Graffiti: A System for Predicting Tags of Web Documents

Andrew Weinrich

May 15, 2007

Abstract

Social bookmarking or tagging sites provide an easy way for users to annotate documents, but the lack of structure in simple tags hinders efforts to analyze that metadata. This paper proposes a model for using the words and phrases contained in web documents to predict the tags that users will attach to the document, a description of a system that implements this model, and a metric for evaluating the performance of such a system.

1. Introduction

Of all the new online services developed in the recent Cambrian explosion of "Web 2.0", one of the most popular categories is tagging sites, sometimes called social bookmarking or social linking. On these sites, users create profiles under which to store their Web bookmarks. Bookmarking was of the first information management tools provided by web browsers, but social bookmarking sites provide several additional features, notably the ability to "tag" bookmarks with a short word or phrase describing the page's content. A user could, for example, annotate all of their bookmarks relating to the Ruby programming language with the tag `ruby`, thus imparting some information that makes a semantic judgement about the "meanings" of those pages; a search for the tag `ruby` would then return all those pages without requiring extra organization.

The abundance of tagged documents suggests that there may be some way of analyzing the content of these documents to discover structure that leads users to choose a particular tag for a document over another. A system that could learn from existing tagged documents could then use the patterns it finds to predict the tags of future documents. Tagging, as commonly used on the internet, has several properties that aid such prediction efforts:

- Tagging is an all-or-nothing activity: for every possible tag, and article either has the tag or does not. This allows the prediction function to be more flexible, by associating a probability or "confidence level" to its predictions. Although an article either will or will not have the tag "python", for example, our function may give the article a prediction of "There is a 75% chance that this article will be tagged with python".

- Each tag is a single entity, unrelated to other tags. At the time of this writing, efforts to categorize or classify tags (e.g. into hierarchies) are not widespread. Each tag may thus be considered in isolation, without worrying how the presence or absence of other tags *either in the prediction or in real life* will affect it.

- Both document locations and tags are typically available from bookmarking sites by RSS, a format designed for ease of interpretation by automated agents. This eliminates the requirement to perform "screen-scraping" of HTML to find tagging structure. Individual documents must be parsed for keywords, but this is a considerably easier task, and the same general algorithm may be used on virtually all documents.

- Tags are completely devoid of semantic meaning. They may be proper names of people, names of companies, actions, abstract concepts, judgements, or made-up words, and thus may all be treated the same way.

- Theoretically sound machine learning algorithms can be rendered useless by small or unrepresentative training samples. If a project's goal is to simulate

> human intelligence and judgement, some amount of that intelligence must be provided up front. The explosion of tagging sites in the last few years, however, has shown that there are many thousands of users who are willing, and even eager, to read, analyze, and tag documents, and then freely share their results.

Graffiti is a system that uses the tagging efforts of a bookmarking site's user community to make predictions about future documents, based on the presence or absence of specific terms in the document text. It is a learning system that continually refines its predictions based on new document data and its performance against the known tags of those documents.

2. Data Model

Graffiti's data model consists of four entities:

1. Tag: a simple string used to annotate a document, with no guarantees as to its meaning or use
2. Term: a word or short phrase that occurs in a document
3. Document: an object that comprises a unique identifier (in our system, a URL), a set of terms with weights, and a (possibly empty) set of tags
4. Keyword-tag relationship: a tuple that represents how often a keyword is found in documents that are determined by the user population to have a certain tag

The association of both keywords and tags to documents are stored using the standard inverted-list model of information retrieval. The keyword-tag relationship is a simple ratio of the number of times a keyword occurs in document with a given tag to the total number of documents with that tag. For example, if the term "iPod" appears in half the documents that have the tag "apple", then the relationship tuple would be $(ipod, apple, 0.5)$.

By weighting and summing the predictions of the individual terms in a document, Graffiti generates a prediction about the presence of tags on that document. Due to the complete lack of structure or meaning in tags, no other judgement is made besides a prediction that, given the occurence of a term, there is a certain probability that the document will have a particular tag.

2.1. Keyword Restriction

In addition to restricting the size of the tag set, Graffiti must also restrict the keyword set. Information retrieval systems that are based on user-defined keyword searches put this onus on the user; if a search for "go" or "lack" returns meaningless results, the user will have to refine the query. However, for tagging prediction, we will be looking at all possible keywords, including those that are "noise", beyond even traditional stop-words such as "the" and "and". It is possible that a term like "new" will have roughly equal predictive values for every possible tag; such a term that predicts everything effectively predicts nothing.

As attempting to compose an exhaustive list of noise terms in advance is futile, Graffiti instead uses the common Term Frequency - Inverse Document Frequency weighting system to identify and discount noise terms. Many variants of this formula have been developed over previous decades; our version is stated in the Prediction Model section. When calculating the contribution of each term to a tag prediction, the term's weight comes from this TF-IDF formula. Thus, terms that have a high predictive value because they are extremely common will hopefully have a lowereffect relative to other terms that also have a high correlation but do not occur as frequently.

