Entity Categorization Over Large Document Collections

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

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

Rares Vernica
Information and Computer
Sciences
University of California, Irvine
Irvine, CA 92697-3425
rvernica@uci.edu

ABSTRACT

Extracting entities (such as people, movies) from documents and identifying the categories (such as painter, writer) they belong to enable structured querying and data analysis over unstructured document collections. In this paper, we focus on the problem of categorizing extracted entities. Most prior approaches developed for this task only analyzed the local document context within which entities occur. In this paper, we significantly improve the accuracy of entity categorization by (i) considering an entity's context across multiple documents containing it, and (ii) exploiting existing large lists of related entities (e.g., lists of actors, directors, books). These approaches introduce computational challenges because (a) the context of entities has to be aggregated across several documents and (b) the lists of related entities may be very large. We develop techniques to address these challenges. We present a thorough experimental study on real data sets that demonstrates the increase in accuracy and the scalability of our approaches.

Categories and Subject Descriptors

H.2.8 [**Database Application**]: Data Mining; I.5.4 [**Pattern Recognition**]: Text Processing

General Terms

Algorithms, Performance, Experimentation

1. INTRODUCTION

Information extraction from large document collections has received significant amount of attention recently. A variety of extraction tasks have been considered: identifying and extracting named entities from documents [4], detection of topics/themes [20], extraction of customer preferences [5], etc. The extracted information is used in a variety of ways, e.g., to provide business analytics or to answer more sophisticated queries [8]. Entity extraction technology has matured and commercial technology (from Verity, Inxight, etc.) is available for identifying various types of entities such as people, organizations, locations. In addition, a large number of solutions to this task have been proposed in research [3, 11].

One particular area of recent interest has been the automatic extraction of unary relations (such as *is-a-painter*, *is-a-researcher*, or *is-a-camera*) and binary relations (such as *is-a-painter-of*, *is-author-of*) between named entities (e.g., [1, 6, 15, 23]). Here,

we differentiate between two approaches: *"open" relation extraction* [6] where arbitrary relations are extracted and *targeted relation extraction* where only a small number of known target relations (e.g., actors, painters, electronic products) are extracted.

In this paper, we focus on the extraction of targeted relations. We view the targeted relation extraction as that of *categorizing* named entities, into a set of target classes such as *painters*, *researchers*, etc. Henceforth, we use the terms *unary relation extraction* and *entity categorization* interchangeably.

1.1 Multi-Context Extraction

Most approaches for unary relation extraction from large document collections base their extraction decisions on the "context" of an entity – a window of words around the entity occurrence – within a *single* document, using rules or machine-learning models to map a context to an extracted target relation. Once an extraction decision has been made, the context information is discarded [3]. While this approach is fast, as only a single traversal through the document corpus is required, accuracy of the extraction is limited by the fact that only very limited information is taken into account before extraction. To illustrate why this is the case, consider the example scenario where we simply want to classify person names into two classes: *is-a-researcher* and *is-non-researcher*. Approaches that consider one context at a time are limited to triggering extraction rules on contexts that allow us to identify a person as a researcher with high confidence from a single context, e.g.,

> *"[Entity] works in research"* → (*[Entity],is-a-researcher*).

Because the context is lost once an extraction decision has been made, any further decision regarding the reliability of a given extraction then has to be based on the number of times a particular extraction has been made , and cannot take into account which contexts were used to give rise to an extraction.

In contrast, first aggregating all contexts for a specific entity allows us to think of each individual context as generating one or more features of this entity, allowing us to subsequently combine several less predictive features to arrive at a high-confidence extraction. For example, we can use the combination of features such as *"[Entity] presents results"* and *"[Entity] publishes"*, each of which is not sufficiently predictive by itself to allow extraction of the tuple (*Entity,is-a-researcher*) (after all, companies present results and newspapers publish), but which – when combined – make it very likely that the entity in question is a researcher.

In this paper, we build upon the above insight. We propose that the contexts within which an entity occurs across multiple documents be "aggregated" and then used to categorize an entity. Informally, the aggregated context is the union of all contexts within which an entity occurs. We identify features from this aggregated

context, which an underlying machine-learning model then uses to categorize the entity. As we illustrated above, aggregation across multiple documents allows us to also leverage several low confidence features instead of using only high confidence features.

1.2 Leveraging Existing Entity Lists

In many scenarios lists of entities related to the target relation are readily available. We can exploit these lists to significantly improve the accuracy of relation extraction by using features that reflect the existence of these related entities in the context of an entity we seek to classify. For example, consider the task of categorizing people as athletes and an example document "*Yao Ming is drafted by the National Basketball Association.*" The input entity recognizer may recognize "*Yao Ming*" as a person name. If we have the list of sports organizations then recognizing the occurrence of one of its members "*National Basketball Association*" is a very useful feature for classifying Yao Ming as an athlete. We refer to such features that reflect the co-occurrence between entites we want to classify and lists of "known" entities as *list-membership* features. The required lists can often be obtained from structured data sources (such as Wikipedia or IMDB) or can be derived from the training data itself.

The main reason why these features make an impact is that the lists allow us to aggregate many instances of a category list into a single feature; while many individual entries in these known lists occur too rarely in documents to become important features in a classifier individually, the resulting list-membership features are sufficiently common to do so. For example, a feature which says that authors are likely to be mentioned along with a book is more likely to be applicable than several different features corresponding to instances of specific books.

1.3 Challenges

The use of multi-document contexts and list-membership features can benefit extraction accuracy significantly, but it also introduces new challenges. Generally, the document corpus we extract from is significantly larger than the available main memory. This means in turn that during extraction we most likely have to consider very large numbers of combinations of entities and their contexts, all of which we cannot store in main memory either. Further, the relevant lists of "known" entities themselves, which we need to identify list-membership features, may also require more space than is available in main memory. This issue is acerbated when extracting many relationships in parallel.

Straight-forward implementations of unary relation extraction thus require either (i) multiple passes over the document corpus, invoking the potentially expensive entity extraction multiple times for each document, or (ii) need to materialize significant amounts of entity-context on disk. Both of these processing strategies may be very expensive computationally. Our approach avoids these costs by leveraging knowledge of the underlying feature semantics and statistical properties of large document corpora.

The remainder of the paper is organized as follows: In Section 2, we review related work. In Section 3, we describe the problem of unary relation extraction in detail. We then describe techniques for efficiently extracting features over the aggregated context (Sections 4 and 5). Extensions are discussed in Section 6. We present an experimental evaluation in Section 7, and conclude in Section 8.

