Facilitating Document Annotation using Content and Querying Value

Eduardo J. Ruiz [#1], Vagelis Hristidis [#2], Panagiotis G. Ipeirotis [*3]

[#] *Department of Computer Science & Engineering*
University of California at Riverside, USA
[1]eruiz009@cs.ucr.edu
[2]vagelis@cs.ucr.edu

[*] *Leonard N. Stern School of Business*
New York University, USA
[3]panos@stern.nyu.edu

Abstract—A large number of organizations today generate and share textual descriptions of their products, services, and actions. Such collections of textual data contain significant amount of structured information, which remains buried in the unstructured text. While information extraction algorithms facilitate the extraction of structured relations, they are often expensive and inaccurate, especially when operating on top of text that does not contain any instances of the targeted structured information. We present a novel alternative approach that facilitates the generation of the structured metadata by identifying documents that are likely to contain information of interest *and* this information is going to be subsequently useful for querying the database. Our approach relies on the idea that humans are more likely to add the necessary metadata during creation time, if prompted by the interface; or that it is much easier for humans (and/or algorithms) to identify the metadata when such information actually exists in the document, instead of naively prompting users to fill in forms with information that is not available in the document. As a major contribution of this paper, we present algorithms that identify structured attributes that are likely to appear within the document, by jointly utilizing the content of the text *and* the query workload. Our experimental evaluation shows that our approach generates superior results compared to approaches that rely only on the textual content or only on the query workload, to identify attributes of interest.

1 Introduction

There are many application domains where users create and share information; for instance, news blogs, scientific networks, social networking groups, or disaster management networks. Current information sharing tools, like content management software (e.g., Microsoft SharePoint), allow users to share documents and annotate (tag) them in an ad-hoc way. Similarly, Google Base [1] allows users to define attributes for their objects or choose from predefined templates. This annotation process can facilitate subsequent information discovery. Many annotation systems allow only "untyped" keyword annotation: for instance, a user may annotate a weather report using a tag such as "*Storm Category 3*".

Annotation strategies that use attribute-value pairs are generally more expressive, as they can contain more information than untyped approaches. In such settings, the above information can be entered as $(StormCategory, 3)$. A recent line of work towards using more expressive queries that leverage such annotations, is the "pay- as-you-go" querying strategy in Dataspaces [2]: In Dataspaces, users provide data integration hints at *query* time. The assumption in such systems is that the data sources *already* contain structured information and the problem is to match the query attributes with the source attributes.

Many systems, though, do not even have the basic "attribute-value" annotation that would make a "pay-as-you-go" querying feasible. Annotations that use "attribute-value" pairs require users to be more principled in their annotation efforts. Users should know the underlying schema and field types to use; they should also know when to use each of these fields. With schemas that often have tens or even hundreds of available fields to fill, this task becomes complicated and cumbersome. This results in data entry users ignoring such annotation capabilities. Even if the system allows users to arbitrarily annotate the data with such attribute-value pairs, the users are often unwilling to perform this task: The task not only requires considerable effort but it also has unclear usefulness for subsequent searches in the future: who is going to use an arbitrary, undefined in a common schema, attribute type for future searches? But even when using a predetermined schema, when there are tens of potential fields that can be used, which of these fields are going to be useful for searching the database in the future?

Such difficulties results in very basic annotations, if any at all, that are often limited to simple keywords. Such simple annotations make the analysis and querying of the data cumbersome. Users are often limited to plain keyword searches, or have access to very basic annotation fields, such as "*creation date*" and "*owner of document*."

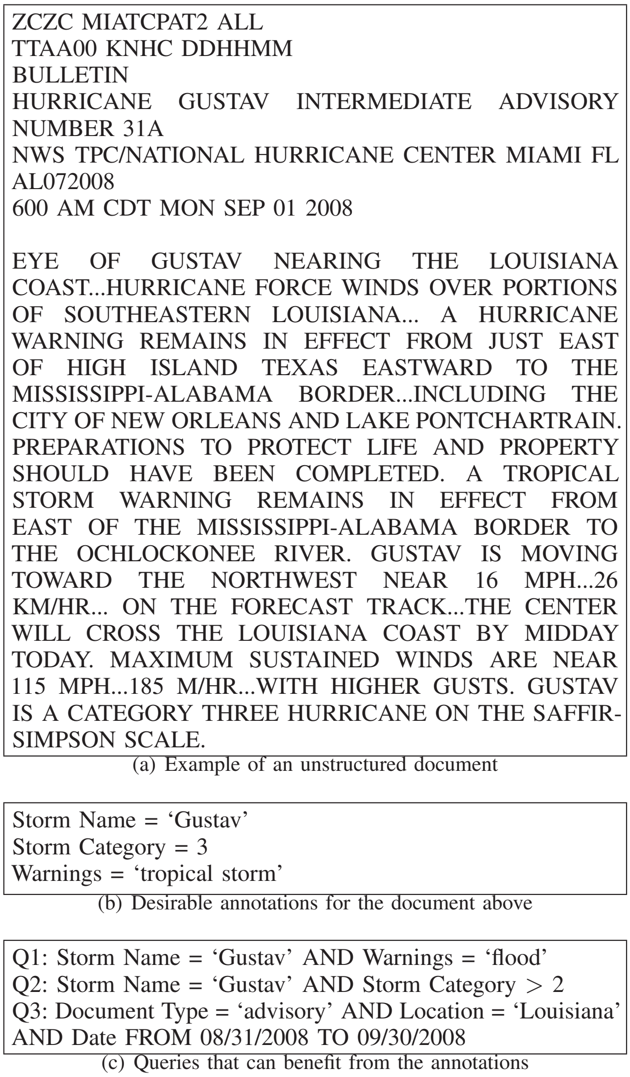
In this paper, we propose CADS (Collaborative Adaptive Data Sharing platform), which is an *"annotate-as-you-create"* infrastructure that facilitates *fielded* data annotation. A key contribution of our system is the direct use of the query workload to direct the annotation process, in addition to examining the content of the document. In other words, we are trying to prioritize the annotation of documents towards generating attribute values for attributes that are often used by querying users.

Example 1: Our motivating scenario is a disaster management situation, inspired by the experience in building a Business Continuity Information Network [3] for disaster situations in South Florida. During disasters, we have many users and organizations publishing and consuming information. For example, in a hurricane situation, local government agencies report shelter locations, damages in structures, or structural warnings. Meteorological Agencies report the status of the hurricane, its position and particular warnings. Business owners describe the status and needs of their stores and personnel. Volunteers share their activities and look for critical needs. The information produced and consumed in this domain is dynamic and unpredictable, and agencies have their own protocols and formats of sharing data, e.g., the Miami-Dade County Emergency Office publishes hourly document reports. Further, learning the schema from previous disasters is hard, as new situations, needs, and requirements arise.

In Figure 1(a) we show a report extracted from the National Hurricane Center repository, describing the status of a hurricane event in 2008. The report gives the current storm location, wind speed, warnings, category, advisory identifier number, and the date it was disclosed. Even though this is a text document, it contains implicitly many attribute names and values, e.g., $(StormCategory, 3)$*. If we had these values properly annotated (e.g., as in Figure 1(b)), we could improve the quality of searching through the database. For instance, Figure 1(c) shows three sample queries for which the report of Figure 1(a) is a good answer and the lack of the appropriate annotations makes it hard to retrieve it and rank it properly.*

The goal of CADS is to encourage and lower the cost of creating nicely annotated documents that can be immediately useful for commonly issued semi-structured queries such as the ones in Figure 1(c). Our key goal is to encourage the annotation of the documents at creation time, while the creator is still in the "document generation" phase, even though the techniques can also be used for post-generation document annotation. In our scenario, the author generates a new document and uploads it to the repository. After the upload, CADS analyzes the text and creates an adaptive insertion form. The form contains the best attribute names given the document text and the information need (query workload), and the most probable attribute values given the document text. The author (creator) can inspect the form, modify the generated metadata as- necessary, and submit the annotated document for storage.

We should note that inserting fielded metadata is not the only scenario in which the CADS strategies are applicable. Consider the case of processing the documents *after* the hurricane, in order to identify and extract important metadata from the documents, so that this is information can be used efficiently in the future (e.g., using a Dataspaces approach).



ZCZC MIATCPAT2 ALL
TTAA00 KNHC DDHHMM
BULLETIN
HURRICANE GUSTAV INTERMEDIATE ADVISORY NUMBER 31A
NWS TPC/NATIONAL HURRICANE CENTER MIAMI FL AL072008
600 AM CDT MON SEP 01 2008

EYE OF GUSTAV NEARING THE LOUISIANA COAST...HURRICANE FORCE WINDS OVER PORTIONS OF SOUTHEASTERN LOUISIANA... A HURRICANE WARNING REMAINS IN EFFECT FROM JUST EAST OF HIGH ISLAND TEXAS EASTWARD TO THE MISSISSIPPI-ALABAMA BORDER...INCLUDING THE CITY OF NEW ORLEANS AND LAKE PONTCHARTRAIN. PREPARATIONS TO PROTECT LIFE AND PROPERTY SHOULD HAVE BEEN COMPLETED. A TROPICAL STORM WARNING REMAINS IN EFFECT FROM EAST OF THE MISSISSIPPI-ALABAMA BORDER TO THE OCHLOCKONEE RIVER. GUSTAV IS MOVING TOWARD THE NORTHWEST NEAR 16 MPH...26 KM/HR... ON THE FORECAST TRACK...THE CENTER WILL CROSS THE LOUISIANA COAST BY MIDDAY TODAY. MAXIMUM SUSTAINED WINDS ARE NEAR 115 MPH...185 M/HR...WITH HIGHER GUSTS. GUSTAV IS A CATEGORY THREE HURRICANE ON THE SAFFIR-SIMPSON SCALE.

