# DataSift: A Crowd-Powered Search Toolkit

Aditya Parameswaran
Stanford and U. Illinois (UIUC)
adityagp@illinois.edu

Ming Han Teh
Stanford University
minghan@cs.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

## ABSTRACT

Traditional search engines are unable to support a large number of potential queries issued by users, for instance, queries containing non-textual fragments such as images or videos, queries that are very long, ambiguous, or those that require subjective judgment, or semantically-rich queries over non-textual corpora. We demonstrate DataSift, a crowd-powered search toolkit that can be instrumented over any corpus supporting a keyword search API, and supports efficient and accurate querying for a rich general class of queries, including those described previously. Our demonstration will allow conference attendees to issue live queries for image, video, and product search, as well as "play back" the results of a wide variety of prior queries issued on DataSift. Attendees will also be able to perform a side-by-side comparison between DataSift and traditional retrieval schemes.

## 1. INTRODUCTION

While information retrieval systems have come a long way in the last two decades, modern search engines still have quite limited functionality. For example, they have difficulty with:

- Non-textual queries or queries containing both text and non-textual fragments: For instance, a query *"cables that plug into <IMAGE>"*, where *<IMAGE>* is a photo of a socket, cannot be handled by any current search engine.
- Queries over non-textual corpora: For instance, a query *"funny pictures of cats wearing hats, with captions"* cannot be handled adequately by any image search engine. Typical image search engines only perform keyword search over image tags, which may not be sufficient to identify if an image satisfies the query.
- Long queries: For instance, a query *"find noise canceling headsets where the battery life is more than 24 hours"* cannot be handled adequately by a product search engine, e.g., Amazon (`www.amazon.com`). Search results are often very noisy for queries containing more than 3-4 keywords.
- Queries involving human judgment: For instance, a query *"apartments that are in a nice area near Somerville"* cannot be handled adequately by an apartment search engine, e.g., Craigslist (`www.craigslist.com`).
- Ambiguous queries: For instance, a query *"running jaguar images"* cannot be handled adequately by an image search engine. Search engines cannot tease apart queries which have multiple or ambiguous interpretations, e.g., the car vs. the animal.

For all of these types of queries, currently the burden is on the user to attempt to express the query using the search interfaces provided. Typically, the user will try to express his or her query in as few textual keywords as possible, try out many possible reformulations of the query, and pore over hundreds or thousands of search results for each reformulation. For some queries, e.g., *"buildings that look like <IMAGE>"*, identifying a formulation based solely on text is next to impossible.

Additionally, there are cases where the user does not possess the necessary knowledge to come up with query reformulations. For instance, for the query *"cables that plug into <IMAGE>"*, a particular not-so-technologically-savvy user may not be able to identify that a socket is of a particular type, say a USB 2.0 socket.

To reduce the burden on the user, both in terms of labor (e.g., in finding reformulations and going through results) and in terms of knowledge (e.g., in identifying that a socket is indeed a USB 2.0 socket), we turn to humans (i.e., the crowd) for assistance. We have developed DataSift, a powerful general-purpose search toolkit that uses humans (i.e., crowd workers) to assist in the retrieval process. Our toolkit can be connected to any traditional corpus with a basic keyword search API. DataSift then automatically enables rich queries over that corpus. Additionally, DataSift produces better results by harnessing human computation to filter answers from the corpus.

We propose to demonstrate DataSift in action, both with prerecorded queries as well as with live queries that are handled by Amazon Mechanical Turk (`mturk.com`) workers. The demonstration will illustrate how DataSift works, how it can outperform traditional approaches, and some of the remaining challenges. The rest of this proposal is divided into two parts: We first give an overview of DataSift and its components. In the second part, we describe the proposed demonstration.

The first part of the proposal is a brief summary of a companion paper [8] that appeared at the HCOMP Conference in November 2013. In the companion paper, we additionally describe in detail the different ways DataSift can be built, their pros and cons, and a thorough performance evaluation of these ways or schemes versus each other and versus traditional retrieval schemes. We also studied how DataSift parameters may be tuned to achieve even higher accuracy for fixed cost.