2. RELATED WORK

A large amount of recent work has focused on extracting entities and relations from large document collections (e.g., [1, 6, 15, 23]). While these techniques vary in the amount of processing they perform for a given occurrence of an entity (e.g., they determine parts
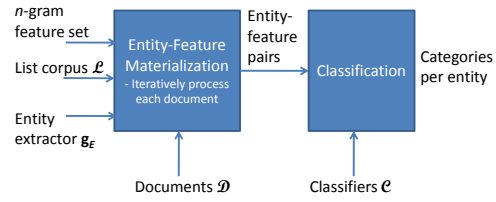

Figure 1: Overview of our approach

of speech of the words around the entity, label head nouns, and often look up dictionaries), most of these techniques only analyze the context of an entity within an individual document when extracting a relation. Our approach is novel in that it exploits the contexts of entities that span multiple documents.

Recently, a large number of techniques have been proposed for speeding up the processing of large document collections in relation extraction (see [1] for an overview), including (heuristic) filtering of documents [2], the use of very simple extraction patterns and specialized index structures [9]. These approaches are orthogonal to ours and may be combined with it.

The problem of computing the list-membership and cross-document features also has strong similarity to the processing of join and aggregation queries in database systems, where such queries have been studied extensively (see [16] for an overview). Our problem scenario differs in that the extraction of features from documents itself is a central part of the problem statement, something that is currently not supported by relational DBMS.

Our techniques to scale the feature extraction leverage statistical properties found in large corpora of natural language text. As shown recently in [17] for different but related tasks, using the underlying statistical properties of skewed or even heavy-tailed distributions can result in much improved processing strategies when processing large document data sets.

3. ENTITY CATEGORIZATION

In this section, we will study the problem of scalable extraction of unary relations from very large text corpora, when using both aggregated contexts as well as list-membership features for classification. For this purpose, we will first define the necessary notation and subsequently set up the problem of unary relation extraction.

3.1 Definitions and Notation

Throughout the paper, we will use the following notation. Let $\mathcal{D} = \{d_1, \ldots, d_m\}$ be the set of input documents. Within these documents, we recognize entities using an entity recognizer g_E, i.e., g_E takes a document $d \in \mathcal{D}$ and returns all named entities of the desired type (say, persons, organizations, or locations) occurring in d. Let $\mathcal{E} = \bigcup_{d \in \mathcal{D}} g_E(d)$ denote the set of all entities, which is unknown when we begin processing the documents. Our approach is not tied to any specific *entity recognition* technique – we can use any approach capable of recognizing occurrences of named entities in a document based on the document and other domain information (e.g., [19, 25]).

Surface forms vs. canonical forms of entities: When referring to entities we need to differentiate between the entity itself and its various *surface forms* (i.e., the different ways to refer to this particular entity in a document). For example, "*Bill_Gates*" "*William H. Gates*" and "*William Gates III.*" all refer to the same person. The problem of mapping entity surface forms to entities has been studied in various domains (e.g., [24]); a detailed discussion of such techniques is beyond the scope of the paper. Henceforth, we as-
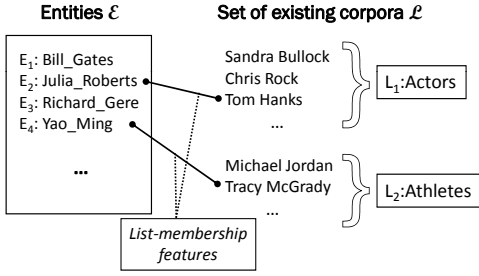
Figure 2: Entity/List-membership graph

Figure 3: Entity/Feature graph

sume that each entity identified is also converted to its *canonical* representation. In the above example, the representations "*William H. Gates*" and "*William Gates*" would be converted to their normalized canonical form, say, *Bill_Gates*.

Document context and aggregated context: In the context of entity categorization, we identify the features occurring in the "*document context*" of an entity. In general, the document context of an entity may involve many aspects of the document such as title, paragraph headers, and the words around the entity occurrence. In this paper, we only consider a window of words around an entity's occurrence. Suppose an entity e (with a surface form s_e of length $|s_e|$ words) occurs in a document d at position p. The *size-K context* of an entity e with respect to d and position p is the window $w_{p-K}, \ldots, w_{p-1}, w_{p+|s_e|+1}, \ldots, w_{p+|s_e|+K}$ of words around the entity occurrence. The union of all size-K contexts of an entity e within a document d form the *document context* for e with respect to d. Suppose the entity *Picasso* occurs in a document containing one sentence: "The painting by Picasso adorns the main hall of the Museum of Art in New Delhi." Then, the size-3 context of Picasso is: "the painting by [Entity: Picasso] adorns the main." The *size-K aggregated context* for an entity e with respect to $\mathcal{D}$ is the union of all size-K document contexts of e in all $d \in \mathcal{D}$ in which e occurs.

Classifiers and features: We denote the set of all classifiers we use for unary relation extraction as $\mathcal{C} = \{c_1, \ldots, c_l\}$. Each classifier c_i is trained to extract one specific relation from the aggregate context of an entity. Specifically, each classifier c_i takes as input a set of features $Features(c_i)$, which are extracted from the appropriate aggregated context. We denote the set of all features $\mathcal{F} = \bigcup_{c \in \mathcal{C}} Features(c)$. We distinguish 2 types of features.

Text n-gram features: One set of features we consider are occurrences of word n-grams, which are identified during the training phase of the classifier to be highly correlated with the target category. For example, the phrase "*painting by*" may be very correlated with the *painters* category. All n-gram features we consider in this paper are binary, i.e., only the absence/presence of an n-gram in an aggregate context is indicated, but not its frequency; however, our techniques can be extended to non-binary features as well.

List-Membership features: The second set of features we consider are the list-membership features discussed earlier. To identify these, we use a set of lists $\mathcal{L} = \{L_1, \ldots, L_f\}$, where each L_i is a list of known entities. Regarding list-membership features, we also primarily consider only binary features in this paper; however, it is possible to extend our techniques to list-membership features that are sensitive to the number of occurrences. Note that the width (K_2) of the context size around each entity for generating list-membership features may be different than that (K_1) for n-gram features. Since it is usually clear which aggregate context we refer to we usually drop the prefixes (K_1 or K_2) while referring to aggregated contexts.

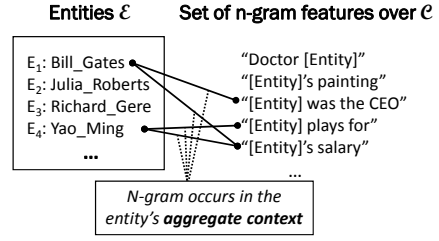We can now define the problem of unary relation extraction.