(a) Example of an unstructured document

Storm Name = 'Gustav'
Storm Category = 3
Warnings = 'tropical storm'

(b) Desirable annotations for the document above

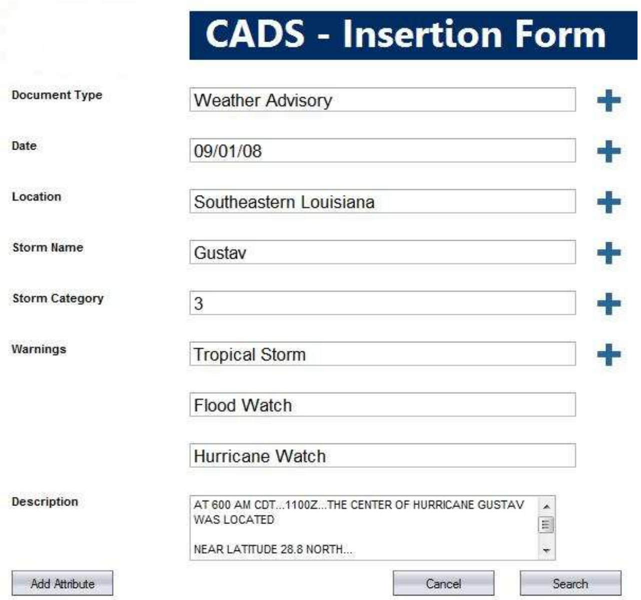
Q1: Storm Name = 'Gustav' AND Warnings = 'flood'
Q2: Storm Name = 'Gustav' AND Storm Category > 2
Q3: Document Type = 'advisory' AND Location = 'Louisiana' AND Date FROM 08/31/2008 TO 09/30/2008

(c) Queries that can benefit from the annotations

Fig. 1. Sample Document and Annotations.

If we use automated information extraction algorithms to extract targeted relations from the document (e.g., addresses of evacuated buildings), it is important to process only documents that actually *contain* such information: when we process documents that do *not* contain the targeted information and we use automated information extraction algorithms to extract such fields, we often face a significant number of false positives, which can lead to significant quality problems in the data [4]. Similarly, if the documents are processed by humans (i.e., where there is low probability of false positives), asking humans to inspect documents where no relevant information is present is expensive and counterproductive. For example, if only 1% of the documents contains information about the address of evacuated buildings, it is going to be unnecessarily expensive to ask humans to inspect *all* documents to identify such information: It is much better to target and process only promising documents, with high probability of containing relevant information.

Going back to our disaster management motivating scenario, after the user submits the hurricane advisory document of Figure 1(a), CADS analyzes the content and



CADS - Insertion Form

Document Type	Weather Advisory
Date	09/01/08
Location	Southeastern Louisiana
Storm Name	Gustav
Storm Category	3
Warnings	Tropical Storm Flood Watch Hurricane Watch
Description	AT 600 AM CDT...1100Z...THE CENTER OF HURRICANE GUSTAV WAS LOCATED NEAR LATITUDE 28.8 NORTH...

Fig. 2. Adaptive Insertion Form.

finds that the following attributes types are relevant and present in the document: “Storm Name,” “Storm Category,” and “Warnings.” Figure 2 presents the adaptive insertion form for that document. The system adds the suggested attributes to a set of default attributes like: “Document Type,” “Date,” and “Location,” which are the basic metadata that the user always provides, as defined by a domain expert. This adaptive generation of metadata forms allows for much more streamlined metadata generation. (Of course, the user can also add new attributes, which are not suggested by the adaptive form.) As we are going to see later, our CADS system prioritizes and suggests first attribute types that are used frequently by users that issue queries against the database.

In short the contributions of this paper are:

- We present an adaptive technique for automatically generating data input forms, for annotating unstructured textual documents, such that the utilization of the inserted data is maximized, given the user information needs.
- We create principled probabilistic methods and algorithms to seamlessly integrate information from the query workload into the data annotation process, in order to generate metadata that are not just relevant to the annotated document, but also useful to the users querying the database.
- We present extensive experiments with real data and real users, showing that our system generates accurate suggestions that are significantly better than the suggestions from alternative approaches.

The rest of the paper is structured as follows: Section 2 describes the framework of our approach. Then, we describe our techniques for identifying important attributes within the document (Section 3). Then we present the implementation details in Section 4. Section 5 presents our experimental evaluation. Section 6 presents the related work and Section 7 concludes.

2 Framework and Problem Definition

In this section, we present the notation that we use in the rest of the paper and describe the problem setting. As discussed in Section 1, our goal is to suggest annotations for a document. We define a document d as a pair (d_t, d_a), composed of the textual content d_t and the set of existing user annotations d_a. We use d_a^{opt} to denote the complete and optimal set of annotations for d. The d_a^{opt} serves as a conceptual baseline, i.e., is created by an oracle with perfect knowledge of the domain of d (e.g., disaster management) and, of course, d_a^{opt} is unknown to the algorithm that is trying to estimate as accurately as possible the d_a^{opt}.

Each annotation $\mathcal{A}$ in d_a has the form (A_j, V_i), where A_j is the attribute name and V_i is the attribute value. The attributes can have multiple values (i.e., d_a may contain both (A_j, V_1) and (A_j, V_2)). We say that a document d is annotated with attribute A_j if there is any value v for which $(A_j, v) \in d_a$. We use the notation D_A and D_V for the domains of the attribute names and values, respectively[1] and $\mathcal{D}$ to denote the repository of all documents stored in the database.

Example 2: In Figure 1(a), we show the text of the document d_t and in Figure 1(b) we see a possible annotation set d_a:

$$d_a = \{(Storm\ Name, Gustav), (Storm\ Category, 3),\\ (Warnings, Tropical\ Storm)\}$$

For the document in Figure 1(a) the optimal annotations d_a^{opt} may also include $(Storm\ Speed, 16\ mph)$, $(Location, Louisiana)$, $(Max\ Wind\ Speed, 115\ mph)$.

The query workload W contains conjunctive queries of the form $Q = q_1 \wedge \cdots \wedge q_m$, where each q_i is a triplet (A_j, p, V), where A_j is an attribute value, p a predicate (e.g., $=, >, <$) and V is an attribute value. The queries in the workload express the information need of the users and we expect similar queries to be asked in the future.[2] The answer to a query Q are all the documents in $\mathcal{D}$, with annotations that satisfy the conditions of Q. For simplicity, and without loss of generality, we only consider the equality predicate in this work, although we also show some examples with more complex predicates (range condition in Figure 1(c)).

Example 3: The workload W in Figure 1(c) contains three queries Q_1, Q_2, Q_3. Q_1 has conditions $q_{11} = (Storm\ Name = Gustav)$ and $q_{12} = (Warnings, CONTAINS, flood)$. Given the annotations in Figure 1(b), Q_2 in Figure 1(c) is satisfied by conjunctive semantics. Query Q_1 is partially satisfied.

Table 1 summarizes the notation presented.

Using the above, we define our problem. A straightforward goal is to produce and display in the adaptive insertion form d_a^{opt}, given d_t; this is usually a very large

[1] Note that D_A and D_V are not known a priori.

[2] This is a common strategy for many learning algorithms that utilize query workloads, .e.g, the Google autocomplete algorithm, and the Microsoft Tuning Advisor for SQL Server.

A	Attributes used in the union of W and $\mathcal{D}$
A_j	Attribute in A
d	Document
d_t	Document text for d
d_a	Document annotations for d
$\mathcal{D}$	Repository
k	Maximum number of suggestions
$Q = q_1, q_2 \dots q_m$	Query
d_a^{opt}	complete and optimal annotations for d
W	Workload
$annotated(d, A_j)$	Document d is annotated with A_j
$use(A_j, q)$	Query q uses A_j
$\mathcal{P}$	System Prior
w	term
$score(A_j)$	Ranking function
D	Database
D_{A_j}	Database Documents annotated with A_j
$D_{A_j,w}$	Database Documents annotated with A_j that contains term w
β_i	Coefficients for Bernoulli Model

TABLE 1
Notation

set of annotations. Even if we could produce all relevant annotations, a large number of such annotations may also overwhelm the user who must examine, modify, and approve all the suggestions.[3] Hence, our efforts focus not only on identifying the potential annotations fields that exist in d_a^{opt}, but also to *rank* them and display on top the most important ones. Since the goal of annotations is to facilitate future querying, we want the annotation effort to focus on generating annotations useful for the queries in the query workload W. So, if users are willing to fill-in at most k annotations for a single document (where k is arbitrary, but fixed), our goal is to generate a subset of d_a^{opt}, while under the constraint of at-most-k- annoations. This set of annotations should be the one that increases the visibility of document d in W, that is, maximizes the number of queries that retrieve d.