Figure 1 shows a high-level overview of DataSift: The user provides a rich search query $Q$ of any length, that may include textual and/or non-textual fragments. DataSift uses an internal pipeline that makes repeated calls to a crowdsourcing marketplace, specifically, Mechanical Turk (`mturk.com`), as well as to the keyword search interface to the corpus. When finished, a ranked list of results are presented back to the user, like in a traditional search engine. As an example, Figure 2 depicts the ranked list of results for the query $Q$ = *"type of cable that connects to <IMAGE: USB B-Female socket of a printer>"* over the Amazon products corpus. The ranked results provide relevant USB 2.0 cables with a B-Male connector.

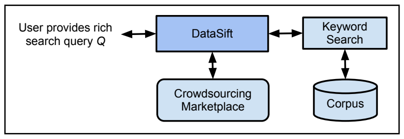
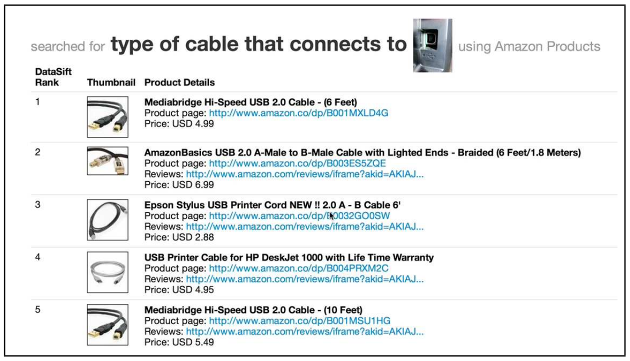


**Figure 1:** DataSift **Overview**



searched for **type of cable that connects to** using Amazon Products

| DataSift Rank | Thumbnail | Product Details |
|---|---|---|
| 1 | | **Mediabridge Hi-Speed USB 2.0 Cable - (6 Feet)**<br>Product page: http://www.amazon.co/dp/B001MXLD4G<br>Price: USD 4.99 |
| 2 | | **AmazonBasics USB 2.0 A-Male to B-Male Cable with Lighted Ends - Braided (6 Feet/1.8 Meters)**<br>Product page: http://www.amazon.co/dp/B003ES5ZQE<br>Reviews: http://www.amazon.com/reviews/iframe?akid=AKIAJ...<br>Price: USD 6.99 |
| 3 | | **Epson Stylus USB Printer Cord NEW !! 2.0 A - B Cable 6'**<br>Product page: http://www.amazon.co/dp/B0032GO0SW<br>Reviews: http://www.amazon.com/reviews/iframe?akid=AKIAJ...<br>Price: USD 2.88 |
| 4 | | **USB Printer Cable for HP DeskJet 1000 with Life Time Warranty**<br>Product page: http://www.amazon.co/dp/B004PRXM2C<br>Reviews: http://www.amazon.com/reviews/iframe?akid=AKIAJ...<br>Price: USD 4.95 |
| 5 | | **Mediabridge Hi-Speed USB 2.0 Cable - (10 Feet)**<br>Product page: http://www.amazon.co/dp/B001MSU1HG<br>Reviews: http://www.amazon.com/reviews/iframe?akid=AKIAJ...<br>Price: USD 5.49 |

**Figure 2:** DataSift **Example**

A disadvantage of our approach is that the response time will be substantially larger than with a traditional search engine. Thus, our approach is only applicable when the user is willing to wait for higher quality results, especially when the user himself does not know whether the results are correct (for example, in the USB cable case), or when he is not willing or capable of putting in the effort to find items that satisfy his query. Our experience so far is that the wait times are of the order of 20 minutes to an hour. (Note that users can see partial results as they come in.) We discuss mechanisms for addressing this issue for our demonstration scenarios in Section 3.

We now present some challenges in building DataSift, using our earlier example: $Q$ = *"type of cable that connects to <IMAGE>"*. We assume that we have a product corpus (e.g., Amazon products) with a keyword search API. Consider the following three (among others) possible configurations for DataSift:

- **Gather:** Provide $Q$ as is to a number of human workers and ask them for one or more reformulated textual keyword search queries, e.g., *"USB 2.0 Cable"* or *"printer cable"*. Then retrieve products using the keyword search API for the reformulated keyword search queries and present the results to the user.
- **Gather-Filter:** The same configuration as **Gather**, but in addition ask human workers to filter the retrieved products for relevance to the query $Q$, e.g., whether they are cables that plug into the desired socket, before presenting the results to the user.
- **Iterative Gather-Filter:** The same configuration as **Gather**, but in addition first ask human workers to filter a small sample of retrieved products from each reformulated textual query for relevance to $Q$, allowing us to identify which reformulations produce better results. Then, retrieve more from the better reformulations, e.g., more from *"USB 2.0 printer cable"* instead of *"electronic cable"*. Finally, ask human workers to filter the retrieved results before presenting the results to the user.

