A Novel Traffic Analysis for Identifying Search Fields in the Long Tail of Web Sites

George Forman
ghforman@hpl.hp.com

Evan Kirshenbaum
evan.kirshenbaum@hp.com

Shyamsundar Rajaram
shyam.rajaram@hp.com

HP Labs
1501 Page Mill Rd.
Palo Alto, CA, 94304 USA

ABSTRACT

Using a clickstream sample of 2 billion URLs from many thousand volunteer Web users, we wish to analyze typical usage of keyword searches across the Web. In order to do this, we need to be able to determine whether a given URL represents a keyword search and, if so, which field contains the query. Although it is easy to recognize 'q' as the query field in 'http://www.google.com/search?hl=en&q=music', we must do this automatically for the long tail of diverse websites. This problem is the focus of this paper. Since the names, types and number of fields differ across sites, this does not conform to traditional text classification or to multi-class problem formulations. The problem also exhibits highly non-uniform importance across websites, since traffic follows a Zipf distribution.

We developed a solution based on manually identifying the query fields on the most popular sites, followed by an adaptation of machine learning for the rest. It involves an interesting case-instances structure: labeling each website *case* usually involves selecting at most one of the field *instances* as positive, based on seeing sample field values. This problem structure and soft constraint—which we believe has broader applicability—can be used to greatly reduce the manual labeling effort. We employed active learning and judicious GUI presentation to efficiently train a classifier with accuracy estimated at 96%, beating several baseline alternatives.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; H.2.8 [**Information Systems**]: Database Applications—*Data mining*; H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Measurement

Keywords

web data mining, clickstream analysis, machine learning classification, active learning

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

1. INTRODUCTION & MOTIVATION

Suppose it were possible to determine what keyword searches are typically performed at sites all across the web—not just at a single site, such as Google or your own private web server. Such information could be useful for marketing, building domain-specific web directories, discovering competitors or potential collaborators, positioning products and future development, etc. For a single, concrete illustration of what could be done with such data, we show for three seed websites in Table 1 their most similar websites, based entirely on the similarity of searches performed by users at those sites. The data indicate that `gap.com` had similar searches as `oldnavy.com` and `lanebryant.com`, with far less overlap with the searches executed at zillions of other websites. This analysis was based on a sample of 2 billion URLs from 236,490 U.S. households who share their browsing clickstreams as part of the Nielsen MegaPanel® dataset.[1] Further details of this particular analysis are outside the scope of this paper, as our goal here is to focus on a key, pre-requisite problem, discussed next.

Table 1: Related websites based on similar searches.

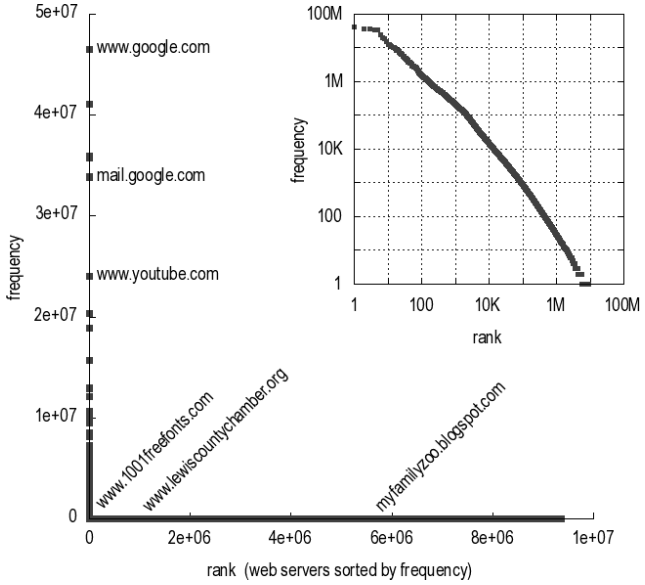
recipe4living.com	hp.com	gap.com
rachaelray.com	staples.com	oldnavy.com
bettycrocker.com	officedepot.com	bananarepublic.com
tasteofhome.com	bestbuy.com	chadwicks.com
pillsbury.com	circuitcity.com	victoriassecret.com
recipes.com	officemax.com	lanebryant.com
cdkitchen.com	tigerdirect.com	childrensplace.com
dlife.com	newegg.com	aeropostale.com

In order to enable such analyses, we need an automated method to determine whether a given URL represents a keyword query by a user, and, if so, which of the set of data fields contains the query itself. Figure 1 shows the query URL that gets sent back to the web server when the user initiates a query from the search box on several example web sites. Following the '?' character is a sequence of field `name=value` pairs from the HTML form. The field that carries the user's query (highlighted in a box only for presentation) is named 'q' in the first example. While this is obvious in these cases, it is difficult to write an accurate decision rule in general.

