with small frequencies (and correspondingly small numbers of coin flips) for which the concentration is more loose are pruned anyhow (see Figure 5). Given the power-law distribution of the keyword frequencies, this means that by choosing the parameter α to be sufficiently large, the tight concentration bounds make it very unlikely for keywords whose occurrence is independent of t to satisfy both constraints.

Keyword-combinations: Under the assumption that the frequency-distribution of multi-keyword combinations follows a power-law as well, the same argument can be made for keyword-combinations, when scaling up N appropriately and modeling the frequency distribution as a power-law over sets of keywords.

Examples: To illustrate this, consider the Wikipedia corpus introduced in Section 3.2. This corpus contains a total of 34M distinct keywords; setting $\alpha = 200$ (i.e., we consider only keywords that occur in at least 200 documents) eliminates all but 145K of them. We then compute a frequency histogram of the corresponding tag ratios, using a bucket-width of 10^{-4}; here, we plot the range of tag ratios on the x-axis and the number of ratios within this range on the y-axis. Figures 4 and 6 show this distribution for the tags *English-language Films* and *American Film Actors*; we can see that the distribution peaks near the expected rate of occurrence of $|\mathcal{D}^t|/|\mathcal{D}|$. At the same time, the distribution is less concentrated than we would expect for full independence between occurrences of the tags and the keywords. The distributions shown in the graphs indicate a positive correlation between a non-trivial fraction of the keywords that occur frequently in the same document as a tag t and occurrence of t itself. Still, as we can see a drastic drop-off in the number of occurrences with increasing tag-ratio, these graphs confirm the effectiveness of the pruning conditions.

Extensions: For very large corpora, we can further reduce the footprint by specifying a suitable set of "candidate" combinations (e.g., a list of queries and their subsets from a search log) up-front. Moreover, by carefully encoding the representation (similar to [24]) we may be able to trade off rare errors in individual tag ratios against a further reduction in memory footprint.

5. COMPUTING INDICATIVE KEYWORD-COMBINATIONS

In this section, we describe efficient algorithms to compute all indicative keyword combinations for a document corpus $\mathcal{D}$ and

a set of tags $\mathcal{T}$. Our algorithms are targeted at large document corpora such as Wikipedia containing on the order of 10^6–10^7 documents; this in turn means vocabularies $\mathcal{V}$ containing 10^8 distinct words, and consequently a very large space of possible combinations of words.

Problem Statement: Given a corpus $\mathcal{D}$, a set of tags $\mathcal{T}$, compute the tag ratios of all keyword-combinations $q \in 2^{\mathcal{V}}$ that are indicative of a tag $t \in \mathcal{T}$ with respect to $\mathcal{D}$.

Recall that our architecture requires that the set of all indicative keyword combinations as well as the corresponding counters/ratios is sufficiently small to be stored in main memory. In particular, this means that we are able to store the string associated with each combination q we select as well as up to $|\mathcal{T}| + 1$ counters $count_{q,i}, i \in \{0\} \cup \mathcal{T}$ for each combination. Here, $count_{q,0}$ records $frq(q)$ and $count_{q,t}$ how many documents tagged with a specific tag $t \in \mathcal{T}$ contain all keywords in q. We make no such assumption about the the size of the document corpus $\mathcal{D}$, which may exceed the available memory.

Note that the storage requirements for these counters (as well as any additional counters kept for keyword-combinations which are filtered out during execution) are the key constraint regarding the algorithm's efficiency. If main memory were no concern, it is simple to solve this problem by scanning $\mathcal{D}$ once and explicitly keeping track of the $count_{q,i}$ values for *all* candidate keyword combinations in main memory. Unfortunately, this approach is clearly unrealistic given the vocabulary sizes. Therefore, we will consider approaches with significantly lower main memory requirements.