In addition to determining which configuration we want to use, each of the configurations above has many parameters that need to be tuned. For instance, for the last configuration, DataSift needs to make a number of decisions, including:

- How many human workers should be asked for reformulated keyword search queries? How many keyword search queries should each worker provide?
- How many items should be retrieved initially for each reformulation? How many should be retrieved later (once we identify which reformulations produce better results)?
- How do we decide if a reformulation is better than a different one?
- How many human workers should be used to filter each product for relevance to $Q$?
- How should the cost be divided between the steps?

Our current implementation of DataSift is powerful enough to be configured to match all of the configurations we have described, plus others. We achieve this flexibility by structuring the toolkit as six plug-and-play components that can be assembled in various ways.

To the best of our knowledge, we are the first in addressing the problem of designing a rich general-purpose search toolkit augmented with the power of human computation. By themselves, traditional information retrieval techniques are insufficient for our human-assisted retrieval task. On the other hand, existing crowd-powered systems, such as Turkit [5], and CrowdPlan [4] do not address the problem of improving information retrieval. CrowdQ [2] tackles the problem of using humans to detect common patterns in search queries, and could be used in conjunction with DataSift to detect and cater for the more popular patterns. Unlike DataSift, CrowdQ does not support rich queries containing images or videos. Unlike social or collaborative search, e.g., [3, 1, 6], we do not leverage the social network, and the system moderates the interaction using reformulations and filtering to ensure high quality results.

## 2. ARCHITECTURE

A user enters a query into DataSift, which could contain textual and non-textual fragments. The corpus of items $I$ (products, images, or videos) over which DataSift is implemented has a keyword search API: it accepts a textual keyword search query and a number $k$, and returns the top $k$ items (products, images, or videos) along with their ranks. The crowdsourcing marketplace $M$ may be used to recruit multiple human workers to attempt each task independently, and returning the answers to DataSift. DataSift makes repeated calls to both $I$ and $M$, and then eventually provides the user with $n$ items in ranked order.

### 2.1 Components

Next, we describe the components internal to DataSift. Components are categorized into: (1) Crowdsourced Components: components that interact with the crowdsourcing marketplace, and (2) Automated Components: components that function independent of the crowdsourcing marketplace. The function signatures and specifications of these components can be found in our companion paper [8].

#### *Crowdsourced Components*

- (G) **Gather Component**: The Gather Component G asks human workers for fully textual reformulations for $Q$, asking human workers to respond to the task "Please provide reformulated keyword search queries for the following query: $Q$". This component provides DataSift a mechanism to retrieve items (using the keyword search API) for $Q$ when $Q$ contains non-textual fragments.

- (F) **Filter Component**: The input to the Filter Component F is a set of items. For each item, F determines whether the item satisfies the query $Q$ or not, by asking human workers to respond to the following task: "Does the item $i$ satisfy query $Q$: (Yes/No)". Since human workers may be unreliable, multiple workers may be asked to respond to the same task on the same item $i$. The number of humans asked is determined by designing a filtering strategy [7] using the overall accuracy threshold $t$ (set by the application designer).

*Automated Components*

- (R) **Retrieve Component**: The Retrieve Component R uses the keyword search API to simultaneously retrieve items for multiple textual queries $T$ from the corpus. For each textual query $T$, items are retrieved along with their keyword search result ranks for $T$ (as assigned in the output of the keyword search API call by the corpus).
- (S) **Sort Component**: The Sort Component S merges rankings, providing a rank for every item based on how well it addresses $Q$. If preceded by the F component, then S returns a rank for every item based on the difference between the number of YES and NO answers for each item. If preceded by the R component, then S simply merges the ranks arising from the results of the textual retrieval.
- (W) **Weighting Component**: For Iterative Gather-Filter (Section 1), the weighting component is the component that actually evaluates reformulations. The component always follows F, using the results from F to compute weights corresponding to how good different reformulations are in producing items that address $Q$.