The above TF-IDF formula is the only one that Graffiti uses to discount the weight of noise terms. There is considerable room for experimentation and refinement in this area, however, and it is possible the use of a better weighting formula will produce more accurate predictions.

2.2. Multi-word Terms

When building the index of document terms, we will likely want to include multi-word phrases. For example, the full phrase "Steve Jobs" is likely to be an excellent predictor of the tags "apple" or "ipod"; the individual words "steve" and "jobs" are probably useless. However, most standard search engines do not store phrases, but only individual terms. They rely on position data attached to the terms to implement exact-phrase searches; thus, the search engine does not "know about" phrases in the document until it specifically goes to look for them.

This is a problem for Grafitti, which should be able to use multi-word phrases for predictions without knowing the phrases ahead of time. There are two approaches to solving this problem: one is to ignore it completely, and depend upon only single terms; the other is to use some strategy, beyond that of a conventional IR indexing engine, to find meaningful multi-word phrases in the document, at which point they are treated as regular terms in our model.

One strategy might be to use techniques from the Natural Language Processing field to parse documents. An NLP processor could use its understanding of the document's language both to identify legitimate multi-word phrases and to cull out terms that are unlikely to be useful, beyond the primitive list of traditional stop-words. Graffiti forgoes such subtlety in favor of brute force, by treating every two-word span as a possible term. This doubles the size of our term set and introduces a good deal of "garbage" terms. Graffiti compensates by ignoring terms (of any length) that have not appeared in a certain number of documents (the threshold is initially set at 10). This prevents our prediction system from being cluttered with meaningless multi-word terms, but allows potentially important phrases like "Web 2.0" to come through.

2.3. Prediction Model

To compute its predictions for document tags, Graffiti uses a model similar to the Vector Space form commonly used for keyword matching. However, instead of the vector space having a dimension for each term, and the individual vectors being documents, Graffiti's space has dimensions that correspond to tags, and vectors that represent terms. The components of the vector are the pre-computed correspondences between the term and its occurence in documents.

First, we define the three fundamental objects that we will be using: a set of terms, T; a set of tags, A; and a set of documents, $D \equiv \{(t, a) : t \subset T, a \subset A\}$, each document being a subset of terms and tags.

We use the notation $d[a]$ and $d[t]$ to indicate that a document d contains a particular tag or term, and $\overline{d[a]}$ and $\overline{d[t]}$ to indicate the opposite (terms that have not met the required frequency threshold are omitted). $|d[t]|$ denotes the number of occurrences of a term in a document.

Given a set of tags A, we define the "prediction vector space" S_A of dimension $|A|$, such that each component vector of the absolute basis of the space is associated with a tag $a \in A$.

For each term t we construct a vector V_t in this space. The components of this vector are restricted to the range $[0, 1]$. Each component V_{ta} represents the probability that, given that the term the vector represents is present in a document d, that document will be annotated by the community with the corresponding tag a:

$$V_{ta} = P(d[a] \mid d[t])$$

This probability is determined by a simple calculation of how many documents contain a, divided by the total number of documents, restricted to the subset of documents D_t that contain term t:

$$D_t = \{d \in D : d[t]\}$$

$$V_{ta} = \frac{|\{d \in D_t : d[a]\}|}{|D_t|}$$

To calculate the tag predictions for a document d, we first collect every vector V_t for which d contains t (we define this subset of terms as T_d):

$$T_d = t \in T : d[t]$$

Then, for each vector we calculate the weighting function $W(v)$, which in our implementation is a standard TF-IDF function (the first factor is the Term Frequency, and the second is Inverse Document Frequency):

$$W(d, t) = \left(\frac{|d[t]|}{\sum_{t_k \in T} |d[t_k]|} \right) \left(\log \frac{|D|}{|\{d \in D : d[t]\}|} \right)$$

The final prediction that a test document d will be marked with a tag a is:

$$P(d, a) = \frac{\sum_{t \in T} V_{ta} W(d, t)}{|T_d|}$$

Given these definitions, we can provide an intuitive description of Graffiti's model: each term in a document represents a point mass in an n-dimensional unit hypercube. Each axis of the hypercube corresponds to a tag, and the coordinate of a term along that axis is the likelihood that, given the presence of that term in a document, the document is marked with that axis' tag. The mass of each point is its TF-IDF score.

The final prediction, then, is at the center of mass of all these points. Once the location of that center is determined, we take the coordinates of the centroid along each tag-axis as the probability that the document in question will be marked with that tag.