Relation Extraction Problem *Given a set $\mathcal{D}$ of documents, an entity recognizer g_E, a set $\mathcal{C}$ of classifiers trained on a set $\mathcal{F}$ of features, a set of existing lists $\mathcal{L}$, the task of unary relation extraction is to classify all $e \in \mathcal{E}$ using the classifiers contained in $\mathcal{C}$.*

3.2 Our Approach

We decompose the relation extraction problem into two components as shown in Figure 1. The first component, *Entity-Feature materialization*, generates the set of features, both n-gram and list membership features, grouped by the corresponding entities across all documents. The second component, *Classification*, takes this set of features as an input and then applies the classifiers in $\mathcal{C}$. A large number of different classifiers has been proposed for categorization in research – in this paper, we do not take a position on the particular classification models used. Instead, we will focus on the extraction of features and their materialization, which is challenging due to the sheer volume of data involved.

Conceptually, the task of list-membership feature computation is to compute the bipartite graph $G_{E,L}$ between entities and the lists L_i. $G_{E,L}$ contains a node for each entity $e \in \mathcal{E}$ and a node for each list $L_i \in \mathcal{L}$. $G_{E,L}$ contains an edge (e, i) whenever there exists a list-member $l \in L_i$ that occurs in the K_2-context of e in a document $d \in \mathcal{D}$ containing e. Figure 2 shows an example bipartite graph. We refer to edges in $G_{E,L}$ as *entity-list* edges.

Similarly, the task of n-gram feature materialization is to compute a bipartite graph $G_{E,F}$ between entities and n-gram text features (Figure 3). $G_{E,F}$ contains a node for each entity $e \in \mathcal{E}$ and a node for each n-gram feature $f \in \mathcal{F}$. $G_{E,F}$ contains an edge (e, f) whenever the feature f occurs in the aggregated context of e across all documents in $\mathcal{D}$. We refer to these edges as *entity-feature* edges.

Figure 4 gives an overview of our architecture. It has four components: *list-membership extraction*, *n-gram feature extraction*, *aggregation*, and *classification*. The list-membership extraction component materializes $G_{E,L}$; the n-gram feature extraction component materializes $G_{E,F}$. The aggregation component groups all features by each entity. The classification component processes each group per entity applying each classifier in $\mathcal{C}$ over the set of features in the group. The output of this component consists of entities and their recognized categories.

The main challenge for the materialization of $G_{E,L}$ and $G_{E,F}$ is scalability: both $\mathcal{D}$ and $\mathcal{L}$ are very likely to exceed in size the main memory available during unary relation extraction, meaning we cannot retain one of them in memory while scanning the other. We will formalize the resulting challenges and describe how to address them in the following two sections.

4. PROCESSING LIST-MEMBERSHIP

In this section, we describe techniques for the computation of list-membership features, i.e., materialization of $G_{E,L}$. For the purpose of list-membership processing, we model each document $d \in \mathcal{D}$ as a set of entity, list-member pairs: $d \subseteq \mathcal{E} \times \mathcal{L}$, with d

containing a pair (e, l) if a list-member l is contained in the K_2-context of e in document d. A straight-forward approach to implement this functionality would be to scan the documents while keeping $\mathcal{L}$ in main memory, extracting entities and list-membership features from each document. The combinations of entities and list-membership features would then be written to disk; after processing all of $\mathcal{D}$, all such features would be grouped together for each entity. However, this approach has two main issues.

First, the set of lists $\mathcal{L}$ may be too large to fit into memory during the document processing. For example, lists of *actors* (useful for identifying categories such as *directors*, *producers*, or even *movies*) or *paper-titles* (which – when grouped by area or conference – can be very useful for categorizing researchers into areas) are readily available from web sources and contain more than a million distinct entries. Also, we want to apply multiple classifiers from $\mathcal{C}$ in parallel, each of which may require different lists of known entities, all of which compete for the limited main memory.

Second, this approach for materializing will produce many identical (and thus redundant) edges in the graph $G_{E,L}$. For example, a popular athlete such as "Michael Jordan" may be mentioned a large number of times along with many members (such as NBA) in a list of sports organizations.

4.1 The Need for Large Lists

While it is easy to establish that $\mathcal{L}$ may indeed contain a large number of related list members, we also need to ensure that the large number of list members actually makes a difference in classification; otherwise, a simple strategy such as reducing $|\mathcal{L}|$ by only retaining the most important list members may suffice. To test if this is the case, we set up the following experiment. Using a known set of directors (as $\mathcal{E}$) and a list of actors (as $\mathcal{L}$) and 3.2 million documents from Wikipedia, we measured the number of entities $e \in \mathcal{E}$ that co-occur with at least one member of $\mathcal{L}$. Note that all such entity/member pairs would result in an edge in $G_{E,L}$. We compare this to the number of entities that co-occur when subsets of the most frequent members (in the corpus) of $\mathcal{L}$ are used. We consider subsets of sizes 1%, 2%, 5% and 10% of $|\mathcal{L}|$ here.

We observed that using a subset of $\mathcal{L}$ sharply reduces the number of entities for which we see (at least one) co-occurrence: in case of a list containing the 1% most frequent members, we "miss" about 44% of such entities, i.e., the missed entities do not have a list-membership feature indicating that their context in some document contains a known actor. The number of missed entities becomes 38% when we consider the 2% most frequent members, 30% when we consider 5% most frequent members, and 25% when we consider 10% most frequent members. Since – as we will show in Section 7.2.1 – list-membership features have a significant impact on accuracy, this experiment illustrates the requirement for all members in $\mathcal{L}$ to be considered.

4.2 Extracting Members from Large Lists

There are a number of execution strategies available to us when computing $G_{E,L}$ for lists $\mathcal{L}$ larger than the main memory available:
Multi-Scan Approach: A straight-forward approach to address the issue of space when computing $G_{E,L}$ for large $\mathcal{L}$ would be to divide $\mathcal{L}$ into k sublists (e.g., via hashing), each of which is sufficiently small to fit in the available memory and then to iterate k times over $\mathcal{D}$, each time keeping a different sublist of $\mathcal{L}$ in memory. This strategy has two problems. (1) The document collection $\mathcal{D}$ itself is very large, meaning that we have to read in a significant amount of text multiple times. (2) This strategy also requires multiple invocations of the potentially expensive entity-extractor g_E[1], as well as all ex-

---

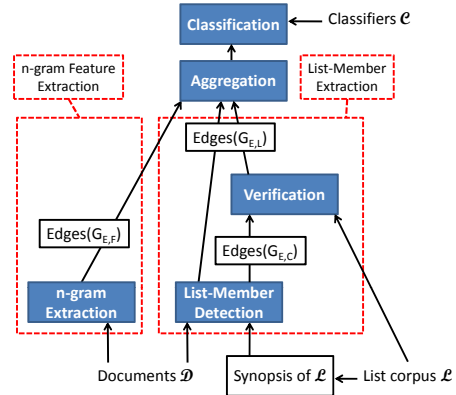[1] A test run using a commercial extractor required 14 hours to process a

Figure 4: Overview of our approach

ecuting all the required preprocessing (such as tokenizing, parsing, etc.).