Baseline: The problem of finding all indicative keywords is similar to the problem of computing *frequent itemsets*, which has been studied extensively in the literature (see [15] for a survey of efficient approaches). Hence, as a baseline approach we first describe a simple technique using pruning rules proposed in the context of frequent itemsets, which leverages an *inverted index* on $\mathcal{D}$. The computation of inverted indexes for large document collections is by now a standard operation provided by many IR and database engines; moreover, once an inverted index is in place, the document frequencies of each keyword in $\mathcal{V}$ are known, allowing us to only consider the set $V_\alpha = \{c \in \mathcal{V} | frq(v) \geq \alpha\}$ of keywords which individually satisfy the *support condition*. Because of the distribution of word frequencies discussed in Section 4.1, this alone reduces the search space very significantly. The algorithm then computes the required counts for keyword-combinations by intersecting the inverted indexes in a bottom-up manner; whenever the size of an intersection is known to be less than α, we terminate the processing for this keyword combination and all supersets of it.

The advantage of this very simple algorithm is that is has very small main memory requirements. At any point, we only need to retain the posting lists of up to w_{max} words in main memory, which should be small even for extremely large $\mathcal{D}$. Unfortunately, it is also very slow, even when discounting the initial cost of building the inverted index. The main issue is that each posting list takes part in up to $(|\mathcal{V}(\alpha)| - w_{max})^{w_{max}-1}$ index intersections and hence is iterated over many times. To overcome this issue, we now describe an approach that iterates over the document corpus directly, making only a limited number ($2 \cdot w_{max}$) of passes over the data.

Combination-Filter: Our approach is based on ideas from the area of data streams [4]. The computation of frequent items in data streams with limited main memory and limited numbers of passes over the data has received significant attention in recent years (e.g., [9, 17, 12]); we will leverage an adaptation of the techniques described in [12].

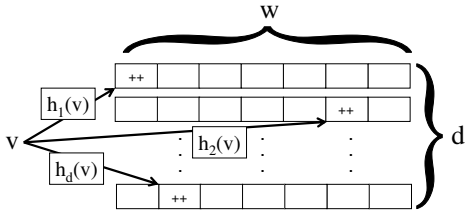

Figure 7: Structure of a CM-Sketch

The algorithm we propose scans the corpus $\mathcal{D}$ a total of $2 \cdot w_{max}$ times; after the $2 \cdot k$-th scan, it outputs all indicative keyword combinations q' (and their ratios) of length (in words) $|q'| = i$. This computation proceeds by alternating between two stages: in the first stage, we scan $\mathcal{D}$ to construct an estimator $\overline{frq}(q)$ of the frequency $frq(q)$ of word-combinations q of k words (the value of k increasing every two scans). As we will show, this estimator only over-estimates frequencies, i.e., $\forall q \subseteq \mathcal{V} : \overline{frq}(q) \geq frq(q)$.

In the 2nd phase we scan $\mathcal{D}$ again then and maintain the value of $count_{q,i}$ for all k-word combinations q for which $\overline{frq}(q) \geq \alpha$. At the end of this scan, we can use these counts to determine all keyword combinations for which the true frequency $frq(q) \geq \alpha$, as well as compute all resulting ratios to validate the support and correlation conditions. Because $\forall q : \overline{frq}(q) \geq frq(q)$, this approach will compute all required tag ratios for the set of indicative keyword combinations, as well as some false positives. The amount of main memory the algorithm consumes depends on the efficiency of the estimator. In the following, we will discuss how we use a modification of *Count-Min Sketches* porposed in [12] to construct an efficient estimator for this task.

Count-Min Sketches: The basic structure we employ to construct an estimator $\overline{frq}(q)$ is a modification of sketching technique called *Count-Min Sketch* (CM-Sketch) [12]. A CM-Sketch is a 2-dimensional array of counters with width w' and depth d': $f\_count[1,1] \ldots f\_count[d', w']$ and d' hash functions $h_1, \ldots, h_{d'} : 2^{\mathcal{V}} \mapsto \{1, \ldots, w'\}$. All counters are set to 0 initially. Whenever a word combination q is inserted into the structure, the basic CM-Sketch algorithm iterates over all hash functions $h_i(q)_{i=1 \ldots d'}$ and increases $f\_count[i, h_i(q)]$ by 1.