Problem 2.1 (Attribute Suggestion): Given a new document d, for which we only know its text content d_t, workload W, and a limit value k, compute a set S of k attribute suggestions, such that if d_a becomes the subset of d_a^{opt} that contains attributes in S, the visibility of d in W is maximized.

The key contribution of this work is the "*attribute suggestion* " problem, which accounts for the query workload, and identifies the *attributes* that are present in the document, but *not* their values. The problem of suggesting values for the identified attributes as been widely studied before in the context of information extraction, as we discuss in 6. While we believe that query workload information can be productively used for the problem of suggesting values for the identified attributes, we consider that research problem out of the scope of the current paper and leave it as an interesting direction for future research.

3 Attributes Suggestion

In this section we study and propose solutions for the "*attributes suggestion*" problem. From the problem definition we identify two, potentially conflicting, properties for identifying and suggesting attributes for a document d:

- First, the attributes must have high *querying value* with respect to the query workload W. That is, they must appear in many queries in W, since the frequent attributes in W have a greater potential to improve the visibility of d.
- Second, the attributes must have high *content value* with respect to d_t. That is, they must be relevant to d_t. Otherwise, the user will probably dismiss the suggestions and d will not be properly annotated.

We combine both objectives, in a principled way, using a probabilistic approach. Our theoretical model is similar to the idea of language models [5], with one key difference: our model assume that attributes are generated by *two* processes, in parallel: (a) By inspecting the content of the document and extracting a set of attributes related to the content of the document, following a probability distribution given by an (unknown to us) joint probability distribution $p(d_a, d_t)$; and (b) By knowing the types of queries that users typically issue to the database, following again an (unknown to us) joint probability distribution $p(d_a, W)$.

As we will describe in this section, in this setting our goal becomes to compute a set of candidate annotation fields $\hat{d}_a$, such that the conditional probability $p(\hat{d}_a|W, d_t)$ is maximized. The value $p(d_a|W, d_t)$ measures how probable a set of annotations is for a document, given the overall query workload for the database and the text of the specific document. Adopting this probabilistic framework, we can redefine the Attributes Suggestion problem as:

Problem 3.1 (Probabilistic Atribute Suggestion): Given a query workload W and a new document d, for which we only know its content d_t, find a candidate set $\hat{d}_a$ of k attributes that maximize $p(\hat{d}_a|W, d_t)$.□

Of course, the problem is inherently intractable, if we consider all possible dependencies across attributes, document content, and workload: it is very difficult to estimate the full *joint* distribution of so many variables. Following the common practice, when estimating language models, we consider each attribute A_j independently, and we compute the k attributes that maximize $p(A_j|W, d_t)$.

Our approach for estimating the values $p(A_j|W, d_t)$ we treat, conceptually, W and d_t as forecasters (sources of evidence) and $p(A_j|W, d_t)$ is the dependent variable that we need to estimate. We leverage established work from statistics, on combining probability estimates from multiple forecasters using a Bayesian approach [6]. Given W and d as the forecasts from different sources of evidence, our system (CADS) is the decision manager, with a specific prior $\mathcal{P}$,[4] that decides how to combine the probability estimates from multiple sources. We experiment with two

[3] This is even more important in scenarios where mobile devices with small display space are used, or users have little time. We need to take into consideration this aspect when formulating our goal.

[4] The form of this prior depends on the combination strategy that we will be using.

approaches: In Section 3.1 we combine the information from the forecasters assuming conditional independence, given A_j; in Section 3.2 we build an approach that assumes conditional independence among the forecasters.

3.1 Conditional Independence given A_j and $\overline{A_j}$

We denote with $p(A_j|W, d_t, \mathcal{P})$ be the posterior probability that document d is annotated with A_j, given the forecast of W,d and a prior belief $\mathcal{P}$ of CADS about the probability of adding A_j in any document.[5] We define the score of attribute A_j as the odds that the attribute should appear in d_a. Using the Bayes theorem:

$$Score(A_j) = \frac{p(A_j|\mathcal{P}, W, d_t)}{p(\overline{A_j}|\mathcal{P}, W, d_t)} = \frac{p(\mathcal{P}, W, d_t|A_j) \cdot p(A_j)}{p(\mathcal{P}, W, d_t|\overline{A_j}) \cdot p(\overline{A_j})}$$

The numerator and denominator are equivalent to the joint distributions $p(\mathcal{P}, W, d_t, A_j)$ and $p(\mathcal{P}, W, d_t, \overline{A_j})$, respectively. Using the chain rule on both terms:

$$Score(A_j) = \frac{p(\mathcal{P}) \cdot p(A_j|\mathcal{P}) \cdot p(W|A_j, \mathcal{P}) \cdot p(d_t|A_j, \mathcal{P}, W)}{p(\mathcal{P}) \cdot p(\overline{A_j}|\mathcal{P}) \cdot p(W|\overline{A_j}, \mathcal{P}) \cdot p(d_t|\overline{A_j}, \mathcal{P}, W)}$$

If W is independent of $\mathcal{P}$, given A, and d_t is independent of $W, \mathcal{P}$, we simplify:

$$Score(A_j) = \frac{p(A_j|\mathcal{P}) \cdot p(W|A_j) \cdot p(d_t|A_j)}{p(\overline{A_j}|\mathcal{P}) \cdot p(W|\overline{A_j}) \cdot p(d_t|\overline{A_j})}$$

Our prior belief $\mathcal{P}$ is independent of $p(A_j)$, as we are not using any external knowledge to affect the estimates. So, the above equation can be further simplified to:

$$Score(A_j) = \frac{p(A_j|W)}{1 - p(A_j|W)} \cdot \frac{p(d_t|A_j)}{p(d_t|\overline{A_j})} \tag{1}$$

Equation 1 is our score function. The first term represents the likelihood of producing A_j, given the workload W. We refer to that term as *querying value* as it expresses the "relevance" of the attribute to the query workload. The second term, which we refer to as *content value* is the likelihood of observing the content d_t given that the attribute A_j appears in the document.

Estimation Process

We now present our process for estimating the values of the parameters in Equation 1.

Querying Value: Let $W_{A_j} = \{Q \in W : use(Q, A_j)\}$ be the set of queries in W that use A_j as one of the predicate conditions. We use Laplace smoothing [7] to avoid zero probabilities for the attributes that do not appear in the workload, we have:

$$p(A_j|W) = \frac{|W_{A_j}| + 1}{|W| + 1} \tag{2}$$

Content Value: For the content value $p(d_t|A_j)$, our probabilistic model assume independence between the terms in d_t, which is a typical assumption when dealing with textual data (e.g., in probabilistic information retrieval, text classification, language models, etc.) We have:

$$p(d_t|A_j) = \prod_{w \in d_t} p(w|A_j) \tag{3}$$

where the product goes over all terms w in d_t.

Let $D_{A_j} = \{d \in D : annotated(d, A_j)\}$ be the set of documents in the database D, annotated with the attribute A_j. Let $D_{A_j,w} = \{d \in D : annotated(d, A_j) \wedge contains(d_t, w)\}$ be the set of documents in the database that are annotated with A_j and also contain the word w in their text d_t. We estimate the probability of each term in Equation 3 as:

$$p(w|A_j) = \frac{|D_{A_j,w}| + 1}{|D_{A_j}| + |D| + 1} \tag{4}$$

Again we use smoothing to avoid zero probabilities. For each term, the prior is uniform and we update the probability using the observed co-occurrences of A_j and w. In a similar way we define:

$$p(w|\overline{A_j}) = \frac{|D_{\overline{A_j},w}| + 1}{|D_{\overline{A_j}}| + |D| + 1} \tag{5}$$

where we examine only documents that have been annotated and the attribute A_j was not added.

Example 4: Figure 3 shows an example of the document collection and the query workload. Documents d_1, d_2, d_3, and d_4 are already annotated on the database. A new document d_5 has just arrived and we need to find the best annotation for it. Let's consider the relative order between attributes *City* and *Supplies*. For each attribute we calculate the score as described in Equation 1. For the attribute City the querying value is calculated using Equation 2:

$$\frac{|W_{A_j}| + 1}{|W| + 1} = \frac{1 + 1}{6 + 1} = \frac{2}{7}$$

The first term in Equation 1 is:

$$\frac{p(A_j|W)}{1 - p(A_j|W)} = \frac{2/7}{1 - (2/7)} \approx 0.4$$

In a similar way we calculate the querying value for Supplies to be $3/7$ and the final contribution to be 0.75. As expected this attribute is ranked higher. To calculate the second term of 1 we need to the estimate the probabilities described in Equation 3. As we assume independence, we calculate the score for each word separately. For example the probabilities for the word ice are $p(ice|City) = 3/8, p(ice|\overline{City}) = 1/6, p(ice|Supplies) = 3/7$, $p(ice|\overline{Supplies}) = 1/7$. The content value contribution is:

$$\frac{p(d_t|A_j)}{p(d_t|\overline{A_j})} = \frac{\prod_{w \in d_t} p(w|A_j)}{\prod_{w \in d_t} p(w|\overline{A_j})} = \frac{2/8 \cdot 3/8 \cdot 2/8 \cdot 2/8}{1/6 \cdot 1/6 \cdot 1/6 \cdot 1/6} \approx 7.59$$

Similarly the content value contribution for Supplies is 6.0. Finally $score(City) = 7.59 \cdot 0.4 \approx 3.03$ and

[5] We keep the prior $\mathcal{P}$ in the conditional, separate from $p(A_j)$ mainly to align with the formal probabilistic models in [6].