This model has the advantage that calculating the predictions for a document is very simple. After parsing the document into terms, we retrieve the term prediction vectors from the database, along with the IDF portion of each term's score. For each tag, we simply average the TF-IDF weight of each term times its prediction.

2.4. Prediction Performance

Graffiti has a performance metric that transforms the model's confidence interval into a simple yes-or-no prediction: the system administrator sets a deviation-from-mean threshold, and any prediction above that threshold becomes a "yes", while anything below becomes a "no", and anything in the middle is "no comment". With the default deviation of 15%, this means that any prediction of 65% or above is "yes", 35% or below is "no", and the metric discards any predictions that fall in the middle.

The performance metric then gives Graffiti one point of every correct positive prediction it makes; negative predictions – i.e. Graffiti says that a document will not have a tag and it doesn't – are not counted, because most documents will only have a handful of tags out of the 140 possible. Graffiti loses a point every time it errors: if it says a document will have a tag and it doesn't, or says that the document won't have a tag and it does.

The final score is the number of points Graffiti accumulates divided by the number of predictions it makes. Under this metric, each tag is treated individually, to acknowledge that broad, vague tags like "software" or "development" may not be as easily predicted as more specific tags like "python" and "perl".

3. Implementation

3.1. Project Restrictions

Although the general outline of tagging prediction is quite general, Graffiti restricts its implementation in the following ways:

- The website del.icio.us is used as the sole source of tags and articles. Del.icio.us was one of the first social bookmarking sites, and has over two million registered users (fewer active), making it the largest source of documents and tags that fits our needs.

- The learning portion of Graffiti uses only documents currently available from del.icio.us's feed, without going back into the site's archives.

- For performance reasons, we could not use a single relational database to store all of the project data. Instead, Graffiti uses a custom tiered caching architecture that reduces the amount of central coordination required.

- Only the most popular tags on del.icio.us are considered. Due to the nature of del.icio.us's user population, this list of 139 tags skews heavily towards technology, particularly programming and the Web. If a site with a different focus, such as DailyKos (politics) or Fark (humor), were used, it would change not only the set of tags but also the system's predictive abilities.

- Non-English websites (those not in the top level `com`, `net`, `org`, or `edu`, `info`, `biz`, `au`, or `uk` domains) were not considered, nor were documents whose HTTP headers indicated that they were not in English.

3.2. Implementation Details

Graffiti is implemented in Java, and is composed of three parts that share a common database:

1. The **document aggregator** retrieves as many document-tag pairs as it can from the document source, downloads the documents from the Web, parses them into terms, then stores the term-document associations in the database. This step need only be performed once, when it generates the initial

2. After a sufficient number of documents have been stored, the **prediction collator** calculates the term IDF and tag-prediction values.

3. The final step is the **performance analyzer**, which is an extension of the aggregator. As well as storing the documents, it also predicts the tags that it think the documents will have, and records these predictions. The performance metric can be computed for each tag after the performance analyzer has examined a sufficient number of documents.

After each run of the performance analyzer, the system parameters may be tweaked. The prediction collator is then re-run over the new documents added during the analysis phase. Graffiti requires a very large number of documents for each phase; in general, the more documents it processes, the more accurate its predictions will be. This leads to sev-