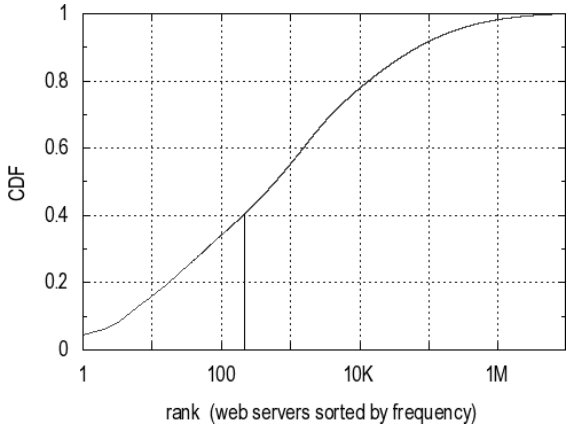
[1] http://www.nielsen-netratings.com/

http://help.beanstalkapp.com/search?**q**=your+search&x=5&y=4
http://www.netflix.com/Search?lnkce=iaswfOf&**v1**=your+search&lnkce=acsNoEnhRt
http://www.recipe4living.com/search?controller=recipes&**searchterm**=your+search
http://search.hp.com/query.html?cc=us&lang=en&charset=utf-8&**qt**=your+search&la=en
http://www.batteries.com/index.asp?N=0&Nu=p_prodcode&Ntk=All&**Ntt**=your%20search&Nty=1
&**D**=your%20search&Ntx=mode+matchpartialmax&Dx=mode+matchpartialmax

Figure 1: Example query URLs that get sent back to their web servers.



(a) Linear graph with log-log inset.



(b) CDF reveals substantial mass in the tail.

Figure 2: The Zipf-like distribution of web site clicks.

Unfortunately for data miners, each website designer is free to use arbitrary field names, with no semantic naming requirements. Moreover, there may be a variety of other fields. Query URLs are often used for purposes that have nothing to do with user queries, e.g., conveying user session number, product identifiers, or preferred language.

We address this problem with a novel application of machine learning classification described in Section 2. The problem has an interesting dual structure, which we exploit to greatly increase the efficiency of manual labeling of training cases. Our software, described in Section 3, also leverages *active learning* to bring the user some of the most useful cases to label. In Section 4 we conduct a series of experiments to compare our recognizer's accuracy against various baselines, such as decision rules that can be implemented in software. The remaining sections discuss related work, conclusions, and future work. We complete the introduction with an additional note on motivation.

This learning problem has the additional aspect that Web traffic is extremely skewed. To illustrate the distribution, Figure 2a (and its log-log inset) shows for our MegaPanel® data sample the number of clicks to each web server on the y-axis and the server's rank on the x-axis. Because this distribution declines so steeply, it is tempting to analyze only the top couple hundred servers, which might be manageably analyzed by hand. But in fact there are many searches performed at the remaining websites. The cumulative distribution of this data is shown in Figure 2b. We see that the top 200 websites cover only about 40% of the total user clicks in our sample, which is typical of *long tail* distributions. Thus, if one desires to analyze user searches for profiling or website characterization, a significant volume of the searches would be missed with any approach that observes searches only at Google or even at the top several search engines. There has been a great deal of existing research on clickstreams and query logs from a single website, typically taken from private web server logs. With our contribution one can get visibility into searches performed at the many smaller websites in the long tail. Arguably, one of the most useful pieces of information that can be observed in a user's data is the free text the user types into website search boxes, since it, more than anything else, is under the user's direct control and indicates the user's immediate interest.