As a consequence, this multi-scan approach is typically too expensive in practice and we will not consider this execution strategy.
Scanning $\mathcal{D}$ once: A simple strategy that avoids multiple scans of $\mathcal{D}$ would be to scan $\mathcal{D}$ and write all out pairs of entities and their contexts to disk for further processing. However, this strategy is not efficient, due to (a) the large amount of data written and (b) the fact that the vast majority of contexts are not expected to contain an entity that is a member of a list in $\mathcal{L}$, meaning that most of the written contexts are irrelevant.
Our Approach: Our solution instead writes out a significantly smaller subset of contexts by constructing smaller, approximate set-representations – which we call *filters* – of each $L_i \in \mathcal{L}$ (e.g., a *Bloom Filter*) which together fit in main memory. These approximate set representations allow us to test substrings in entity contexts for being members of L_i with no false negatives (but some false positives), allowing us to prune most substrings and contexts right away. We refer to the substrings that are accepted by at least one of the filters and subsequently written to disk as *candidate contexts*. However, the existence of false positives means that we have to "verify" if such a candidate context does contain a list member.

This leads to the processing strategy described in Figure 4: we process each document $d \in \mathcal{D}$ using the filters to identify all entities with candidate contexts which may contain a list member. Because of false positives, the candidate contexts then undergo a subsequent verification-step that identifies all contexts containing members of $\mathcal{L}$ and writes out all resulting entity and list-membership feature pairs. As part of the document processing, we also extract n-gram text features from each document (which we'll describe in detail in Section 5). Finally, all features found (i.e., both list-membership features and n-gram features) are grouped per entity (we refer to this as the *aggregation*-step) and then submitted to the classifiers in $\mathcal{C}$.
Challenges: To make the above processing strategy efficient, we have address two challenges: (a) efficiently identifying all candidate contexts in each document and (b) minimizing the number of "redundant" candidate contexts and features written out. We will describe these challenges and how we address them in detail in the following two sections. We first describe the processing of individual documents using filters (Section 4.3) and then the pruning of redundant contexts and features (Section 4.4); finally, we describe the implementation of the verification step itself (Section 4.5).

---

million Wikipedia documents while it just takes a few minutes to scan them.

4.3 Detection of Candidate List Members

Identifying list members within an entity's context corresponds to identifying whether sub-strings in the context match with any member of a list in $\mathcal{L}$.[2] Therefore, recognizing list-membership features within an entity's context is similar to the *multi-pattern matching problem* in the string matching literature [21]. Most prior techniques for solving the multi-pattern matching problem (e.g., the *Aho-Corasick* algorithm) build a *trie* over all list members. The trie is used to significantly reduce the number of comparisons between subsequences of words in an input document and patterns.

Recall that we use *filter*-structures, based on *bloom filters* [7], which are compact probabilistic structures for representing set S of elements. Given an element e, bloom filters allow us to probabilistically check whether or not e belongs to S. If $e \in S$, then the bloom filter returns true. However, if $e \notin S$ it may still return true with a low probability. Let $Filter_{L_i}$ denote bloom-filter representation of the list L_i.

For a substring c in a document we write $c \in Filter_{L_i}$ if c is accepted by the filter. We can trade off the required memory-footprint for the expected false positive rate of the filter, thereby ensuring that the $Filter_{L_i}$ structures can all fit in main memory.
Document Processing using Filters: Straight-forward usage would be to check *all* substrings in entity contexts against each filter. In order to reduce the number of these membership checks against each filter, we also maintain a *token table* TT_{L_i}, which consists of all *tokens* (e.g., single words) occurring in any member of the list L_i. We then further sub-divide a given context c into *hit sequences*, which are sequences of tokens present in TT_{L_i}. We now need to evaluate membership of token sub-sequences against the filters for these hit sequences only, as opposed to all substrings of the context. Similar to Bloom Filters, the token table can be compressed by hashing, allowing us to trade off memory against additional membership checks.

4.4 Pruning Redundant Output

We now introduce some notation. The output of the document processing described in the previous section is a list of pairs of entities and candidate contexts, which are accepted by one or more of the filters. We use $G_{E,C}$ to denote the bipartite graph between entities and candidate contexts. Conceptually, $G_{E,C}$ contains a node for each entity $e \in \mathcal{E}$ and a node for each candidate string in an entity context which is accepted by a filter. $G_{E,C}$ contains an edge (e, c) whenever the candidate context c is accepted by some filter $Filter_{L_i}$ and occurs in the K_2-context of e in a document $d \in \mathcal{D}$. We refer to these (e, c) edges as *entity-candidate* edges and denote the list of all such edges that we materialize to disk during the processing as $Edges(G_{E,C})$. After a single pass over $\mathcal{D}$, all members of $Edges(G_{E,C})$ undergo a verification-step to identify all candidate contexts that actually contain a member of $\mathcal{L}$ (see Section 4.5). All entity-list combinations $(e, i) \in G_{E,L}$ identified during the verification are written to disk; we use $Edges(G_{E,L})$ to denote the list of these combinations.
Problem Statement: *Given a document collection $\mathcal{D}$ and a set of lists $\mathcal{L} = \{L_1, \ldots, L_f\}$, compute the set of all co-occurrences of entities and (members of) lists in $\mathcal{L}$: $G_{E,L} = \{(e, i) \in \mathcal{E} \times \{1, \ldots, |\mathcal{L}|\} | \exists l \in L_i : (e, l) \in D\}$ using the minimal overhead under the constraints that (a) each document in D can only be processed once, (b) during the processing only a limited amount of state (bounded by a threshold M) may be retained in memory.*
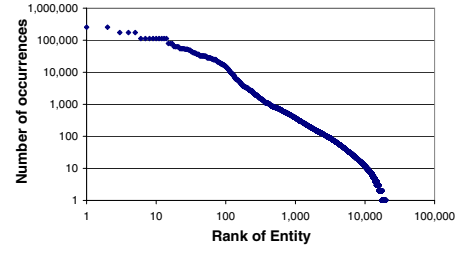
Figure 5: The Frequency-Distribution of Entities follows a power-law

In our processing framework, the overhead is closely tied to the amount of disk I/O required. Hence, in order to minimize the overhead of the processing, we need to minimize (a) the number of redundant (e, c) entity-candidate pairs (i.e., the pairs that do not result in a new (e, i)-edge in $Edges(G_{E,L})$) written to $Edges(G_{E,C})$ and (b) the number of redundant edges written to $Edges(G_{E,L})$.

In the following, we will describe two different pruning strategies, based on the observed skew in the number of occurrences in list members and entities themselves, that allow us to avoid writing a significant number of edges to $Edges(G_{E,C})$ and to $Edges(G_{E,L})$, thereby reducing the processing overhead significantly.