Using these counts, we estimate the frequency of a word combination q as $\overline{frq}(q) := \min_{j \in \{1,\ldots,d'\}} f\_count[j, h_j(q)]$. Obviously, it must hold that $\forall q : \overline{frq}(q) \geq frq(q)$, since each f_count cell contains the sum of the frequency of all values hashed to it.

Modified CM-Sketches: Now, we leverage two observations to improve this structure for our specific scenario: first, as discussed in Section 4.1, the frequency-distribution of words and word-combinations roughly follows a power-law. This means that only for a small fraction of word-combinations q does it hold that $frq(q) \geq \alpha$. Hash-collisions between the other keyword combinations in turn will result in a significant number of false positives by the filter. Here, it is particularly important to filter out such false positives – otherwise all of these keyword combinations have to be tracked explicitly in the subsequent processing.

Second, for the purpose of constructing a filter, we don't require individual cells to contain the sum of the frequencies of all keyword combinations hashed to them – instead, if we ensure that $\forall q \in 2^{\mathcal{V}} : f\_count[i, h_i(q)] \leq frq(q)$ (i.e., we are able to keep an upper bound on the maximum value in each counter), the filter will produce the correct output.

We now combine these two observations to modify the original CM-Sketch algorithm: consider the stream of keyword-combinations that is (via the hash-functions) mapped to a specific

counter $f\_count[i,j]$. Note that we are only interested in an upper bound on the maximum frequency of a keyword combination mapped to this counter. This means that if we – when we observe sequence of *distinct* keyword combinations mapped to $f\_count[i,j]$ – only have to increase the counter once. Since keeping track of all distinct values hashed to a particular bucket is clearly not practical (due to the required space) we instead use a small bitmap to approximately keep track of distinct values. We know that the distribution of keyword combinations follows a power law, so we expect many counters to have many low-frequency values mapped to them.

```
CM_UPDATE(q) // Update the modified CM-sketch with item q
// Input: Keyword-combination q
// Output: Updated f_count-array, s.t. ∀q : f_count[i,h_i(q)] ≥ frq(q).
for i ∈ 1,...,d' do
    if b[i,h_i(q)] & 2^ĥ_i(q) = 2^ĥ_i(q) OR f_count[i,h_i(q)] = 0 then
        b[i,h_i(q)] := 2^ĥ_i(q) // New sub-sequence
        f_count[i,h_i(q)] := f_count[i,h_i(q)] + 1
    else
        b[i,h_i(q)] := b[i,h_i(q)] & 2^ĥ_i(q) // Update bit-array
    end if
end for
```

Algorithm 1: Updating CM-Sketch Counters

To implement this idea, we modify the original CM-Sketch structure as follows: with every counter $f\_count[i,j]$ we associate a bit-array $b[i,j]$ of width m, requiring a total of $d' \cdot w' \cdot m$ bits. We associate a second array of hash-functions $\hat{h}_1, \ldots, \hat{h}_{d'} : 2^{\mathcal{V}} \mapsto \{1,\ldots,m\}$ which map a set of keywords to one of the m positions. Now, when we first encounter a keyword-combination q that is hashed to a specific $f\_count[i,j]$, we first test if $b[i,j]$ is 0; if so, we increment $f\_count[i,j]$ and set the $\hat{h}_i(q)$-th bit in $b[i,j]$. For every subsequent item q' we see for this counter, we test if the $\hat{h}_i(q')$-th bit in $b[i,j]$ has already been set; if not, we know that q' has not occurred since we last increased the counter, and only set the $\hat{h}_i(q')$-th in the bit array, but do not increase the counter. Only if we find that the corresponding bit had been set do we increase the counter and set all bits (except for the $\hat{h}_i(q')$-th one) to zero. The algorithm is shown in detail as Algorithm 1.