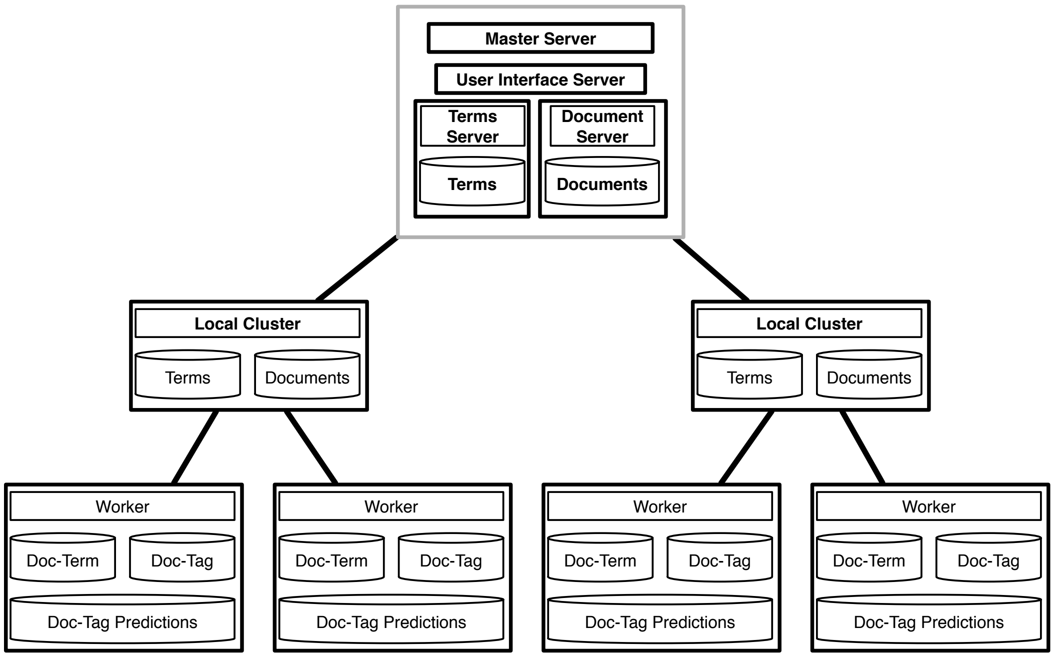

Figure 1: Graffiti architecture diagram. Because the set of tags is small and static, it is not shown in the databases here.

eral interesting design problems, described below.

3.3. Architecture

There are two main bottlenecks in the document aggregator:

1. All of the documents come from remote locations, requiring tens of thousands of individual HTTP requests to unique servers

2. All document and term data must be centrally coordinated, so that all parts of the system will use the same ID numbers for documents and terms

The first bottleneck is the most immediately obvious; an early, single-threaded version of the document aggregator required over two days to process fewer than 4,000 documents. This suggests that multiple clients should be used, perhaps with each having responsibility for one tag. However, using one client for each tag would expose the second bottleneck, as each client tried to simultaneously connect to the database and make updates to the same tables. Graffiti solves this problem by assigning the clients to local cache clusters, which then communicate with the main database on need.

3.4. Database Structure

As shown in Figure 1, Graffiti uses a three-level tiered architecture to alleviate both bottlenecks. A single worker is created for each tag, and has sole responsibility for querying the document source (i.e. `del.icio.us`) and processing that tag's documents. Each worker is assigned to a cluster, which it uses to take load off the main database. For efficiency, each database is stored as a SQLite file.

At the core of the system is a set of central servers. Two dedicated processes manage the master lists of terms and documents. A user interface server communicates commands (start or stop workers, show current status, run prediction collation, shutdown system, etc) from the user to the the Grafitti master server, which manages all other elements of the system. Currently, the interface system takes commands from a command line, although it is possible to substitute other interfaces, such as a GUI or web application.

All communication between the components is by message passing. All distinct entities (workers, cluster nodes, master, interface, central DBs) have unique numeric identifiers used to denote the source and destination of a message, which may be sent synchronously or asynchronously. When a node needs the ID number for a new term, it sends a message to the central term server and waits for the response. Certain messages, such as "Stop Execution", have priority over other messages.

All communication is through an abstract MessageCenter object, which handles the details of inter-thread, -process, and -host communications. This communication system allows for fine-grained distribution of tasks among processes and hosts. In the current Grafitti system, each cluster is a single process, with the workers running in separate threads. There is a single master. However, the system may be configured to separate each worker into a separate process, and to similarly separate the master servers. The central configuration file also describes how the cluster nodes and workers are to be distributed among available hosts.

3.5. Document Processing Procedure

When a worker finds a term in a document it is processing, it performs the following actions:

1. The worker checks the local cluster database to find the ID of the term.

2. If the term is not present in the local cluster, it asks the master database for the term's ID.

3. If the central term database has already registered that term, it returns the ID number. Otherwise, it creates a new term entry with a new unique ID and returns that ID.

4. The worker records the term in the cluster database.

5. Finally, the worker records the document-term association in its personal database.

A similar procedure is followed with new documents, with the exception that the main document database also tells the workers whether or not the document has been processed; if it has, the worker skips that document but still records its presence in the cluster database. In this way, the contents of the central document and term databases are replicated "on need" to each of the cluster copies.

The document aggregator and performance analyzer share this general structure; the performance analyzer also records predictions in the worker databases. The prediction collator is a separate program, which pulls in data from each of the worker databases to calculate each term's prediction.

3.6. Multi-Word Terms

As described earlier, phrases that consist of multiple words may be very valuable for predictive use, but are usually not stored in a usable form by most search engine databases. Our implementation uses a brute-force approach: we treat every two- and three-word substring of a document as a term, and record them all in the term database. This drastically increases the number of possible terms that we must store (during an early run of the learning phase, fewer than 1000 documents produced over 1.3 million terms).

This approach has clear deficiencies, not least our term catalog is now polluted with a great deal of single-occurrence phrases. Compounding that problem is the fact that we analyze the document after the parsing engine has stripped all punctuation and stop-words, creating illegitimate multi-word terms that span sentence and phrase boundaries. Our hope was that the prediction model would be able to wash out this noise and only make use of the truly valuable multi-word phrases.

To eliminate some of the extraneous phrases, the prediction collator ignores any term that occurs in fewer than 10 documents. These occurrences are not deleted from the database, in case they occur in future documents. The occurence threshold is configurable

4. Results

Forthcoming.

References