Table 2: Sample field values of 16 URLs for an example web form with three fields.

h_q	h_client	h_page
hp deskjet d1400 series	c-h-r602-1	hpcom
HP Photosmart C5200 all-in-one	s-h-e002-1	hpcom
photosmart express	c-h-r602-1	hpcom
PP2182D	c-h-r602-1	hpcom
PP2182D lithium-ion battery	C-H-R2515-1	hpcom
hp pavilian 6000	C-S-R2515-1	hpcom
lap top battery	C-S-R2515-1	hpcom
HP PSC 1350 Three in One	C-S-R2515-1	hpcom
Officejet 7410	c-s-r295-1	hpcom
tshirt transfer	c-s-r295-1	hpcom
6800series	c-s-r295-1	hpcom
HP 88XL	C-A-R163-1	filter
HP Deskjet D1420	C-A-R163-1	filter
HP Officejet 7210 Oxi	C-A-R163-1	filter
Officejet7210oxi	C-A-R163-1	filter
MFC-7820N	C-A-R163-1	filter

Table 3: Example features generated per instance, based on its column of 100–5000 sample values.

- Percent of values that occur only once
- Percent of values that occur only once if lowercased
- Min, max, average and stdev. of each string feature:
  - String length
  - Average word length
  - Number of words; capitalized words
  - Number of: letters; lowercase; uppercase; digits; hex digits; non-hex letters; whitespace; control characters; non-ASCII
  - Number of each punctuation character, such as '@' (email addresses), '.' (IP addresses), '$' (prices), '_' (identifiers), '-'
- Percent of values having lowercase; uppercase; both
- Percent of positives among training instances with same field name

2. DOMAIN & PROBLEM FORMULATION

In typical clickstream datasets only the URLs are available, not the HTML page contents. Given just a URL, it can sometimes be impossible for a person to determine whether any of its fields represent a keyword query, such as when someone has searched for an error code number on an obscure support website. But by aggregating the various URL samples for a website, the pattern usually becomes clear. Table 2 shows the field values of several separate form submissions at a single website. Each of the three fields is represented by a column of text values. Clearly the first is the query field.

2.1 Case-Instances Structure

In most machine learning work, the terms *instance* and *case* are used interchangeably, but in this domain there is an interesting distinction. We reserve the term *case* to mean a web site form, including all its fields (e.g., Table 2), and the term *instance* to mean a single field of a case including the sample values for the field (e.g., a single column in Table 2). The duality between cases and instances, though peculiar, is applicable to some other problem domains. For instance, consider a photo album tagging scenario where the objective is to go over photos in an album and tag them based on the individuals in them. Typically, face recognition is preceded by a face detection step where all the candidate face regions in the photo are identified. This tagging problem has a similar case-instances structure, as a photo corresponds to a case and the candidate face regions in the photo correspond to instances. We note that our case-instance setting appears to be similar to the Multi-Instance Learning (MIL) framework and we provide a fuller comparison in the related work section (Section 5).

When manually labeling examples, it is more efficient to show the user a whole case at a time, such as the example in Table 2, and have them click to identify the keyword query field (if any), than it would be to show each single instance column in isolation and have the user label it as positive or negative. Thus, labeling one case typically generates several labeled instances. In the data we have labeled so far, cases tend to have 4–7 distinct fields (instances), and about 7% of those fields are labeled as being search fields. One reason that there are so many instances on average is that some web pages contain many small forms, each of which has its own set of fields. The number of fields that actually co-occur in any given URL will typically be smaller.

2.2 Machine Learning Formulation

Labeling and/or classification decisions are made at both the case level and the instance level. A labeled case indicates which field(s), if any, are positive, i.e., contain user keyword queries. In the dual view, this entails individually labeled instances, which are positive or negative. We build a binary classifier from the training set composed of all the labeled instances, and then apply it to each of the fields of each case.

In order to apply machine learning to individual instances, we must represent each instance with a feature vector. Although we are dealing with text, the widely successful *bag of words* feature space of text classification is not appropriate here: user keyword queries are generally very short, diverse, and domain-specific at particular web sites. For example, if we were to label the query field name used at http://shopping.hp.com, the query terms the system would learn are likely to be computer-related and not useful to predict query fields at other arbitrary sites, such as http://cats.about.com (except perhaps the word *mouse*). Also, general-purpose search engine sites have very large, non-specific vocabularies rather than a focused set of topic words associated with the positive class, as needed for typical text classification.