Putting it all together: We can now formulate the overall approach in Algorithm 2 (showing the two stages only): we first iterate over $\mathcal{D}$ to construct the filter $\overline{frq}()$, which is then used in the subsequent scan to track highly frequent word-combinations. Hence, the memory requirements of our approach depend on the quality of this filter, and we'll evaluate it in Section 6.4. In the next phase, we iterate over $\mathcal{D}$ in a similar manner, enumerating all possible high-frequency word combinations using $\overline{frq}$. When encountering a document containing a keyword combination q for which $\overline{frq}(q) \geq \alpha$, we then use the tag-information stored with d to identify all tags for which we need to increment $Count_{q,t}$.

```
// Stage I: Filter Construction for word-combinations of size k
// Invariant: V_α^{k-1} := {q ∈ 2^V : |q| = k-1 and frq(q) ≥ α}
//            is known from earlier iterations.
for d ∈ D do
    S = ∅
    for i = 1,...|d| do
        w_i := i-th word in d.
        if w_i ∉ S then
            for q' ∈ 2^S ∩ V_α^{k-1} do
                CM_UPDATE({w_i} ∪ q')
            end for
            S := S ∪ w_i
        end if
    end for
end for
// Stage II: Explicit tracking of the ratios for all q that pass the filter
for d ∈ D do
    S = ∅
    for i = 1,...|d| do
        w_i := i-th word in d.
        if w_i ∉ S then
            for q' ∈ 2^S ∩ V_α^{k-1} do
                q := {w_i} ∪ q'
                if frq(q) ≥ α then
                    Count_{q,0} = Count_{q,0} + 1
                    for t ∈ T with t ∈ d do
                        Count_{q,t} = Count_{q,t} + 1
                    end for
                end if
            end for
            S := S ∪ w_i
        end if
    end for
end for
```

Algorithm 2: Combination-Filter Approach

6. EXPERIMENTS

In this section, we evaluate the accuracy resulting from the proposed features, their ability to generalize and quality of the algorithm proposed in Section 5. We evaluate our approach on three different real-life query classification tasks:

Consumer Electronics Classification: This task is identical to the classification of product intent described in Section 1.2 for the product category of *consumer electronics*; as the corpus, we use a 7.2 Million document Wikipedia snapshot tagged with the 157 product categories as described in the example in Section 3.2 and we use 30K web search queries (20K training and 10K test) to train the classifier. The search queries were labeled manually.

Health Query Classification: Here, the classification task is to identify queries that are related to health issues. As $\mathcal{D}$ we use a corpus of 600 Million advertisement bids, and the 5000 most prolific advertisers as the set of tags $\mathcal{T}$; each bid phrase is tagged with the advertiser(s) who submitted this phrase to the corpus. Unlike the previous classification task, the training/test data for this task encompasses over 800K distinct labeled queries; the labels were obtained using a combination of manual and automated labeling.

Retail Classification: This task is identical to the identification of retail queries described in Section 1.2. For this task, we use a combination of the advertiser tags described previously and the Wikipedia category tasks described in Section 1.2: for the latter, we again use Wikipedia as $\mathcal{D}$ and the 1K most frequent page categories as $\mathcal{T}$; for the former, we use only the 1K most prolific advertisers. For this task, we have training/test data of 330K labeled queries; the labels were obtained using a combination of hand-labeling and automated techniques [19].

6.1 Evaluation of Classifier Accuracy

To compare our approach to text-based classifiers, we used *Maximum Entropy* classifiers [22] (which perform well for very sparse feature sets). These classifiers are trained on *n-gram features* extracted from the training-set queries. An n-gram feature is a binary function that indicates the occurrence of a specific n-gram in an input query. In our evaluation, we use $n = 1, 2, 3$ for all experiments, extracting all n-grams found in the training data. As has been shown in [19], n-gram features can lead to remarkable classification performance given abundant training data. As the training

data volume varies significantly between the different classification tasks, we will be able to assess if this affects the relative performance compared to our technique.

In addition, for the consumer electronics classification task, we further sought to improve the (generalization) performance of the text-based classifier by adding *lexicon* features. A lexicon feature is a binary function indicating whether *any* n-gram that belongs to a pre-defined lexicon appears in an input query. Such features typically improve generalization performance, as they can be triggered even by an n-gram that does not occur in the training data. A lexicon is typically constructed as a cluster of semantically similar words/phrases. Here we use four lexicons extracted from a product database, representing *brand*, *model*, *product type* and *product attribute* respectively. A lexicon feature is activated if the input query contains a word that belongs to the corresponding lexicon. The size of *brand*, *model*, *product type* and *attribute* lexicons are 3.8K, 91.3K, 3.1K, 2.8K entries, respectively.