Document Id	**Content**	**Annotations**
d1	Hurricane Wilma Wind forty mph	Hurricane:Wilma, Wind Speed: 40 Mph
d2	damaged traffic signals Peembroke	Reported Damage: traffic signals, City: Peembroke
d3	Water Ice Doral HS, Miami	Supplies: Water, Ice POD: Doral HS, School: Doral HS City:Miami
d4	Ice need Miami	Supplies: Ice, City: Miami
d4	need ice at Peembroke HS	?

(a) Example Collection

Query
County: Browards, POD: Peembroke Pines HS
POD: Marlins Stadium, City: Miami
POD: Doral High School, Supplies: Water
School Status: Open, School: Doral HS
County: Broward, Damage: 140 Trees
POD: Downtown POD, Supplies: Water

(b) Example Workload

Fig. 3. Running Example

$score(Supplies) = 6.0 \cdot 0.75 \approx 4.5$. So we can see that even if we have a stronger confidence on the content we use the query value to invert the order and present to the user the most promising annotation.

3.2 Conditional Independence Among Forecaster Evidence

The second model is based on the assumption that each of the two forecasters has independent information about the event "attribute A_j appears in the document," (which is different than conditioning on A_j). We capture this information as a variable with distribution p that models the occurrence of the event "attribute A_j appears in the document" as a Bernoulli experiment. Given the $\mathcal{P}$, $p_W = p(A_j|W)$, and $p_d = p(A_j|d_t)$ our final estimates is computed based on independent pieces of evidence, using the following model [6]:

$$p(A_j|W, d_t, \mathcal{P}) = \beta_0 \cdot \mathcal{P} + \beta_1 \cdot p_W + \beta_2 \cdot p_d \tag{6}$$

In this model, each forecaster provides an independent view of the annotation. To combine the evidence from multiple sources, we use the theory model from [6], which results in a weighted average combination of forecasts. We can interpret the weights as the relative expertise of each component (forecaster/source). If the prior information that we have for the attributes is non-informative, then $\beta_0 = 0$ and we have:

$$p(A_j|W, d_t, \mathcal{P}) = \beta_1 \cdot p_W + \beta_2 \cdot p_d \tag{7}$$

where p_W and p_d are the querying value and content value components.

Estimation Process

Now, we present the estimation process for the probabilities in Equation 7.

Querying Value: Let $W_{A_j} = \{Q \in W : use(Q, A_j)\}$ be the set of queries in W that specify A_j. As in Section 3.1, we have:

$$p_W = p(A_j|W) = \frac{|W_{A_j}| + 1}{|W| + 1} \tag{8}$$

Content Value: For the content value we use effectively the same approach as a Naive Bayesian Classifier:

$$p_d = p(A_j|d_t) \propto p(A_j) \cdot \prod_{w \in d_t} p(w|A_j) \tag{9}$$

where again we assume independence among the terms. We estimate $p(A_j)$ as the smoothed frequency of A_j in the database:

$$p(A_j) = \frac{|D_{A_j}| + 1}{|D| + 1} \tag{10}$$

Example 4: **Continued:**

We continue our example calculating the score using the Bernoulli Model. First we find the p_W value for Equation 7. As the querying value is the same we estimated before we have that the $p_w(City) = 2/7$ and $p_w(Supplies) = 3/7$. To calculate the content value of 7:

$$p_d(A_j) = p(A_j|d_t) \propto p(A_j) \cdot \prod_{w \in d_t} p(w|A_j)$$

That gives $p_d(City) = 3/4 \cdot (2/8 \cdot 3/8 \cdot 2/8 \cdot 2/8) \approx 0.004$ and $p_d(Supplies) = 2/4 \cdot (2/7 \cdot 3/7 \cdot 1/7 \cdot 2/7) \approx 0.002$.

To use Equation 7, we first need to normalize p_W and p_d to sum up to 1 accross all candidate attributes. To simplify the example, consider only our two attributes. Then we have $p_d(City) = 0.4$ and $p_d(Supplies) = 0.6$. For the workload contribution we have $p_W(City) = 0.33$ and $p_W(Supplies) = 0.66$

Finally $score(City) = \beta \cdot 0.4 + (1 - \beta) \cdot 0.66$ and $score(Supplies) = \beta \cdot 0.6 + (1 - \beta) \cdot 0.33$. The value of β can be used to change the order. In particular $score(Supplies) > score(City)$ if $\beta > 0.6$.

Weight Coefficients Estimation

In the Bernoulli model of Equation 7, the probability estimates provided by both sources of evidence (document content and query workload) are independent, given the (latent) A_j. A key challenge is to assign values to coefficients β_1 and β_2 in Equation 7. For that, we adopt an incremental learning technique: We use as training data the queries and documents that have been annotated so far (i.e., for which the d_a^{opt} is given) and we pick the coefficient values that maximize the likelihood that the annotation will improve the querying and content value. The process for estimating works as follows. Let $d_{a,qv}$, $d_{a,iv}$ be the top-k suggestions computed for a document d using just the *Querying Value* score or just the *Content Value* score, respectively. Using the set of optimal annotations d_a^{opt} we compute the intersection s$d_{a,qv} \cap d_a^{opt}$, $d_{a,iv} \cap d_a^{opt}$, which is the set of correctly suggested annotations. Then, for each document, we count how many queries are satisfied using $d_{a,qv} \cap d_a^{opt}$ or $d_{a,iv} \cap d_a^{opt}$. Let z_{qv}, z_{iv} be the sums of these two numbers across all documents respectively; we define $\beta_1/\beta_2 = z_{iv}/z_{qv}$, which is the ratio of the (correct) contribution of the two components towards the maximization of coverage of the document in the query workload.[6]

One problem with the presented equations for calculating *Querying Value (QV)* and *Content Value (CV)* is that QV and CV may have different ranges, and hence the overall score may be dominated by either QV or CV in Equations 1 and 7. We normalize CV and QV in the range between 0 and 1 by calculating their maximum value across all attributes, and dividing the rest with this number. In this way we guarantee that *QV* and *CV* have the same impact.

4 Efficiency Issues and Solutions

In this section, we discuss the algorithmic approaches that allow us to implement efficiently the algorithms described in the previous section. In particular, we show how pipelined algorithms can be employed to compute the top-k attributes with the highest scores, where scores are defined using Equation 1 (*Bayes* strategy) or Equation 7 (*Bernoulli* strategy).

In both strategies, we need to find efficient ways to calculate the *Querying Value* (QV) and *Content Value* (CV) components, which are defined in similar ways for the two strategies. We observe that in both strategies the score is a monotonically increasing function ($f(QV, CV) = CV{\cdot}QV$ for *Bayes* and $f(QV, CV) = \beta_1 \cdot QV + \beta_2 CV$ for *Bernoulli*).

QV computation

A key observation is that the *QV* of an attribute is independent of the submitted document, as seen in Equation 2; *QV* only depends on the query workload. Hence, we maintain a precomputed list L^{QV} of *QVs* of the attributes in D_A, ordered by decreasing *QV* values. Since the query workload does not change significantly in real-time, we update L^{QV} only periodically, as new queries arrive, since is not critical for the *QV* metrics to be absolutely up-to-date: approximations suffice.

CV computation

In contrast, it is expensive in terms of time and space to maintain all the *CVs* for all pairs of documents and attributes, where *CV* is defined in Equation 3. For that, we compute the *CVs* at run-time when a document arrives. The goal is to minimize the number of such computations when computing the top-k attribute suggestions. Given a document d_t, we compute *CV* as follows. We first parse d_t. For each term $w \in d_t$ we compute its contribution using Equation 5. For that, we exploit two indexes: the inverted index I_t indexes the text of all documents, and the inverted index I_a stores for each attribute name A_j the list of documents for which $A_j \in d_a$. To compute the numerator $D_{A_j,w}$ of Equation 5 we intersect the lists for A_j from the two indexes I_t and I_a. The denominator D_{A_j} is computed directly using I_a. We refer to this algorithm as $GetCV(A_j)$.

Combining QV and CV

We employ a variation of the Threshold Algorithm with Restricted Sorted Access (TA_Z), described in [9]. The pipelining algorithm performs sequential access on L^{QV} and for each seen attribute A_j it performs a "random access" to compute *CV* by executing $GetCV(A_j)$.

The algorithm executes as follows:

1) Retrieve next A_j from L^{QV}.
2) Get the Content Value (*CV*) for attribute A_j.
3) Calculate the Threshold value $\tau = F(\overline{CV}, QV(A_j))$ where $\overline{CV}$ is the maximum possible *CV* for the unseen attributes and $QV(A_j)$ is the QV of A_j.
4) Let R be the set of k attributes with highest score that we have seen. Add A_j to R if possible.
5) If the k-th attribute A_k has $Score(A_k) > \tau$ we return R. Else we go back to Step 1.