Before we describe our features, we give a note on the feature philosophy we used and how it impacts the choice of classifier. Rather than laboriously engineer a few features we supposed might be strongly predictive, our approach was to quickly implement a large set of possibly useful features that were easy to program, and trust that the downstream classifier would be able to leverage the many weak features to produce good accuracy. While this is not a good strategy for building a decision tree classifier, which can only leverage a few features given limited training data, it is a fine strategy

for a (linear) Support Vector Machine classifier, which excels in text classification where there are many weak features. Furthermore, we conditioned the feature space with BNS feature scaling [5] to make the better features have a larger effect on the linear kernel distance function. Later we verified that a variety of other models gave substantially inferior performance.

We constructed a set of statistical features based primarily on the whole column of string values in the instance, plus a pair of features based on the field names. Table 3 describes most of the features we used. Eyeing the sample data in Table 2, consider how many of these features may be predictive but not conclusive. For example, the second column of text will have especially low statistics for the maximum and average number of whitespace characters, as well as an unusually high minimum number of '-' characters; that said, one would expect some level of occurence of '-' in keyword queries, especially as a term prefix. Of our 211 features, all but two of them are generated by just 228 lines of Java code, due to their regular structure. One exception is a feature that estimates the prior probability (with Laplace smoothing) that the current instance is positive based on the count of positive and negative training instances that have the identical field name. For example, if the field name for a test instance is 'q' and 30 positive training instances have that exact field name as well as 4 negatives, then the value of the feature is $(30+1)/(30+4+2) = 86\%$. In Section 4.5 we give an analysis of which features were most predictive.

3. SOFTWARE & EXPERIENCE

In this section, we describe the system used to perform our experiments, the data used and the experimental protocol, and we present the results of our experiments.

3.1 Case Extraction

The first task facing us is the decision of which URLs to group together into cases. The naïve approach of treating URLs as belonging to the same case only when they have identical hostnames and paths would result in fracturing the available data into an enormous number of cases, most of which would have to be discarded as containing too few URLs to be useful and many of which looking indistinguishable from one another to a human labeler. Ignoring the path and grouping together URLs with identical hostnames ameliorates this somewhat, although it still differentiates unnecessarily among different hosts in domains that use consistent conventions throughout, including domains in which load balancing is achieved by means of servers named, e.g., *www-28* and *www-37*. It also introduces the problem of failing to distinguish when different conventions are used for different paths on the same host. Going further and grouping URLs by the host component immediately below administrative domains such as *.com* or *.co.uk* further helps and hurts in the same ways. It is clear that for optimal case blocking, different domains and hostnames need to be treated differently.

Our case extractor begins by obtaining from our 8-month, 2-billion URL sample of the 2008 Nielsen MegaPanel® database all of the URL strings that match the syntax of a query and have at least one keyword/value pair, resulting in approximately 1.2 billion URLs. For each URL, we remove all pairs whose value is excessively long (more than 80 characters), is empty, or which does not contain any alphabetic characters, discarding any URLs that have no remaining fields. Next, whenever the same URL appears more than once in temporal succession for any user, we keep only one copy. This removes many URLs in the data only due to refreshes or due to navigation to subsequent pages within an article. The resulting corpus contains approximately 738 million URLs.

This corpus is then sorted such that URLs from the same top-level domain (e.g., *hp.com* or *bbc.co.uk*) wind up together. For each such top-level domain, a lattice is constructed in which the leaves represent actual host/path combinations in the corpus and the parents of a node represent possible generalizations, obtained by removing either a leftmost component of the hostname or a rightmost component of the path. For each node, we keep track of the total number of URLs associated with dominated leaves as well as counts of the number of times each named query field appears in those URLs. Finally, the lattice is pruned by removing a node's children whenever no child considers a query field "important" if the parent does not also do so, where a field is considered important if it appears in at least 60% of the URLs. The leaves of the resulting pruned lattice are taken to define the cases for that domain. For each case, up to 5,000 randomly-sampled URLs are used to define the features for the case and to present to the user for labeling.

After pruning, the case set contains 228,695 cases that have at least 100 samples of support and which represent 429.3 million samples (58.2% of the total), for an average of 1,877 samples per case. Since the distribution is so skewed, the median is considerably lower, at 230 samples per case. Approximately 3.5% of the cases (7,945) have at least 5,000 samples apiece and account for 44.1% of the total samples in the supported cases. (The top 32 cases each account for at least a million samples apiece, and the top four each have more than ten million, the highest having 33.1 million.) At the other end of the distribution, 28.9% of the cases (66,142) have between 100 and 150 samples apiece. The cases also appear to follow a Zipf-like distribution, with the top 200 cases encompassing 29.5% of the URLs and the next 1,000 cases only covering an additional 5.8% more. The final 100,000 cases with sufficient support only represent 1.9% of the total URLs.