4.4.1 Leveraging Skew in List-Member Frequencies

Looking at list-member distributions within Wikipedia for a number of entity categories, we observed that the distribution of entities in large corpora is similar to that of the word distribution found in natural-language text corpora: both (approximately) exhibit a power-law. Figure 5 illustrates the distribution of named entities found in a large corpus of Wikipedia documents. Only a small fraction of list members occur very frequently in documents, whereas most occur rarely. Furthermore, documents containing members of a list L_i often contain more than one member. Many list-members frequently co-occur with one of the few frequent members in their list.

We can leverage these properties as follows: during the scan of $\mathcal{D}$, we keep small sets $Prune_{L_i} \subseteq L_i$, each containing a subset of members from L_i that frequently occur in $\mathcal{D}$. To determine the most frequent list members, we use a sample of $\mathcal{D}$ to estimate their relative frequencies. Because of the significant skew of the underlying distribution, a small sample of $\mathcal{D}$ is likely to be sufficient for a high-confidence estimate of the most frequent entities. Now, when processing a document d containing an element $l \in Prune_{L_i}$ in the document context of an entity e, we can write out the resulting (e, i) tuple directly to $Edges(G_{E,L})$, while avoiding writing any (e, c) tuples for any $c \in Filter_{L_i}$ and the document context.

The key question for this optimization becomes if a small number of frequent entities can make a big difference with regards to pruning. To understand this, we set up the following experiment: Using a set of 1.7 million known *actors*, and a corpus of documents $\mathcal{D}$ from Wikipedia, we first compute the set of all members of $\mathcal{L}$ contained in $\mathcal{D}$. Then we selected subsets of the 1%, 2%, 5% and 10% most frequent members of $\mathcal{L}$ and measured for each the fraction of candidates (using filters without false positives for this experiment) that are pruned by having the corresponding $Prune_{\mathcal{L}}$ list available during processing; we use the entire size of each document as the size (K_2) of the entity-context for list-membership extraction. The results are plotted in Figure 6; note that we vary both the size $|\mathcal{D}|$ of the document corpus used as well as the size $|Prune_{\mathcal{L}}|$ of the pruning set. Even small sizes of $Prune_{\mathcal{L}}$ result in

---

[2]Note that our approach can generalize to allow approximate matches between sub-strings in a document context and list members to identify entity-list edges. We can adopt recently developed approximate matching techniques for this purpose (e.g., [10]).
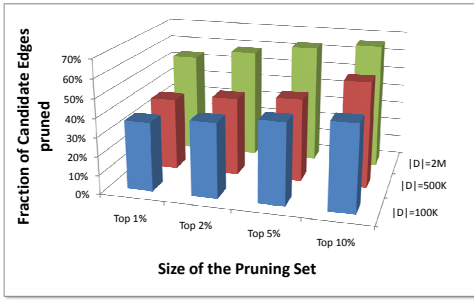
Figure 6: Small sets of frequent entities result in significant pruning.

Figure 7: Frequency-Distribution of Features

significant pruning. For example, selecting the 10% most frequent list-members means that more than half of the candidate edges are pruned. Note that the relative pruning effectiveness increases with the size of the document corpus.

4.4.2 *Leveraging Skew in the Entity Distribution*

A second observation we leverage uses the skew of the entity-distribution. If we know that (e, i) is in $Edges(G_{E,L})$, any candidate context $c \in Filter_{L_i}$ (that passes none of the other filters) occurring in the context of e can only result in a redundant edge (e, i). Therefore, we can ignore such candidates, saving the corresponding entity-candidate edges. For this reason we maintain in memory the most frequently occurring entities identified by g_E. For each of these entities we maintain all list-ids for which we have observed a list-member (using the $Prune_{L_i}$ structures) in the context of e. We refer to this set of entities as $Prune_{\mathcal{E}}$; now, when we encounter a string $c \in Filter_{L_i}$ in the context of an entity e for which $(e, i) \in Prune_{\mathcal{E}}$, we can ignore this observation. However, one challenge is that we cannot compute (an estimation) of the entities' frequencies a priori, as $\mathcal{E}$ is not known. Recall that we only have an entity extractor g_E as input which when applied to documents extracts entities.

Estimation of entity frequency: Keeping exact frequency-information for *all* entities identified in documents requires a significant amount of main memory. Therefore, we resort to a very space-efficient hash-based approximation scheme to track entity frequencies. We employ a 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 $h_1, \ldots, h_{d'} : \mathcal{E} \mapsto \{1, \ldots, w'\}$. All counters are set to 0 initially. Whenever g_E extracts an entity e within a document, we iterate over all hash functions $h_i(e)_{i=1 \ldots d'}$ and increase $f\_count[i, h_i(e)]$ by 1. We disregard multiple occurrences of an entity within a single document.

Using these structures, we estimate the frequency of an entity e as $\hat{frq}(e) := \min_{j \in \{1, \ldots, d'\}} f\_count[j, h_j(e)]$. This estimate will not be exact because multiple entities may map to the same bucket due to hash-collisions. Further, the entity frequency estimates are only based on the subset of documents we observed at any point during processing. As is standard in many online estimation scenarios, we assume that "current" frequency is indicative of the frequency over the entire input collection. We are primarily interested in identifying highly frequent entities and CM-sketches have been shown to perform well for the task of identifying such outliers [13].

Allocating memory between structures: The fact that the filter structures as well as $Prune_{L_i}$ and $Prune_{\mathcal{E}}$ compete for the same limited main memory available during the document processing
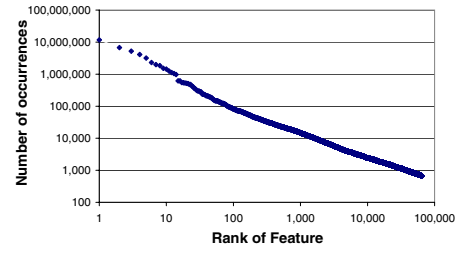
makes the tuning of their sizes an interesting research challenge. We leave this as a challenge for future work.

4.5 Verification

We eliminate all false positives edges in $G_{E,C}$ as follows. Note that each edge consists of the entity and the candidate string which potentially equals a list member. Therefore, we join (*equi-join* in SQL) $\mathcal{L}$ with $Edges(G_{E,C})$ in order to obtain all list member values which equal a candidate string in $Edges(G_{E,C})$. In fact, if the lists $\mathcal{L}$ and $Edges(G_{E,C})$ are both stored in a database, it is straightforward to remove the false positives through a SQL query, which can be executed efficiently by the database system. For every member of a list L_i found, we write the corresponding (e, i)-tuple to the materialized list $Edges(G_{E,L})$.

5. EXTRACTION OF TEXT FEATURES

Unlike the lists $\mathcal{L}$, we expect the number of n-gram features used in the classifiers to be small enough to retain a *trie* of the n-grams used to identify feature occurrences in main memory during the document processing. While processing a document d we match any n-gram features occurring in the document context of an entity e and write the corresponding (e, f) pairs to $Edges(G_{E,F})$.