Note that instead of using TA_Z to combine *CV* and *QV*, we could have used the MPro algorithm [10], where the key difference is that sequential accesses has cost 0, and the execution is scheduled such that the number of random accesses are minimized. For simplicity, and since the efficiency of such computations is not the core contribution of this paper, we only present the results that we observed using the TA_Z algorithm.

5 Experiments

5.1 Datasets

Documents: For our experiments we use two document collections:

- The *Emergency* corpus consists of 270 documents, generated by the Miami-Dade Emergency Management Office. The documents are advisory, progress and situation reports submitted by various county stakeholders during the five days before and after Hurricane Wilma, which hit Miami-Dade county in October 2005.
- The *CNET* corpus consists of 4,840 electronic product reviews obtained from CNET[7]. The dataset contains different kinds of products like cameras, video games, television, audio sets, and alarm clocks.

[6] An alternative approach, that we leave for future work, is to treat each the two sources of evidence as "noisy labelers" that identify an attribute as present or not, and then estimate the quality of the sources using a framework similar to the one used to evaluated crowd-sourcing workers

	Emergency	CNET	Amazon
Number of Documents	270	4840	19700
Maximum Size (KB)	35.8	90.4	4.0
Average Size (KB)	2.87	10.56	1.06
Minimum Size (KB)	0.37	0.16	0.002
Annotations per Document (Max)	24	53	43
Annotations per Document (Min)	1	1	1
Annotations per Document (Avg)	7.9	9.61	4.82

TABLE 2
Corpus Statistics

Emergency	CNET	Amazon
Date	Diagonal Size	Memorabilia
County	Color Support	Autographed
Phone	Included Accessories	Batteries Included
Person	Data Link Protocol	Model
Hours	Technology	EAN
Report Number	Enclosure Color	NumberOfItems
Storm	Exposure Modes	Display Size
Address	Device Type	MPN
Agency	Color	Optical Size
Web Page	Supported Battery	Lens Type

TABLE 3
Attributes with the highest frequency in each data set

- The *Amazon Products* corpus are 19700 documents downloaded from Amazon [8]. This dataset also included electronic products, books and other items that are sell at Amazon.

Statistics for each collection are presented in Figure 2.

Annotations: We generated annotations for the datasets, which we use as training and test data, to train and evaluate our algorithms.

For the *Emergency* corpus we acquired the annotations in two phases. In the first phase, a group of five students read a random sample of 40 documents and identified a set of 76 attributes that best describe the domain (see Table 3 for a sample of these attributes). We mapped attributes with similar meaning (e.g. time, hour) to a single attribute. In the second phase, we used Amazon Mechanical Turk[9] to have Mechanical Turk users annotate the documents, using the attributes identified in the first phase. For each document, we assigned five users ("workers"), who each submitted at least five annotations for each documents. Workers selected the attribute names from a-drop down list and specified the values using text boxes. Users were limited to annotating at most ten documents, to avoid excessive influence of any single participant in the overall extraction process. To avoid invalid annotations, we sampled five annotations from of each annotator and validated that the identified attributes and values were correct. We check manually that the annotations are relevant (not noise) and are related to the document content. Users with invalid annotations were disqualified and their annotations discarded. For each document, we use as ground truth the annotations given by at least two (of the five) qualified workers that examined and annotated the document.

To annotate the *CNET* reviews we used the CNET specifications page for each product. The page contains structured data for a product in the form of "*attribute name, value*". Given that we are only interested in annotations that come from the document text (i.e. the product's review), we removed annotations that are not mentioned in any sentence in the review text. To decide when a sentence s is related (mentions) to an annotation $\mathcal{A} = (A_j, V_i)$, we used the containment ratio heuristic; specifically, we computed:

$$cr(\mathcal{A}, s) = \delta \frac{|A_j \cap s|}{|A_j|} + (1-\delta)\frac{|V_i \cap s|}{|V_i|} \qquad (11)$$

where A_j, V_i, s are the set of words for the attribute name, value and the sentence, respectively; δ conveys the importance of matching the name and value. To set the δ parameter we consider some special cases: For Boolean attributes (yes or no values) we focus only on the attribute name ($\delta = 1$). For values that only appear in one attribute, we assign a higher weight to the value ($\delta = 0.8$). In all other cases, we allocate the same weight ($\delta = 0.5$). We performed a manual check on a sample of 50 documents to confirm the precision of this heuristic: we found that in all cases the heuristic worked for our purpose.

For the Amazon dataset we divide the page into two parts: the textual part formed with the product description and the list of features, and the annotations formed with the structured attribute/value section on the web-page. We consider the same strategy as used on the CNET corpus to find those annotations that appears on the text.

Table 3 shows the top 10 most frequent attributes names and their distribution for both corpora.

Queries: When generating the query workload for our datasets we had to address two main challenges. First, we did not have a query workload that was used to query the data sets in our disposal. So, we had to generate a workload, with an attribute distribution representing the user interests in a realistic way. Second, we had to create queries of the form attribute-value as described in Section 2.

To obtain a realistic query distribution we leveraged the Google Trends and Google Insights tools, which list statistics about the popularity and time variations of queries issued against the Google search engine. Furthermore, these tools allow us to focus on a specific geographical area to extract such statistics, and we can also compare two keywords and see the relative lift for the query based on a particular time and location. In our first pass, we picked the queries that presented a significant increase in usage during the time-frame of the dataset, and for the specified location. Then, used the relative frequencies of the queries in the Google Insights/Trends to weight appropriately the workload in the results. For example, for the *emergency data set* we could see more queries related with the status of the schools on the city compared to queries asking for the status of the ports. Given that the former were submitted five times more often, then we generate five queries related to school information, and only one asking for a the ports.

The second problem is how to transform keyword queries to structured queries. For this, we examined the queries that were returned by Google, to examine whether they mention or imply an attribute value. For example, these are two query transformations: “open schools” → “School Status: open,”; the query “kodak 1120 color blue” → “model: kodak 1120, camera color: blue”. Effectively, we injected attributes in the otherwise flat keyword workload. We accepted a transformation as correct only when entered independently by multiple independent users on Amazon Mechanical Turk.

For our *emergency* data set, since the documents are from the Wilma Hurricane in 2005 in Florida, we specify Florida as location and 2005 as year in Google Trends. To avoid zero frequencies, we smooth these Florida-specific frequencies with the US and World frequencies for the same period. In particular:

$$f(w) = 0.5 \cdot f_{FL}(w) + 0.25 \cdot f_{US}(w) + 0.25 \cdot f_{W}(w) \quad (12)$$

where f_{FL}, f_{US}, f_{FL} are the Florida, US and World frequencies respectively. We use $f(w) = \epsilon$ as a default probability for those with zero frequencies.

To generate queries for the *Product* and *Amazon* database we use a two phase process. First, we check the attributes popularity using Google Insights restricted to the Technology domain. For each attribute we submit a single query with the attribute name to the service, and obtain the relative returned popularity. Then we use this value to create vector with one entry per attribute name and some probability. Attributes with zero popularity in the Google Insights are added to the vector with some minimal probability ϵ to avoid zero entries. In the second phase we generate 10,000 queries using the following procedure:

1) Select the length of the query l by sampling from a uniform probability distribution with lengths varying from 1 to 3.
2) Select an attribute A_1 using the popularity that they have on the vector we obtained from Google Insights.
3) Select the next attribute A_2 using the co-occurrence ratio with the previous attribute A_1.
4) Repeat from Step 2, until we get l different attributes

Note that when generating the queries in *Emergency* we do not consider their pairwise correlations because the correlations across the emergency queries were significantly lower. In contrast, for CNET, we observed significant dependencies across attribute pairs. Using the co-occurrence in step 3 we favor attributes from the same product type (cameras, notebooks, air conditioners), as opposed to independently combining attributes across such product types.

Table 4 shows the top 10 most frequent attributes for the workload and their distribution for both corpora.

5.2 Experimental Setup

To evaluate the algorithmic approaches that we introduce in this paper, we compare our algorithms with a variety of existing baselines.

- DataFreq: Suggest the most frequent attributes in the database of annotated documents.

Emergency	*CNET* Amazon	
School Status (3738)	Software (1793)	studio(1178)
County (2202)	Audio (1520)	model(1141)
Weather (2087)	Power (876)	type(816)
Address (1695)	Speed (628)	publisher(812)
Supplies (1640)	Additional Features (600)	size(767)
Storm (1500)	Control (549)	label(745)
Phone (1254)	Modem (536)	CPU manufacturer(747)
Location (1106)	Connection (448)	brand(668)
Water Status (954)	Color (422)	creator(666)
Library Status (859)	Service Support Details(413)	binding (595)

TABLE 4
Attributes with the highest frequency in the query workload

- **QV**: Suggest attributes based on the querying value component of Section 3, which is similar to ranking attributes based on their popularity in the workload.
- **CV**: Suggest attributes based on the content value component of Section 3.
- **Calais**: We use the Open Calais[10] information extraction system, as a black box. Calais can recognize persons, locations, dates and other entities that are common in news articles. The entities extracted are fixed to a particular schema that we map to our own attributes. We annotate the documents and consider all the attributes that correspond to an entity. We use the Calais relevance score to rank the attributes. If the same attribute is annotated with multiple values we use the highest relevance score value to score it. Products have specialized attributes and hence we can not use this generic extractor as a baseline, so we only use this strategy as a baseline for the Emergency data set.
- **RAKEL** We use *RAKEL* [11] a state-of-the-art multilabeler that take into account the correlation between tags for annotations. We use the implementation provided in Mulan[11] using the default parameters provided in the tool, i.e, a LabelPowerset transformation and the J48 algorithm.
- **Bayes**: combine *QV* and *CV* as presented in Section 3.1.
- **Bernoulli**: combine *QV* and *CV* as presented in Section 3.2 with a specific β_1. If we do not specify β_1 we are referring to the β_1 estimation strategy described in Section 3.2.