In the experiments described in this section, we only used the simplest form of backoff features, using the tag ratios for single-word queries only. As the total number of distinct words occurring in the corpora are 34 Millions (*Wikipedia*) and 12 Million (*Advertisements*), this means that all tag ratios (even for relatively large tag sets with more than 10^4 tags) can be stored in memory.

For each experiment, we evaluated both the accuracy resulting from using the n-gram classifier and the classifier based on tag ratio features only, as well as the accuracy resulting from combining both feature types. To asses the gain resulting from combining the two feature types; we combined the two classifier's output as follows. Given an input query, the MART and the MaxEnt classifiers each produce a posterior probability. We then trained a meta-classifier (based on MART) using additional, held-out labeled queries, using the two posteriors as the two input features.

For the *Consumer Electronics* task, the n-gram features achieved classification accuracy of 93.02%, which increased to 93.20% when features based on the the various lexica were added. The features based on tag ratios resulted in significant improvement in accuracy: 95.64%. Combining the output of both classifiers improved this to 96.5% accuracy. For the *Health Query* task, using either the n-gram classifier or the classifier using only tag-ratios in isolation resulted in (almost) identical classification accuracies of 98.2%. However, combining the two classifiers again resulted in significantly improved accuracy of 98.8%. For for the *Retail* task, the accuracy of the n-gram based classifier was 92.5%, the accuracy based on tag ratios was 93.3% and combining both classifiers resulted in 94.2% accuracy.

In summary, the combination of tag ratios and n-gram features consistently resulted in significant improvements in the classification accuracy for each of the classification tasks.

6.2 Evaluation: Training Data Size

A second important characteristic of the features we propose is that they can yield accurate classifiers even for very small amounts of training data. This is important, as the acquisition of manually labeled training data is often a bottleneck in real-life applications.

To illustrate this, we varied the size of the training data for each of the studied classification tasks, and compared the resulting accuracy for the both n-gram and tag ratio features described in Section 6.1. We compared the accuracy for training data sizes corresponding to 100%, 33%, 10%, 3% and 1% of the original training examples. The results are shown in Figures 8–10.

As we can see, reducing the training data size to as little as 1% of the original training data results in only a small reduction in classification accuracy for the classifier based on tag ratio features,

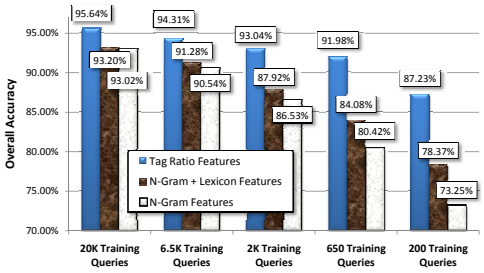

Figure 8: Generalization of Featuresets in Product Categorization.

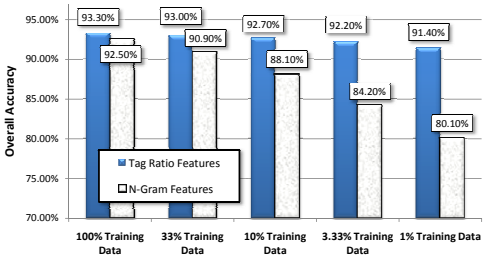

Figure 9: Generalization of Featuresets in Retail Categorization.

whereas it significantly impacts the n-gram based one and also, to a lesser degree, the classifier incorporating lexicon features.