  Component W receives as input items from the Filter Component F and the textual query $T$ that generated the items. For each textual query $T$, given the output of the filtering component F, the weighting component returns a weight based on how useful the textual query is in answering $Q$. These weights are then used by the R component in preferentially retrieving more items from better reformulations.

## 2.2 Configurations

We now describe the configurations that DataSift can support, among others. (We focus on those that we think are the most interesting and important.) We will allow conference attendees to examine the performance for each of these configurations.

- RS: (Only possible if $Q$ is textual) This configuration refers to the traditional information retrieval approach: component R uses the query $Q$ to directly retrieve the top $n$ items with ranks using the keyword search API. In this case, component S does nothing, simply returning the same items along with the ranks.
- RFS: (Only possible if $Q$ is textual) From the items retrieved by component R, component F uses humans to better identify which items are actually relevant to the query $Q$. Component S then uses the output of F to sort the items in the order of the difference in the number of Yes and No votes for an item as obtained by F, and return $n$ items along with their ranks.
- GRS: (Gather from Section 1) Component G gathers textual reformulations for $Q$, asking human workers for reformulations. Subsequently, R retrieves the top items along with ranks for each of these reformulations. Then, S sorts the items by simply merging the ranks across the reformulations, with ties being broken arbitrarily. Items are returned along with their ranks.
- GRFS: (Gather-Filter from Section 1) Component G gathers textual reformulations, after which component R retrieves items with ranks for each of the reformulations. Then, component F filters the items using human workers. Subsequently, the items are sorted by component S based on the difference in the number of Yes and No votes for each item, and the top $n$ are returned along with their ranks; ties are broken arbitrarily.
- GRFWRFS: (Iterative Gather-Filter from Section 1) Component G gathers textual reformulations, after which component R retrieves a few items from each of the reformulations. Component F then filters the set of retrieved items. The output of F provides us with an initial estimate as to how useful each reformulation is in answering the query $Q$.

  Subsequently, component W computes a weight for each of the textual reformulations based on the results from F. These weights are then used by component R to preferentially retrieve additional items from reformulations in proportion to the weight. Component F filters the retrieved items once again. Eventually, the component S sorts the items in the order of the difference between the number of Yes and No votes, and returns the items along with their ranks.

| **Easy Queries (5)** |
|---|
| funny photo of barack obama eating things |
| bill clinton waving to crowd |
| matrix digital rain |
| eiffel tower, paris |
| 5 x 5 rubix cube |

| **Hard Queries (5)** |
|---|
| tool to clean laptop air vents |
| cat on computer keyboard with caption |
| handheld thing for finding directions |
| the windy city in winter, showing the bean |
| Mitt Romney, sad, US flag |

| **Selected Others** |
|---|
| funny photos of cats wearing hats, with captions |
| the steel city in snow |
| stanford computer science building |
| database textbook |

**Table 1: Sample textual queries**

## 3. DEMONSTRATION SCENARIOS

A challenge in demonstrating DataSift is that DataSift takes much longer than traditional web search (producing higher quality human-curated results). While users will be able to see partial results for queries as they come in and see how their query is progressing, typical DataSift queries take twenty minutes to half an hour to complete. These inherent delays means that we need to be creative in demonstrating the system.

We consider two demonstration scenarios: the first scenario involves browsing pre-recorded queries. For a large set of queries (described below) issued to DataSift, we have recorded every action taken within DataSift to produce results for the queries (e.g., when DataSift moves from component G to F, or when it decides to get an additional human opinion on an item in F) and outside DataSift (i.e., responses from human workers), and allow attendees to "play back" every step of the action. This scenario is analogous to videos that show how to bake a cake: you begin by showing how to mix the ingredients; but then, instead of waiting, you fast forward through the intermediate steps to the finished product.

The second scenario involves attendees issuing live queries to DataSift, viewing progress of live queries issued by other attendees, and lastly, but certainly not the least, acting as human workers and answering tasks generated by queries issued by other attendees. Due to the challenges described earlier, we will encourage attendees to (a) fill out an email address field to have the results for the queries they issued sent to them when complete, and/or (b) return within half an hour to check the results of their queries. (Alternatively, if demo sessions are split between two days, attendees will be able to view the results of the queries they issued the previous day.)