5.3 Precision and Recall of Suggested Attributes

In this experiment, we measure the quality of the suggested attributes for a document, compared to its ground-truth attributes. Note that this experiment ignores the query workload, and hence does not measure the success of the strategies in solving the Attribute Suggestion problem, which is the key contribution of this paper, and is evaluated in Section 5.4. Nevertheless, the purpose of this experiment is to show that a strategy does not suggest attribute that are irrelevant to the content of a document.

For each execution, we pick a document d for evaluation (testing) and use the rest as training set, that is, as the annotated documents database. We calculate the precision for the test document d as the ratio of the suggested attributes d_a that are in the ground truth attributes d_a^{opt} of d. We use the full workload to estimate the querying value. We report the precision and recall averaged over all documents d in $\mathcal{D}$.

Figures 4 show the average precision and recall for varying number of top-k suggestions. As expected, the most precise strategy is *CV*, since it focuses on using the document text to suggest attributes. *DataFreq* is very close to *CV*, because as we see in Table 3 the frequency of the highest attribute is very common suggesting the most frequent attributes is quite effective. Another reason is that there are very few training documents for the infrequent attributes for *CV* to be effective. We also observe that our proposed strategies *Bernoulli* and *Bayes* have also high precision, even though they are designed to also consider the query workload. A key difference between the two datasets is that in CNET and Amazon *DataFreq* performs much worse due to the richness of the schema. The precision decreases with the number of suggestions because each document only has few ground-truth attributes. This is typical behavior where the precision decreases and the recall increases as we show more results (suggestions).

Calais shows why a fixed schema extractor can not adapt to the kind of domains we are addressing. As we can see generic extractors for multiple domains focus on generic attributes like person, location, dates, organization that are very useful on more general domains but not for domain specific task. *RAKEL* on the other hand can learn with the schema in the same way as the proposed techniques. As we can see is as competitive as the other techniques. On the other hand is still worse than the Naive Bayes classifier. This strange behavior can come from two reasons: one our tagging is noisy, users are missing tags or annotate with incorrect annotations. On the other hand attributes on this domain are poorly correlated so searching for correlations can be misguiding. Naive Bayes seems to be less sensitive to the noise. KaREL also is exponential on the number of attributes combinations that it test. Even testing for pairwise correlations we can not make the system finish for the Amazon Dataset. We report only for the CNET and Emergency datasets.

5.4 Attributes Suggestion Problem

In this experiment, we examine how the different strategies solve the Attributes Suggestion Problem, which is the core focus of our work. That is, if a strategy is used for attributes suggestion, how well are the queries of the workload answered? To measure this we use the sum of documents returned by the queries in the workload, where a document is counted multiple times, once for every query that returns it. We refer to this measure as *Full Match*. We also consider a simpler variant, *Partial Match*, where we count how many query conditions are satisfied by the documents, that is, we view each query condition as a separate query.

We first introduce the optimal suggestion techniques, which will be used as baselines to evaluate the strategies.

- **OPTFullMatch** suggests the subset of the ground-truth attributes for each document that maximize its query visibility in the query workload, that is, that satisfies the maximum number of queries. Miah et al. [12] prove that this problem is NP-Hard. However, given the relatively small size of our query workload, we were able to compute an exact solution using the exact algorithm from [12], following a brute-force approach, which took a significant amount of time but allowed us to measure exactly how close to the optimal each algorithm is.
- **OPTPartialMatch** suggests a subset of the ground-truth attributes that maximize the number of query conditions satisfied. This can be computed making a single pass on the workload

We report the average coverage across all documents, where the coverage for one document is defined as the number of matches (or partial matches) divided by the number of matches of *OPTFullMatch* (or *OPTPartialMatch*, respectively). Figures 5 and 6 show the average coverage for full and partial matches for the strategies. The proposed strategies *Bayes* and *Bernoulli* dominate the rest strategies by up to 50%, especially for fewer numbers of suggestions, which are the most practical cases. We observe that the baselines *Calais* and *RAKEL* are not competitive for the reasons discussed above: the former is highly imprecise is does not capture many attributes in the document and queries, and the latter ignores the workload information.

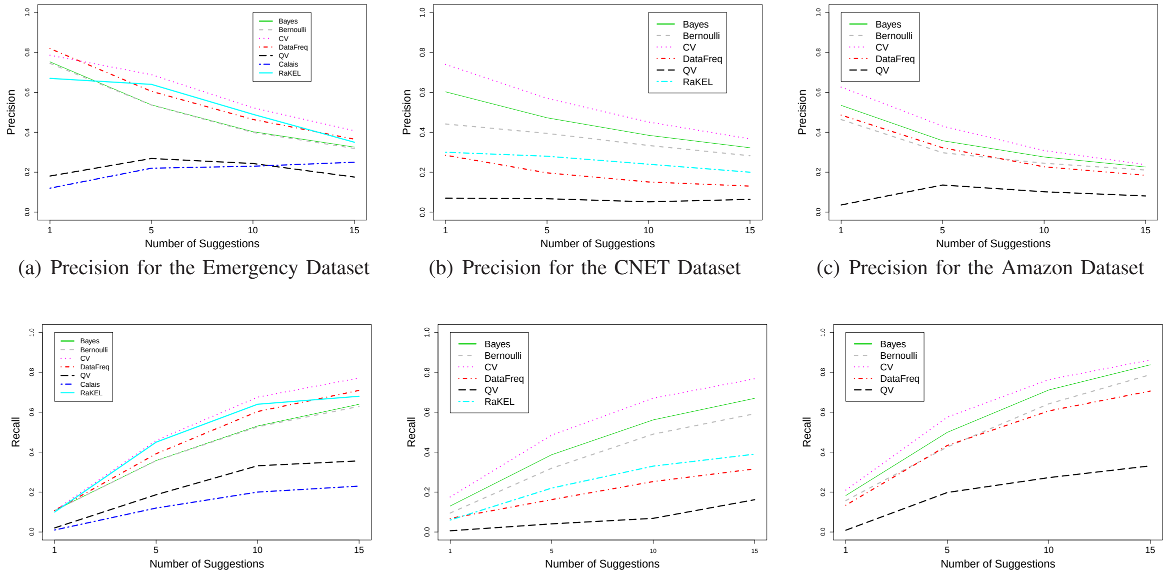
Interestingly, the *QV* strategy performs well, even though it ignores the text of the documents. The reason is that the frequency of the attributes in the workload decreases very quickly, so covering the top attributes is a successful strategy. Nevertheless, as we discussed in Section 5.3 the precision for this strategy is too low, so much of the user effort will be wasted on removing spurious suggestions.

We also note that *QV*'s rate of improvement (in number of matches) drops considerably after 10 suggestions, compared to *DataFreq*. The reason is that in the query workload, the attributes after the top-10 (in terms of frequency) cover few documents.

Note that for the CNET and Amazon dataset it was infeasible to calculate the optimal results, so we report the raw (non-normalized) number of matches instead of the normalized version.

Discussion: The results confirm our intuition that the best ways to select a number of candidate attributes to annotate a document need to balance both CV and QV, as is the case for *Bayes* and *Bernoulli*. The reason is that high CV is needed to avoid suggesting attributes that are irrelevant to the document (which the user annotator will have to manually discard – no second round of suggestions is allowed in our model if some attributes are judged as irrelevant), and high QV is needed to prefer attributes that are popular in the query workload.

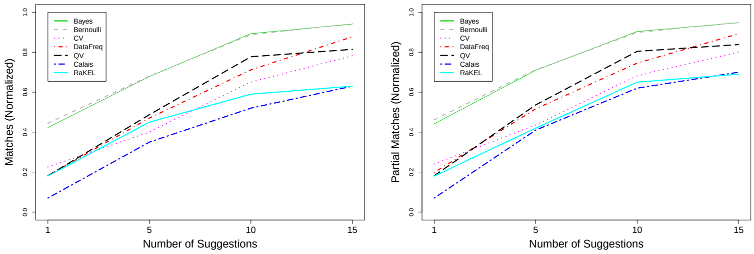
For both datasets the CV performs better than the baseline. The reason is that the elements of the schema that are used to annotate one product depends on the particular product type (e.g attributes for camera). As the CV behaves



(a) Precision for the Emergency Dataset (b) Precision for the CNET Dataset (c) Precision for the Amazon Dataset

(d) Recall for the Emergency Dataset (e) Recall for the CNET Dataset (f) Recall for the Amazon Dataset

Fig. 4. Attributes Precision/Recall for All Datasets



(a) Full Matches (b) Partial Matches

Fig. 5. Number of Full and Partial Query Matches in Emergency DataSet.

like an object classifier, it picks the right attributes for the object, even when the are not the most frequent in the database or the workload ($DataFreq$ and QV).