Along with the cases, the data set contains 1,455,059 instances, each representing a uniquely named field in a query associated with a case. This represents 6.4 instances per case on average, but again this distribution is quite skewed, with a maximum of 8,310 instances associated with one case but a median of only 4 instances per case. 406,655 of these cases represent "important" fields, by the measure used in determining cases, and on average, each case had 1.8 important fields.

3.2 Labeling Protocol

Once the cases have been extracted, we can turn to using them to train our classifier. Given the highly skewed distribution we see, it is clearly imperative that the classifier do well on the relatively small number of cases that constitute the large portion of the URLs, and it therefore appears reasonable to expend training effort in explicitly labeling them. We therefore decided that we could take as a baseline that the 200 most frequent cases would be part of the training set (and that any resulting classifier would get them correct) and that the task would be to see how such

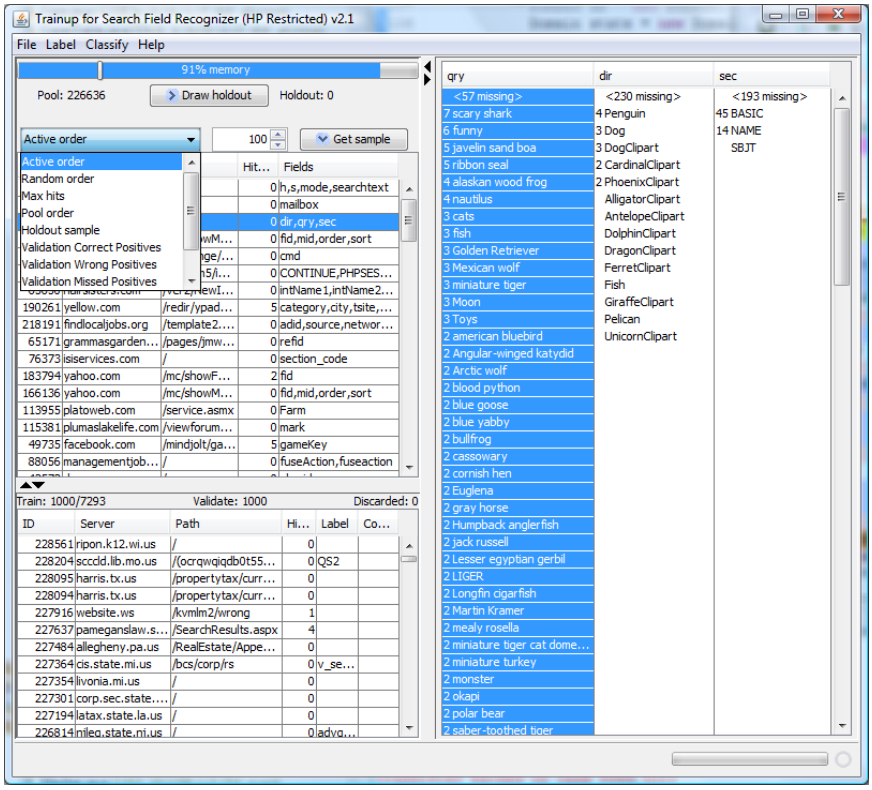


Figure 3: GUI for active learning, labeling, training and holdout-validation.

a baseline classifier could be improved when judged on how well it did on a random sample of the remaining cases.

Preparatory to the experiment, therefore, we labeled the 200 most frequent cases using the system described in Section 3.3 to constitute the initial training set, removing them from the pool of cases. We then labeled 1,000 randomly-drawn cases to constitute the validation set. All performance measurements are made against this test set.

For the first experiment, a further 800 randomly-drawn cases were labeled and added to the training set, bringing it up to 1,000 cases.

For the second experiment, the 800 cases labeled in the first experiment were returned to the unlabeled pool (and their labels erased) and 800 cases were selected via active learning, labeled, and added to the training set, with the classifier being retrained after each case was labeled. The selected case was the one that had an instance whose predicted likelihood of being a search field was closest to the threshold of 0.5, although an anytime algorithm was used, which means that much of the time only some of the 228,000 cases had been rescored before the next case was requested.