This means that processing of n-gram features only requires a single pass over $\mathcal{D}$. However, many entity-feature edges may be written to disk multiple times from different documents. We found the distribution of the occurrences of the n-grams associated with the n-gram features over large corpora of documents to resemble a power-law as well. To illustrate this, we have plotted the total occurrence-frequencies (for the entire corpus of Wikipedia) of the 32K text n-grams which were most common adjacent to entities (in training data) in Figure 7. This distribution implies that a small subset of entity-feature combinations is likely to be extremely frequent. By processing each of these combinations in isolation, we would create many copies of identical (and thus redundant) edges in $Edges(G_{E,F})$. If instead we we were to store the frequent entity-feature combinations in memory during the document processing and only write them out at the end of processing once, we would be able to significantly reduce the number of redundant copies of edges are written to $Edges(G_{E,F})$.

Problem Statement: n**-gram Feature Extraction** *Given a document-corpus $\mathcal{D}$ and a set of n-gram features $\mathcal{F}$, compute the respective bipartite graph $G_{E,F}$ containing each edge $(e, f) \subseteq \mathcal{E} \times \mathcal{F}$ for each entity-feature pair occurring within $\mathcal{D}$ (and those edges only), while minimizing the number of redundant entity-feature edges generated.*

The main issue is that there is a constraint M_{agg} on the space we can allocate to entity-feature pairs in memory. In the following, we denote the space required to store an entity e by $size(e)$, whereas we assume unit size to store a feature ID. Now, in a static formulation where we know all frequencies $frq(e, f)$ a priori, the problem

of minimizing the number of redundant edges would be to find a set $\mathcal{R}$ of entity-feature pairs that satisfy the memory-constraint M_{agg} and maximizes the number of times these pairs occur, since every occurrence beyond the first one corresponds to a redundant edge.

$$\underset{\substack{\mathcal{R} \subseteq \mathcal{E} \times \mathcal{F} \\ \sum_{e \in Entities(\mathcal{R})} size(e) + |\mathcal{R}| \leq M_{agg}}}{\operatorname{argmax}} \sum_{(e,f) \in \mathcal{R}} \operatorname{frq}(e, f). \qquad (1)$$

However, we do not know which (e, f) pairs will be observed as we process documents in the input collection. Hence, we cannot solve the static problem formulation, but need to optimize the expected benefit of the entity-feature pairs we keep in memory dynamically. Techniques of this kind have been studied extensively in the context of main memory caching algorithms. However, our problem scenario is somewhat different because the objects that are "cached" (i.e., the entity-feature pairs) are so small that the space-requirement of retaining statistics on each them is difficult to justify. To maximize the utility of the available memory, we evaluated two different classes of approaches.

Write-on-Full: At any point in time, we add each entity-feature pair seen to the available main memory (without storing any duplicate features). Once the main memory store fills up, we write *all* (e, f) pairs contained in it to $Edges(G_{E,F})$ and start over. The advantages of this algorithm are its simplicity and robustness.

Leveraging frequency estimation: In addition, we evaluated a number of techniques that estimate frequencies of entities (using a CM-sketch similar to the one used in list-membership processing) and those of features (using a small sample of $\mathcal{D}$). For this purpose, we evaluated a number of different heuristics to decide which pairs to retain, including (i) keeping the features of the most frequent entities, (ii) retaining the most frequent entity-feature pairs themselves and (iii) maximizing the expected benefit per entity relative to the required storage space (which favors retaining frequent entities with many features). Due to space constraints, we omit the details of the algorithms we evaluated.

6. EXTENSIONS

Combined n-gram and list-membership features: One interesting extension we are considering for future work is to combine the two types of features considered in this paper to form features such as *born_in_[European_City]* or *works_for_[Tech_Company]* where the text in brackets denotes a list whose members we match to obtain the features (e.g., the first feature would match text strings such as "*born in Paris*"). We can extend our technology easily to process these additional features as well.

n-ary relation extraction: Due to space constraints we did not describe or evaluate the extraction of n-ary relationships (for $n > 1$) in detail. However, it is possible to extend our techniques to this scenario. We need to extend the notion of an entity's context to contexts of a set of entities (i.e., a set of entities occurring within a limited window of words within a document d); the corresponding notions of document- and aggregate context follow directly. Regarding the processing strategies, we need to use the corresponding set of entities as the key for grouping features (instead of a single entity) and for purposes of list-member co-occurrence.

7. EXPERIMENTAL EVALUATION

7.1 Experimental Setup

For our experiments, we use linear support vector machine (SVM) models as the underlying classifier. The training algorithm we use for our experiments is *Sequential Minimal Optimiza-*

tion [22]. All experiments were executed on an 2.4Ghz Intel dual-core 660 Processor with 4GB of main memory. We used a *SQL Server 2005* database as the underlying storage engine. We use 3.2 million page Wikipedia dataset as the document corpus $\mathcal{D}$.

7.2 Evaluation of Impact on Accuracy

To study the impact of aggregate context and list-membership features in entity-categorization, we compare our approach to single-context rule-based extractors similar to several state-of-the-art techniques used for fast relation extraction from documents. The classification task is to correctly categorize different classes of people (*writers, actors, painters, etc.*) based on Wikipedia documents. To generate training and test data, we used Wikipedia documents that contain lists of instances of the respective category (e.g. `http://en.wikipedia.org/wiki/-List_of_painters_by_name`), from which instances of the category were extracted using specialized extraction-script. For purposes of repeatability, we – instead of using custom entity extraction code g_E – matched entities in $\mathcal{D}$ by simply finding these names within each document. To compute the n-gram features, we extracted all n-grams in the size-4 *document context* for each occurrence of an entity, using the presence/absence of the 10K most frequent (i.e., occurring in the largest number of contexts within the training data) n-grams among them as features. For the list-membership features, we used the entire document an entity e occurs in as e's aggregate context, using a 10% sample of the entities in the training data for each category as the corpus $\mathcal{L}$.

To instantiate the single-context rule-based classifier with extraction patterns that are highly indicative of the different categories, we used the same document contexts as above as training data and used the algorithm described in [18] (Section 3.2) to determine the $100K$ most frequent text patterns (which may include gaps) that are highly correlated to the entity type. An example of such a pattern containing a gap would be " *'s . . . novel*", which would match strings such as "*'s latest novel*" or "*'s acclaimed novel*" which may be highly correlated to instances of the class *writer*. Note that the algorithm is exhaustive, in the sense that it outputs the most frequent patterns that have high correlation to a class over the space of all possible patterns. When we have two patterns in our result set where one pattern is a substring of the other, we prune the more complex pattern, avoiding large numbers of similar extraction rules.