Effect of the β_1 parameter: In the *Bernoulli* strategy we can manipulate the weights of the components, *CV* and *QV*. In this experiment we check the effect of the parameter β_1 (see Equation 7) for full and partial matches. We set the number of suggestions to $k = 5$.

Figure 8 shows the changes on number of matches and partial matches for both datasets. The vertical line shows the β_1 computed using the method proposed in Section 3.2 for both databases. For the Emergency dataset this value is around 0.51. For the CNET dataset the value is around 0.31. We see that the number of matches increases as we add more weight to the workload and decrease only slightly at the end. As we already discussed the most common attributes are highly common so a simple strategy is competitive as we see in Figure 8(a). For the CNET dataset we see in Figure 8(b) that the most competitive strategy is the CV. So, adding too much weight to the workload only makes the results worse.

5.5 Database Size Effect

In this experiment we check the effect of the size of the database on the precision of attribute suggestions and the number of query matches. Recall that the content value is computed using the database of annotated documents, as shown in Equation 5. We consider subsets of the database of documents of different sizes. As the number of documents of the *Emergency* dataset is very small ,we only report the results obtained in the *CNET* and *Amazon* dataset.

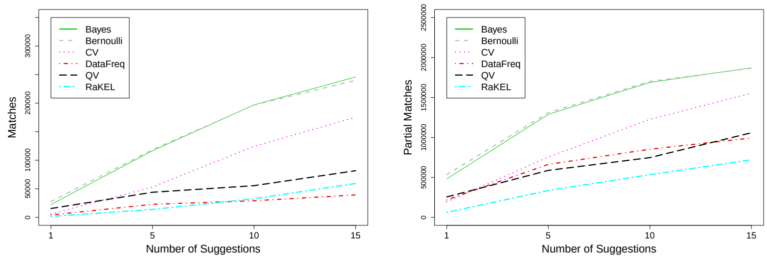
Figure 9 shows the variations in precision of suggested attributes and in the normalized number of partial query matches when we use $1/16, 1/8, 1/4$ and $1/2$ of the database for the estimation and the rest of the database as test set, and report on the average values.

In particular, Figures 9(a) and 9(c) show the precision changes when we increase the size of the training set. As expected the proposed strategies increase their quality when we increase the training data size. The QV line is constant since it does not use the database but only the query workload.

Figures 9(b) and 9(d) show the number of partial query matches for each strategy as the training size increases. The proposed query agnostic strategies improve again as we increase the size of the training set.

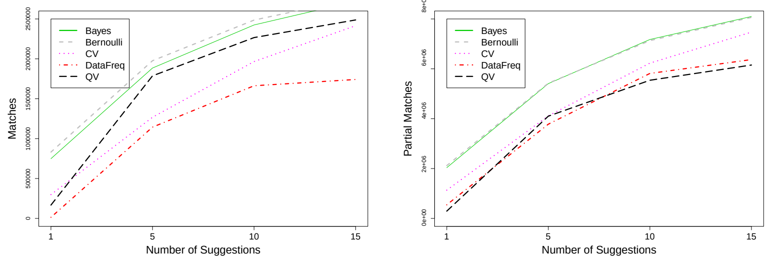
6 Related Work

Collaborative Annotation: There are several system that favor the collaborative annotation of objects and use previous annotations or tags to annotate new objects. There



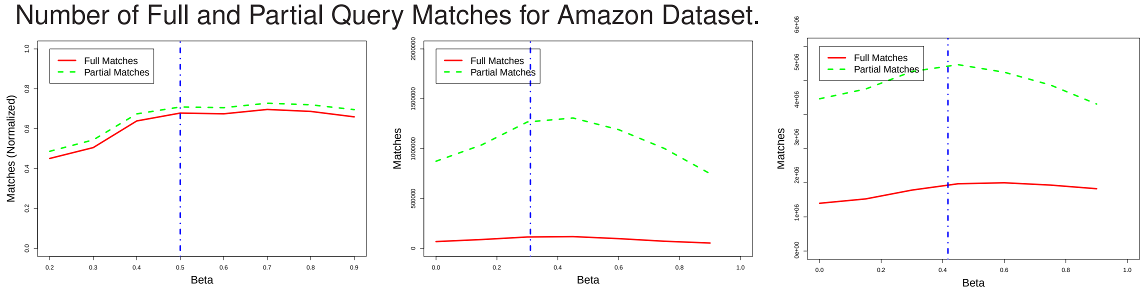
(a) Full Matches (Non-Normalized) (b) Partial Matches (Non-Normalized)

Fig. 6. Number of Full and Partial Query Matches for CNET Dataset.



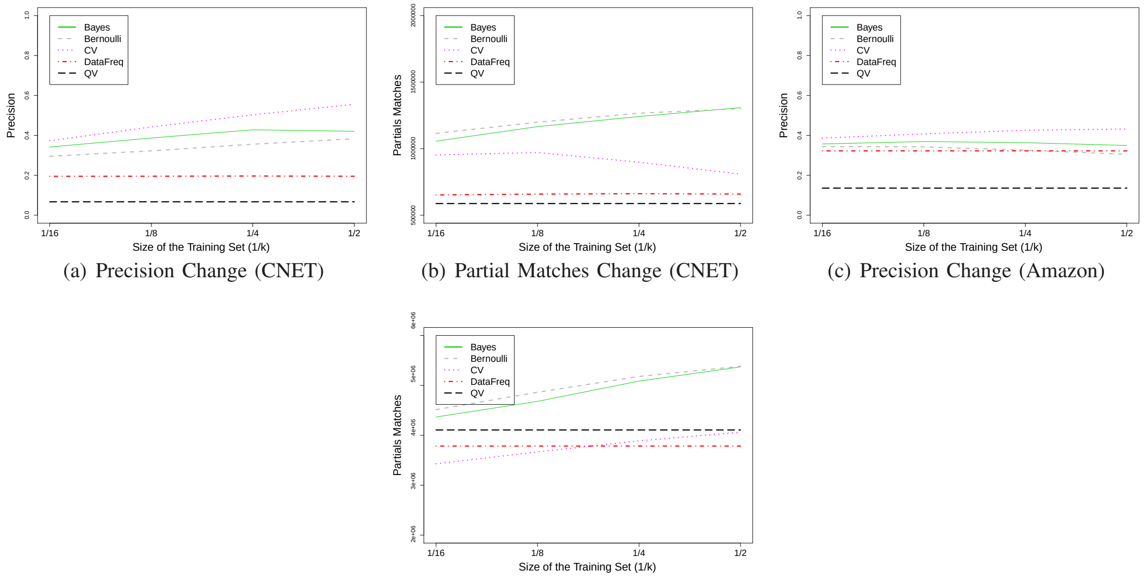
(a) Full Matches (Non-Normalized) (b) Partial Matches (Non-Normalized)

Fig. 7. Number of Full and Partial Query Matches for Amazon Dataset.



(a) Emergency database (b) CNET database (c) Amazon database

Fig. 8. Effect of β_1 in the Bernoulli Model.



(a) Precision Change (CNET) (b) Partial Matches Change (CNET) (c) Precision Change (Amazon)

(d) Partial Matches Change (Amazon)

Fig. 9. Effect of Training Set Size in CNET/Amazon Dataset.

have been a significant amount of work in predicting the tags for documents or other resources (webpages, images, videos) [13], [14], [15], [16], [17]. Depending on the object and the user involvement, this approaches have different assumptions on what is expected as an input, Neverthless the goals are similar as the expect to find missing tags that are related with the object. We argue that our approach is different as we use the workload to augment the document visibility after the tagging process. Compared with the other approaches precision is a secondary goal as we expect that the annotator can improve the annotations on the process. On the other hand, the discovered tags assist on the taks of retrieval instead of simply bookmarking.

Dataspaces and pay-as-you-go integration: The integration model of CADS is similar to that of dataspaces [18], where a loosely integration model is proposed for heterogeneous sources. The basic difference is that dataspaces *integrate existing annotations* for data sources, in order to answer queries. Our work *suggests the appropriate annotation* during insertion time, and also takes into consideration the query workload to identify the most promising attributes to add. Another related data model is that of Google Base [1], where users can specify their own attribute/value pairs, in addition to the ones proposed by the system. However, the proposed attributes in Google Base are hard-coded for each item category (e.g., real estate property). In CADS, the goal is to *learn* what attributes to suggest. Pay-as-you go integration techniques like PayGo [19] and [2] are useful to suggest candidate matchings at query time. However, no previous work considers this problem at insertion time, as in CADS. The work on Peer Data Management Systems [20] is a precursor of the above projects.

Content management products: Microsoft Sharepoint [21] and SAP NetWeaver [22] allow users to share documents, annotate them and perform simple keyword queries. Hard-coded attributes can be added to specialized insertion forms. CADS improves these platforms by learning the user information demand and adjusting the insertion forms accordingly.