For the purposes of the experiment, the positive class was intended to be roughly "a field indicating something arbitrary that the user was searching for on the web." Thus there was a semantic component that ruled out many fields that obviously reflected user-typed input, such as user names, addresses, chat messages, and even some searches, such as white pages lookups. This made the task of the classifier more difficult.

3.3 Classifier Training GUI

To label cases, we use a graphical training prototype written in Java and leveraging the WEKA v3.6.0 machine learning library [7], shown in Figure 3. The table at the bottom left shows labeled cases, with positive instances identified, the table at the top left shows the current pool of cases to label, and the table at the right shows the current case to label.

In this last table, each instance (field) is represented by a column, and the rows in each column represent the distinct values associated with that query field in URLs within the case (or within the 5,000-URL sample for large cases). When a particular value occurs more than once, its multiplicity is shown, and the values in a column are sorted by multiplicity. Thus the labeler can see at a glance whether there are a large number of values that each appear infrequently (often indicative of a hand-typed value) or a relatively small number of values, many of which appear many times apiece.

Further aiding the user, each time a case is displayed, the current classifier is used to sort the fields so that the most-likely-positive fields appear leftmost, with fields that are predicted positive pre-selected. The user clicks on fields to change their labels and then presses a key to record the current labeling and move on to the next case. As the classifier becomes reasonably good with a small number of training cases, this turns largely into an exercise of confirming that it is making the right predictions and making occasional corrections, typically near the boundary between positive and negative predictions. This is a

relatively simple task to perform in most cases and so labeling a case rarely takes more than one or two seconds.

After each new case is labeled, we retrain the classifier with the additional labeled instances and then re-scan the whole pool for the best cases to label. We use the WEKA Support Vector Machine[2], which does not provide for incremental retraining, although empirically, batch learning appears to be reasonably fast typically taking only a few seconds even with training sets in the thousands of instances. (A consequence is that typically the classifier used to sort the columns for the next case to label does not take into account the instances in the most recently labeled case or two. In practice, this does not appear to matter.)

For active learning, on the other hand, scoring the entire 228,000-case pool of candidates can take a substantial amount of time (typically tens of seconds). Rather than delay the user, we maintain a priority queue of the best known cases to label as judged by the most recent classifier. As we scan the pool in random order, we fill this priority queue incrementally, yielding an anytime algorithm that always has something to hand back to the user.

3.4 Evaluation Metrics

Although the classifier we train makes decisions about instances, for our case-centric purposes, the normal instance-based measures of accuracy and precision and the like are inappropriate. If a validation case is labeled as having one positive instance and ten negative instances, and the classifier calls all of them negative, it would be misleading to say that it was 91% accurate. From the case-centric point of view, there was something there to find and it failed to find it.

Designing case-centric evaluation metrics is still an open question, but for the experiments analyzed in this paper, for each case we consider only the instance most strongly indicated by the classifier as being a search field. If this indication exceeds the classifier's threshold and the field is labeled as being a search field, the case is considered to be a 'correct positive' (CP). If the indication does not exceed the threshold and no instance within the case is labeled as being positive, the case is considered to be a 'correct negative' (CN). In either situation, the case is considered to have been correctly classified for purposes of computing accuracy, otherwise it is considered to have been misclassified.

This case-based focus allows there to be three different ways of getting a case wrong. First, an instance may be deemed to be a positive even though the case has no positively-labeled instances (a 'missed negative' or MN). Second, no instance may be deemed to be positive, even though the case has a positively-labeled instance (a 'missed positive' or MP). And finally, the most strongly-indicated instance may have been labeled as negative, but there may be another instance that is labeled as a positive (a 'wrong positive' or WP). This last case could reasonably be considered as either a false negative (the labeled-positive instance was missed) or as a false positive (the instance called positive was labeled negative). For computing precision and recall, we count it against both. The case-centric precision measure is $CP/(CP + MN + WP)$, the number of correct positive cases divided by all cases that the classifier predicts as having one or more search fields. Similarly, the case-centric recall measure is $CP/(CP+MP+WP)$, the number of correct positive cases divided by the number of cases in which there was any search field to find. Case-centric F-measure falls out straightforwardly as the harmonic average of case-centric precision and recall, ($\frac{2PR}{P+R}$).