Now, we compare the precision/recall tradeoff for the resulting classifiers. To measure precision/recall for our approach, we applied the classifier to held out test data and varied the distance from the decision surface we required to assign a category label. Entities that were "too close" to the decision surface were not labeled.

In unsupervised or weakly supervised rule-based extraction, the frequency (also referred to as redundancy) of extraction has commonly been used to assess the confidence assigned to an extracted relation (e.g., [6, 14]). Hence, we use the following approach to obtain precision/recall data for the single-context classifier: we set a threshold on the number of extractions we have to have seen for a specific *(entity, category)* pair, before we accept it. By sliding this threshold across the domain of extraction frequencies, we can generate a similar precision/recall graph for the rule-based extraction.

Figure 8 shows the results for two example target classes – *painters* and *presidents*. We can see that the rule-based classifier generates rules with high accuracy, but cannot guarantee high recall: many of the entities in our training data do not exhibit a single instance of a frequent high-confidence pattern. Overall, the classifiers based on aggregate contexts perform better, in particular the ones that leverage list-membership features. We consistently saw these gains across all target classes. For the rule-based classifiers,
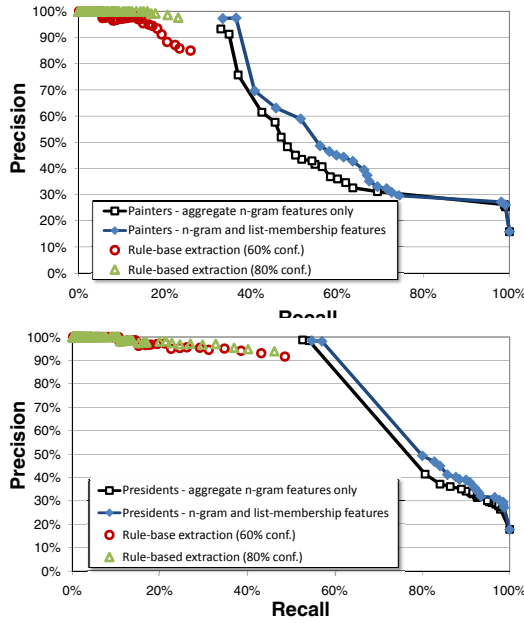
Figure 8: Accuracy of various techniques

Figure 9: List-Membership impacts "hard" classes significantly

Figure 10: Reduction in total edges

we also varied the minimum correlation to the target label that was required for a rule to be output – our graphs show the results for two settings (60% and 80%). Requiring lower correlation reduces accuracy, but results slightly higher recall; even so, there is no crossover point with the classifiers using aggregate context features.

7.2.1 Impact of List-Membership Features alone

To study the impact of list-membership features in scenarios where there is no aggregation across many documents, we ran the following experiment: using a set of 21K products from a existing shopping web site, we studied the task of classifying these into the 7 categories , one of which each of them belonged to: *CD & DVD Drives*, *T-shirts*, *Laptop Computers*, *Books (Drama)*, *Books (Thriller)*, *Books (History)*, *Books (Romance)*. The motivation behind this particular choice of categories was that some of these are very "easy" to distinguish (e.g. t-shirts and laptops) whereas the various subcategories of books are very difficult to classify accurately. For classification, we used SVM-classifiers trained on: (a) word n-grams extracted from the product names and descriptions and (b) word n-grams plus seven list-membership features – one for each category in the product description (as $\mathcal{L}$, we used a sample of 10% of the entities in the training data), using Wikipedia as the corpus from which we derived co-occurrences between entities and list-members. Here, the second classifier increased the accuracy over all entities and categories from 93.8% to 95.5%.

Interestingly, while the precision-recall curves for the categories *CD & DVD Drives*, *T-shirts*, *Laptop Computers* were nearly unchanged for the two classifiers, we saw a significant increase in accuracy for the book-categories, all of which were more difficult to classify accurately. We plotted two of the resulting precision/recall graphs in Figure 9. As we can see, the improvement in accuracy for the *Books - Drama* category is very significant (whereas the ones for t-shirts are slight), resulting in the overall gains.

7.3 Evaluation of List-Member Extraction

To evaluate the effect of our techniques for extraction of list-membership features for large $\mathcal{L}$, we use the similar setup as previ-
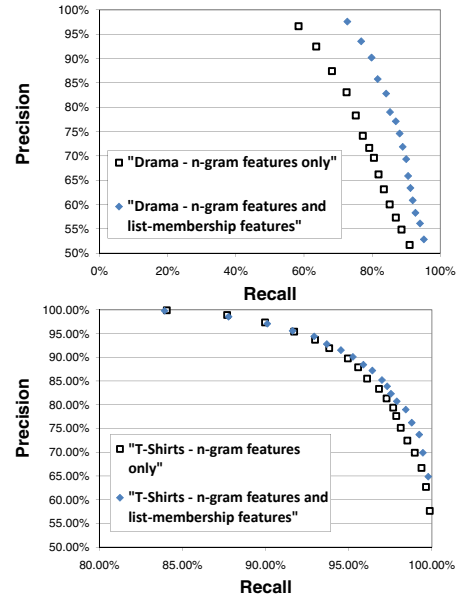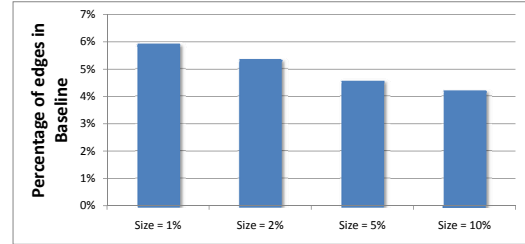
ously, and use a corpus of 1.7 million known *actors* as $\mathcal{L}$. For transparency of the results, we "simulate" the accuracy of the $Filter_{\mathcal{L}}$ structure, using a rate of false positives to true matches of entities in $\mathcal{L}$ of 4:1. First, we measured the number of edges in $Edges(G_{E,C})$ written to disk for different sizes of $Prune_{\mathcal{E}}$ and $Prune_{\mathcal{L}}$, varying the amount of space given to $Prune_{\mathcal{E}}/Prune_{\mathcal{L}}$ between 1% and 10% of the space required to store all entities/list-members occurring at least once in $\mathcal{D}$. As the baseline, we use an approach that writes all edges in $Edges(G_{E,C})$ without any pruning (we do not introduce any false positives for this baseline). Even retaining only a small subset of frequent entities from $\mathcal{E}$ and $\mathcal{L}$ results in significant reduction in the number of redundant edges (Figure 10).