## 3.1 Scenario 1: Pre-recorded Queries

In this scenario, we will allow conference attendees to play back, for a broad variety of queries, how they were addressed by various DataSift configurations. This will allow attendees to get a better intuition for the design choices and challenges underlying DataSift. For example, will allow attendees to view:

- how each search query was broken down into questions asked to humans;
- how, in the case of ambiguity in F, additional human workers were asked the same question;
- what kinds of interfaces were used to ask questions to human workers, and why they were designed that way;
- how, even though some human workers were answering questions incorrectly, DataSift was able to recover and reduce the impact of workers with poor ability;
- how GRFWRFS ends up giving better results than GRFS due to the reweighting component; and
- the cases where DataSift gave less-than-perfect results, and why.

The list of textual queries that we plan to allow attendees to peruse include the ones shown in Table 1, while the list of "rich media" queries (i.e., queries containing non-textual fragments such as images or videos) that we plan to allow attendees to peruse include the ones shown in Table 2.

We will also allow conference attendees to study side-by-side comparisons between the same queries addressed by various crowd-powered DataSift configurations and the traditional fully-automated retrieval scheme (RS). Since rich media queries are simply not supported by traditional retrieval approaches, for the side-by-side comparison, we can only support fully textual queries. We will demonstrate to the attendees that GRFWRFS clearly outperforms all other configurations, with up to 200% higher precision than RS on average. Furthermore, we will demonstrate that GFRWRFS significantly outperforms the simpler configurations such as GRFS and GRS, because GFRWRFS preferentially retrieves more from the better reformulations.

## 3.2 Scenario 2: Live Queries

In this scenario, we will allow conference attendees to pose queries to DataSift. DataSift supports four keyword search corpora: Youtube Videos (`youtube.com`), Amazon Products (`amazon.com`), Google Images (`images.google.com`), and Shutterstock Images (`www.shutterstock.com`). Attendees will be able to issue queries on-the-fly to any of these corpora, and view the progress of queries issued by other attendees.

We will support two crowdsourcing marketplaces for live queries: first, Amazon's Mechanical Turk, and second, the marketplace of SIGMOD conference attendees — that is, we will allow conference attendees to act as workers, and complete tasks generated from queries issued by other attendees. To support the latter interaction, we will have two laptops set up, one with the DataSift query interface, and one with the conference attendee marketplace interface.

| **Rich Queries (5)** |
|---|
| buildings around <IMAGE: UC Berkeley's Sather Tower> |
| device that reads from <IMAGE: Iomega 100MB Zip Disk> |
| where to have fun at <IMAGE: Infinity Pool at Marina Bay Sands hotel in Singapore> |
| tool/device that allows me to do hand gestures such as in: <VIDEO: motion sensing demonstration using fingers > |
| type of cable that connects to <IMAGE: USB B-Female socket of a printer> |

**Table 2: Sample Rich Queries**

Consider the following trace of attendees interacting with DataSift and the conference attendee marketplace:

- Attendee X issues two queries Q1 and Q2.
- Attendee Y issues two queries Q3 and Q4 of their own, then logs onto the conference attendee marketplace, answers tasks generated by (say) component G for Q1 and Q2
- Attendee Z logs into the conference attendee marketplace, answers tasks generated by (say) component F for Q1 and Q2, and by (say) component G for Q3 and Q4
- ...

As the above trace indicates, as part of this scenario, conference attendees will be able to play the role of both the potential users of DataSift, as well as the human workers working on tasks generated by DataSift. This scenario will demonstrate that, perhaps somewhat surprisingly, even though some conference attendees may suggest poor reformulations for the G component or answer filtering questions incorrectly for the F component, DataSift is resilient enough to ensure that the final results returned are unaffected by individual incorrect responses and still have a high degree of precision.

## 4. REFERENCES

[1] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *WWW*, 2008.

[2] G. Demartini, B. Trushkowsky, T. Kraska, and M. J. Franklin. Crowdq: Crowdsourced query understanding. In *CIDR*, 2013.

[3] D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *WWW*, 2010.

[4] E. Law and H. Zhang. Towards large-scale collaborative planning: Answering high-level search queries using human computation. In *In AAAI*, 2011.

[5] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP*, 2009.

[6] M. R. Morris and J. Teevan. Collaborative web search: Who, what, where, when, and why. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–99, 2009.

[7] A. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD*, 2012.

[8] A. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom. Datasift: An expressive and accurate crowd-powered search toolkit. In *HCOMP*, 2013.