Information extraction (IE): Information extraction is related to this effort, mainly in the context of value suggestion for the computed attributes. (See [23] for an overview of IE.) We can broadly separate the area into two main efforts: Closed IE and Open IE. Closed IE requires the user to define the schema, and then the system populates the tables with relations extracted from the text. Our work on attribute suggestion naturally complements closed IE, as we identify what attributes are likely to appear within a document. Once we have that information, we can then employ the IE system to extract the values for the attributes. Open IE [24] is closer to the needs of CADS. In particular, Open IE generates RDF-like triplets, e.g., (Gustav, is category, 3) with no input from the user. Open IE leads to a very large number of triplets, which means that even after the successful extraction of the attribute values, we still have to deal with the problem of schema explosion that prevents the successful execution of structured queries that require knowledge of the attribute names and values that appear within a document. In principle, we could use Open IE, and then pay-as-you-go solutions for identifying equivalency relations across attribute names: however, it is much better to deal with the problem early-on, during document generation, instead of trying to fix issues that could be prevented with proper design. The CIMPLE project [25], [26] uses IE techniques to create and manage data-rich online communities, like the DBLife community. In contrast to CIMPLE, where data is extracted from existing sources and a domain expert must create a domain schema, CADS is a data sharing environment where users explicitly insert the data and the schema automatically evolves with time. Nevertheless, the IE and mass collaboration techniques of CIMPLE can help in creating adaptive insertion forms in CADS.

Schema Evolution: Note that the adaptive annotation in CADS can be viewed as semi-automatic schema evolution. Previous work on schema evolution [27] did not address the problem of what attribute to add to the schema, but how to support querying and other database operations when the schema changes.

Query Forms: Existing work on query forms can be leveraged in creating the CADS adaptive query forms. Jayapandian and Jagadish [28] propose an algorithm to extract a query form that represents most of the queries in the database using the ”querability” of the columns, while in [29] they extend their work discussing forms customization. Nardi and Jagadish [30] use the schema information to auto-complete attribute or value names in query forms. In [26] keyword queries are used to select the most appropriate query forms. Our work can be considered a dual approach: instead of generating query forms using the database contents, we create the schema and contents of the database by considering the content of the query workload (and the contents of the documents, of course). The work in USHER [31] is also related: in USHER the system automatically decides which questions in a survey are the most important to ask, given past experience with the completion of past surveys. In a sense, USHER is complementary to CADS: once we identify the attributes and values in the documents using CADS, we can then use USHER to model the dependencies across attributes and minimize the number of questions asked.

Probabilistic Models: Probabilistics tag recommendation systems [32], [33] have a similar goal like our system. However, the main difference is that we use the query workload in our model, reflecting the user interest.

7 Conclusion

We proposed adaptive techniques to suggest relevant attributes to annotate a document, while trying to satisfy the user querying needs. Our solution is based on a probabilistic framework that considers the evidence in the document content and the query workload. We present two ways to combine these two pieces of evidence, content value and querying value: a model that considers both components conditionally independent and a linear weighted model. Experiments shows that using our techniques, we can suggest attributes that improve the visibility of the documents with respect to the query workload by up to 50%. That is, we show that using the query workload can greatly improve the annotation process and increase the utility of shared data.

Acknowledgments

Vagelis Hristidis was partially supported by NSF grants IIS-1216032 and IIS-1216007. Panagiotis G. Ipeirotis was partially supported by NSF grant IIS-0643846, and a George A. Kellner Faculty Fellowship. Both were also supported by a Google Award.

References

[1] Google, "Google base, http://www.google.com/base," 2011.

[2] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in *ACM SIGMOD*, 2008.

[3] K. Saleem, S. Luis, Y. Deng, S.-C. Chen, V. Hristidis, and T. Li, "Towards a business continuity information network for rapid disaster recovery," in *International Conference on Digital Government Research*, ser. dg.o '08, 2008.

[4] A. Jain and P. G. Ipeirotis, "A quality-aware optimizer for information extraction," *ACM Transactions on Database Systems*, 2009.

[5] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '98. New York, NY, USA: ACM, 1998, pp. 275–281. [Online]. Available: http://doi.acm.org/10.1145/290941.291008

[6] R. T. Clemen and R. L. Winkler, "Unanimity and compromise among probability forecasters," *Manage. Sci.*, vol. 36, pp. 767–779, July 1990. [Online]. Available: http://portal.acm.org/citation.cfm?id=81610.81609

[7] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, July 2008. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0521865719

[8] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ser. HCOMP '10. New York, NY, USA: ACM, 2010, pp. 64–67. [Online]. Available: http://doi.acm.org/10.1145/1837885.1837906

[9] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, pp. 614–656, June 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=861182.861185

[10] K. C.-C. Chang and S.-w. Hwang, "Minimal probing: supporting expensive predicates for top-k queries," in *ACM SIGMOD*, 2002.

[11] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *Proceedings of the 18th European conference on Machine Learning*, ser. ECML '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 406–417. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74958-5_38

[12] M. Miah, G. Das, V. Hristidis, and H. Mannila, "Standing out in a crowd: Selecting attributes for maximum visibility," *ICDE*, 2008.

[13] P. Heymann, D. Ramage, and H. Garcia-Molina, "Social tag prediction," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 531–538. [Online]. Available: http://doi.acm.org/10.1145/1390334.1390425

[14] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles, "Real-time automatic tag recommendation," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 515–522. [Online]. Available: http://doi.acm.org/10.1145/1390334.1390423

[15] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green, "Automatic generation of social tags for music recommendation," in *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press, 2008.

[16] B. Sigurbjörnsson and R. van Zwol, "Flickr tag recommendation based on collective knowledge," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 327–336. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367542

[17] B. Russell, A. Torralba, K. Murphy, and W. Freeman, "Labelme: A database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, pp. 157–173, 2008, 10.1007/s11263-007-0090-8. [Online]. Available: http://dx.doi.org/10.1007/s11263-007-0090-8

[18] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Rec.*, vol. 34, pp. 27–33, December 2005. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107502

[19] J. Madhavan and et al., "Web-scale data integration: You can only afford to pay as you go," in *CIDR*, 2007.

[20] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov, "Schema mediation in peer data management systems," in *Data Engineering, 2003. Proceedings. 19th International Conference on*, march 2003, pp. 505 – 516.

[21] M. Sharepoint, "http://www.microsoft.com/sharepoint/," 2011.

[22] S. N. C.-C. Management, "https://www.sdn.sap.com/irj/sdn/nw-cm," 2011.

[23] M. J. Cafarella, J. Madhavan, and A. Halevy, "Web-scale extraction of structured data," *SIGMOD Rec.*, vol. 37, pp. 55–61, March 2009. [Online]. Available: http://doi.acm.org/10.1145/1519103.1519112

[24] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld, "Open information extraction from the web," *Commun. ACM*, vol. 51, pp. 68–74, December 2008. [Online]. Available: http://doi.acm.org/10.1145/1409360.1409378

[25] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen, "Community information management," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 64–72, 2006.

[26] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton, "Combining keyword search and forms for ad hoc querying of databases," in *SIGMOD*, 2009.

[27] J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth, "Semantics and implementation of schema evolution in object-oriented databases," in *ACM SIGMOD*, 1987.

[28] M. Jayapandian and H. V. Jagadish, "Automated creation of a forms-based database query interface," *Proc. VLDB Endow.*, vol. 1, pp. 695–709, August 2008. [Online]. Available: http://dx.doi.org/10.1145/1453856.1453932

[29] M. Jayapandian and H. Jagadish, "Expressive query specification through form customization," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, ser. EDBT '08. New York, NY, USA: ACM, 2008, pp. 416–427. [Online]. Available: http://doi.acm.org/10.1145/1353343.1353395

[30] A. Nandi and H. V. Jagadish, "Assisted querying using instant-response interfaces," in *ACM SIGMOD*, 2007.

[31] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh, "Usher: Improving data quality with dynamic forms," in *ICDE*, 2010.

[32] D. Liu, X.-S. Hua, L. Yang, M. Wang, and H.-J. Zhang, "Tag ranking," in *WWW*, 2009.

[33] D. Yin, Z. Xue, L. Hong, and B. D. Davison, "A probabilistic model for personalized tag prediction," in *ACM SIGKDD*, 2010.

Eduardo J. Ruiz is a Ph.D. student at the Department of Computer Science & Engineering in UC Riverside. Previously he received his M.Sc. and B.S. at the Universidad Simon Bolivar, Venezuela.

Vagelis Hristidis is an Associate Professor of Computer Science at UC Riverside. His key areas of expertise are Databases, Information Retrieval, and particularly the intersection of these two areas. His work in these areas has received more than 3,000 citations according to Google Scholar. His key achievements also include the NSF CAREER award, a Google Research Award, an IBM Award, and a Kauffmann Entrepreneurship Award.

Panos Ipeirotis is an Associate Professor and George A. Kellner Faculty Fellow at the Department of Information, Operations, and Management Sciences at Leonard N. Stern School of Business of New York University. He has received three Best Paper awards (IEEE ICDE 2005, ACM SIGMOD 2006, WWW 2011), two Best Paper Runner Up awards (JCDL 2002, ACM KDD 2008), and is also a recipient of a CAREER award from the National Science Foundation and several other industry grants.