To quantify the impact this has on execution times in practice we measured the overhead of (a) the final aggregation and sort step for all entity-feature edges in $G_{E,F}$ and $G_{E,L}$, (b) the verification step that matches all edges in $G_{E,C}$ with $\mathcal{L}$ and (c) the overhead imposed during the the iteration over $\mathcal{D}$ by writing out the edge-sets to disk (measured by computing the difference in time to the processing of $\mathcal{D}$ without persisting any edges), using a relational DBMS for the aggregation and verification steps. The results are shown in Figure 11. We can see that our techniques reduced the overhead of verification by more than an order of magnitude and halved the overhead of disk writes, reducing the overall processing cost by over 80% compared to the baseline. The remaining overhead is dominated by the cost of writing out the edges in $Edges(G_{E,F})$ and aggregating them, the reduction of which we will discuss next.
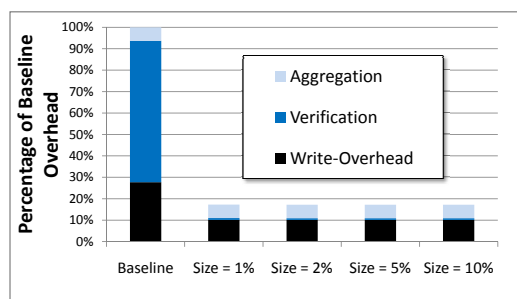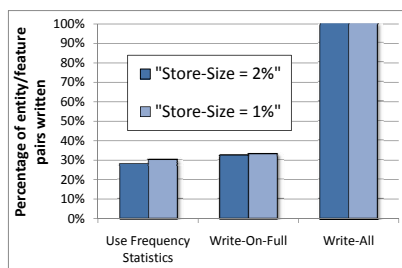
Figure 11: Reduction in overhead

Figure 12: Write-on-Full is competitive with complex techniques

7.4 Evaluation of n-gram Extraction

To assess the effectiveness of the techniques discussed in Section 5, we used a single classifier using 10K n-gram features. To choose the appropriate memory-size to use in these experiments, we pre-computed the size required to store $G_{E,F}$ in its entirety (without any duplicate edges) and set the memory size to a fraction of this value (we used 1% and 2% in our experiments). Figure 12 shows the number of entity-feature pairs written for three different strategies: *Write-All*, which is the baseline that uses no additional memory, *Write-on-Full* which we described in Section 5 and the best of the more complex strategies utilizing statistical information on entity- and feature-frequency, which seeks to minimize redundant edges by retaining the entities with the largest (estimated) benefit divided by the size required to store the entity and all its features. We set the amount of storage given to the underlying CM-Sketch to 5% of the available memory. We can see that both *Write-on-Full* as well as the more complex strategy reduce the number of tuples written to disk by a factor of about 1/3. The main reason for this result is the heavy skew of both the entity/feature distribution: the most frequent pairs are so frequent that they will re-occur (often many times) before *Write-on-Full* has to to write the buffer to disk, meaning that we can realize most of the benefit of the "caching" without frequency-statistics. One of the complex strategies outperforms *Write-on-Full* slightly, but given its simplicity and robustness *Write-on-Full* is still likely the method of choice in practice. Moreover, *Write-on-Full* actually outperformed other statistics-based approaches, such as retaining the entities that were (estimated to be) most frequent.

8. CONCLUSIONS

Entity categorization within large text corpora is of importance to a number of emerging applications such as structured querying over unstructured data. For this task, we have shown that the use of aggregate entity contexts and list-membership features in classification can improve accuracy significantly, when compared to techniques using single contexts only. We have shown how to scale up the processing of these aggregate features significantly by leveraging properties of the entity/feature distribution as well as entity co-occurrence, identifying a number of effective techniques to minimize the amount of redundant data produced.

9. REFERENCES

[1] E. Agichtein. Scaling Information Extraction to Large Document Collections. *IEEE Data Eng. Bull.*, 28(4):3–10, 2005.
[2] E. Agichtein and L. Gravano. Querying Text Databases for efficient Information Extraction. In *ICDE*, 2003.
[3] E. Agichtein and S. Sarawagi. Scalable Information Extraction and integration. In *ACM SIGKDD*, 2006.
[4] D. E. Appelt and D. Israel. Introduction to Information Extraction Technology. *IJCAI-99 Tutorial*, 1999.
[5] N. Archak, A. Ghose, and P. G. Ipeirotis. Show me the Money!: Deriving the Pricing Power of Product Features by Mining Consumer Reviews. In *ACM SIGKDD*, pages 56–65, 2007.
[6] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, pages 2670–2676, 2007.
[7] B. Bloom. Space/Time Tradeoffs in Hash Coding with Allowable Errors. In *Communications of the ACM 13(7)*, pages 422–426, 1970.
[8] M. Cafarella, M. Banko, and O. Etzioni. Relational Web Search. In *WWW Conference*, 2006.
[9] M. J. Cafarella and O. Etzioni. A Search Engine for Natural Language Applications. In *WWW Conference*, 2005.
[10] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient Batch top-k Search for Dictionary-based Entity Recognition. *IEEE ICDE Conf.*, 2006.
[11] W. Cohen and A. McCallum. Information Extraction and Integration: an Overview. In *SIGKDD*, 2004.
[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] G. Cormonde and S. Muthukrishnan. What's Hot and What's Not: Tracking Most Frequent Items Dynamically. *ACM TODS*, 30(1):249–278, 2005.
[14] D. Downey, O. Etzioni, and S. Soderland. A Probabilistic Model of Redundancy in Information Extraction. In *IJCAI*, 2005.
[15] R. Feldman, B. Rosenfeld, S. Soderland, and O. Etzioni. Self-supervised Relation Extraction from the Web. In *ISMIS*, 2006.
[16] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.*, 25(2), 1993.
[17] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To Search or to Crawl?: towards a Query Optimizer for Text-Centric Tasks. In *SIGMOD*, pages 265–276, 2006.
[18] A. C. König and E. Brill. Reducing the Human Overhead in Text Categorization. In *SIGKDD*, 2006.
[19] A. McCallum and W. Li. Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-Enhanced Lexicons. In *CoNLL*, 2003.
[20] Q. Mei and C. Zhai. A Mixture Model for Contextual Text Mining. In *ACM SIGKDD*, pages 649–655, 2006.
[21] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings*. Cambride University Press, 2002.
[22] J. Platt. Fast Training of SVM's Using Sequential Minimal Optimization. In *Advances in Kernel Methods: Support Vector Machine Learning*, pages 185–209. MIT Press, 1999.
[23] B. Rosenfeld, R. Feldman, M. Fresko, J. Schler, and Y. Auman. TEG - A Hybrid Approach to Information Extraction. In *Proceedings of the 2004 CIKM Conference*, 2004.
[24] W. Winkler. The State of Record Linkage and Current Research Problems. Technical report, U.S. Bureau of the Census, 1999.
[25] G. Zhou and J. Su. Named Entity Recognition using an HMM-based Chunk Tagger. In *ACL*, 2002.