6.3 Evaluation: Feature Sets

In this experiment we evaluate the impact of using indicative keyword combinations only on classification accuracy and the number of keyword combinations for which we need to store tag ratios. We use the following set-up: for the *Retail* classification task we compute the ratios for all sub-queries of length up to 3 which occur in the test training data. We then evaluate the accuracy resulting from using 3 different feature sets: (a) features based on the full set of the computed ratios, (b) features based the ratios for all single words and ratios for all indicative keyword combinations ($\alpha = 50, \beta^t_{low} = 0.8, \beta^t_{high} = 1.2$) and (c) the ratios for all single word queries only. The accuracies obtained with these three features sets are as follows: the full feature set had the best accuracy at 93.47%, followed by the indicative features (93.38%) and the single-word features (93.30%). As we can see, the difference between these features sets is not very pronounced. Moreover, while the full feature set gives us the best accuracy overall, materializing all possible 3-word queries is not feasible in practice, even for much smaller document sets (again, due to the power-law distribution). Here, the indicative features offer comparable accuracy and even the choice of using single keywords does not result in a big drop in classifier performance. We ran a similar experiment for the *Health* classification task: here, the difference in accuracy between features based on (a) all ratios and (b) single words was even smaller: less than 0.10%.

For the *Retail* classification task, we measured the fraction of all generated keyword-combinations that were filtered out by the support and correlation conditions: the set of indicative keyword combinations contained only 0.8% of the set of all 3-word combinations in the training data. Here, the value of α has the most impact on the number of combinations: setting α to 100 prunes all but 0.3% of the combinations, whereas $\alpha = 20$ retains 2.5%. Note that these factors are likely to underestimate the reduction in size we would obtain when considering all possible 3-word subqueries,

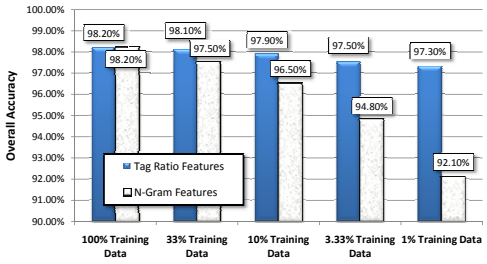

Figure 10: Generalization of Featuresets in Health Categorization.

as we only use keywords that occur in the labeled training data, which is more likely to contain relatively common words.

6.4 Evaluation: Indicative Keyword Mining

In this section we briefly evaluate the algorithms proposed in Section 5. For this purpose, we used varying subsets of Wikipedia ranging from 10K to 200K documents as well as varying values of $\alpha \in \{100, 200, 500, 1K\}$ and evaluated the estimator quality when using Algorithm 2 to compute all 2-keyword combinations satisfying the support-condition. All experiments were run on an 2.21 Ghz Opteron 2214 PC with 16 GB memory.

For the (modified) CM-Sketches, we used 16-bit counters and bit-array widths $m \in \{0, 2, 3, 4, 8\}$ – the setting $m = 0$ corresponds to the standard CM-Sketch. To allow for a fair comparison, the space required for the bitmaps was subtracted from the total space available for the counters. After initial experiments we only used $d' = 1$ hash functions; larger values always resulted in reduced performance. Now, in each experiment, we set the total size of the sketch as equivalent to storing 2% of $|V_\alpha|^2$ (i.e. 2% of all possible 2-keyword combinations satisfying the support threshhold). We also used the total frequency of a word over all of $\mathcal{D}$ for pruning, meaning, we considered – for each document – the same set of keyword combinations we would have considered if we were processing the entire corpus.

Comparing the different filters, we found that the modified CM-Sketch performed best for $m = 2$ and $m = 3$, outperforming the unmodified sketch (in terms of the number of keyword-combinations retained during the 2nd scan) by between 2–17%. The modified sketch using bit-arrays of width $m = 4, 8$ performed worse than the original structure.

7. CONCLUSION AND OUTLOOK

In this paper we have proposed a new class of *tag ratio* features for query classification based on co-occurrence between various types of tags and query terms in large document corpora. These features allow us to avoid sparse feature spaces, and result in significant improvements in overall classification accuracy. We proposed the notion of *backoff ratios* which allow us to accurately classify queries for which the corresponding tag ratios are not known, by leveraging the ratios of smaller sub-queries. This in turn allows us to set up a framework in which we pre-compute the tag ratios for a small subset of all possible queries only. Because we can index this subset in main memory, it becomes possible to realize the features based on tag ratios with small latency, making the approach attractive for use within the tight latency constraints of search engines. Experiments using various real-life classification tasks and different tag sets show that combining the proposed features with n-gram based classifiers yields significant gains in accuracy over using n-gram features alone, even when very large sets of labeled examples are available.

8. REFERENCES

[1] http://blogs.zdnet.com/BTL/?p=3925.

[2] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of Evidence for Vertical Selection. In *ACM SIGIR*, 2009.

[3] H. Baayen. *Word Frequency Distributions*, volume 18 of *Text, Speech and Language Technology*. Kulver Academic Publishers, 2001.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J.Widom. Models and Issues in Data Stream Systems. In *Proceedings of the ACM PODS Conference, Madison, USA*, pages 1–30, 2002.

[5] S. M. Beitzel, E. C. Jensen, O. Frieder, D. D. Lewis, A. Chowdhury, and A. Kolcz. Improving Automatic Query Classification via Semi-Supervised Learning. In *ICDM Conf.*, pages 42–49, 2005.

[6] A. Z. Broder, P. Ciccolo, M. Fontoura, E. Gabrilovich, V. Josifovski, and L. Riedel. Search Advertising using Web Relevance Feedback. In *CIKM*, pages 1013–1022, 2008.

[7] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust Classification of Rare Queries using Web Knowledge. In *ACM SIGIR*, pages 231–238, 2007.

[8] J. Brutlag. Speed Matters for Google Web Search. http://code.google.com/speed/files/delayexp.pdf, 2009.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *ICALP*, 2002.

[10] S. Chaudhuri, K. Church, A. C. König, and L. Sui. Heavy-Tailed Distributions and Multi-Keyword Queries. In *ACM SIGIR*, 2007.

[11] K. Collins-Thompson and P. Bennett. Estimating Query Performance using Class Predictions. In *ACM SIGIR*, 2009.

[12] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: the Count-Min Sketch and its Applications. In *Journal of Algorithms, 55(1)*, pages 58–75, 2005.

[13] J. Friedman. Greedy Function Approximation: a Gradient Boosting Machine. *Annals of Statistics*, 29(5), 2001.

[14] A. Fuxman, A. Kannan, A. B. Goldberg, R. Agrawal, P. Tsaparas, and J. Shafer. Improving Classification Accuracy using Automatically extracted Training Data. In *ACM SIKDD*, pages 1145–1154, 2009.

[15] B. Goethals and M. J. Zaki. Advances in Frequent Itemset Mining Implementations: report on FIMI'03. *SIGKDD Explor. Newsl.*, 6(1), 2004.

[16] X. He, J. Yan, J. Ma, N. Liu, and Z. Chen. Query Topic Detection for Reformulation. In *WWW Conference*, 2007.

[17] R. Karp, S. Schenker, and C. Papdimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. In *ACM Transactions of Database Systems, 28(1)*, pages 51–55, 2003.

[18] A. C. König, K. Church, and M. Markov. A Data Struture for Sponsored Search. In *IEEE ICDE*, 2009.

[19] X. Li, Y.-Y. Wang, and A. Acero. Learning Query Intent from Regularized Click Graphs. In *In Proc. ACM of SIGIR*, 2008.

[20] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[21] D. Metzler, S. T. Dumais, and C. Meek. Similarity Measures for Short Segments of Text. In *ECIR*, 2007.

[22] K. Nigam. Using Maximum Entropy for Text Classification. In *In IJCAI-99 Workshop on Machine Learning for Information Filtering*, 1999.

[23] D. Shen, J.-T. Sun, Q. Yang, and Z. Chen. Building Bridges for Web Query Classification. In *SIGIR*, 2006.

[24] D. Talbot and T. Brants. Randomized Language Models via Perfect Hash Functions. In *Proceedings of ACL-08: HLT*, 2008.

[25] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Ranking, Boosting, and Model Adaptation. Technical report, Microsoft Research, 2008.

[26] W. Yih and C. Meek. Improving Similarity Measures for Short Segments of Text. In *AAAI*, 2007.