[2]The classifier is weka.classifiers.functions.SMO with options -V 2 -M, to enable Platt scaling tuned with internal 2-fold cross-validation. It uses the default linear kernel.

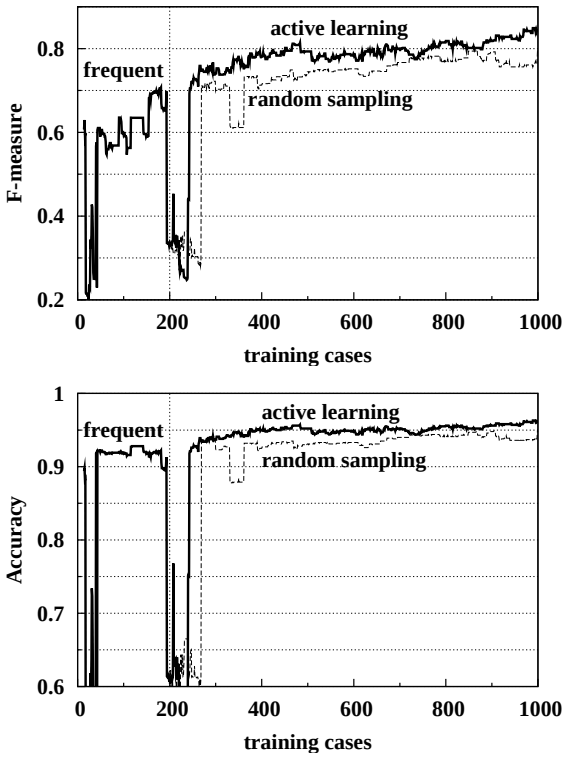


Figure 4: Learning curve by active learning vs. random sampling.

3.5 Results

Figure 4 shows the results of running the experiments described in Section 3.2, with the upper graph showing case-centric F-measure and the lower graph showing case-centric accuracy, both computed over the 1,000 randomly-selected validation cases, as the number of training cases increases. (Note that the y-axis scales are different in the two graphs and that the accuracy curve is cropped at the bottom.) In each graph, the first 200 training cases are the cases in the pool that represent the greatest number of URLs, largest first. Following this, the solid line represents the curve seen when cases are added by active learning, and the dashed line represents the curve seen when cases are added randomly.

Overall, extending the initial training set with 800 cases selected by active learning resulted in a classifier whose F-measure is 0.84 and whose accuracy is 0.96, while extending it instead with 800 cases selected at random resulted in a classifier with F-measure 0.79 and accuracy 0.95. The accuracy numbers are superficially close together because the class prior is highly skewed. F-measure is a preferred metric under imbalance.

One thing to notice is that the curves are far from monotonic over the first 200 training cases, displaying two

"cliffs." The curves begin with 14 cases, when there are enough positive instances to train the classifier, at an F-measure of 0.61 and an accuracy of 0.89. This remains relatively constant until case 18 is added, at which point they drop to 0.22 and 0.17 respectively. They then rise to an F-measure of 0.70 and an accuracy of 0.91 before case 194 is added, at which point they fall to 0.34 and 0.61. By the time the two experiments begin, the baseline classifier they are trying to improve has an F-measure of only 0.33 and an accuracy of only 0.61. Thus the naïve business strategy of simply labeling the most frequent cases may well be a bad idea. Or, rather, whether it is a bad idea may depend on where you stop. Stopping after 193 cases would have produced a classifier that was far superior than one using 200, but, of course, that is only detectable with the hindsight of having labeled a large number of validation cases.

Another thing to notice is that from case 240 onwards,[3] both the accuracy and F-measure for the active learning curve are higher than for the random curve.[4] (The statistical significance of this difference will be discussed in Section 4.1.) At case 240, the active learning F-measure jumps from 0.25 to 0.44, and at case 243, it jumps further to 0.69. The random F-measure starts its recovery with a jump at case 269, when it rises from 0.29 to 0.71, by which point the active learning curve has already risen to 0.75. Another difference between the two curves is that the active learning curve is relatively smooth, while the random curve has another, less severe, cliff at case 331, when the F-measure drops from 0.71 to 0.62 and the accuracy falls from 0.93 to 0.88, recovering at case 362.

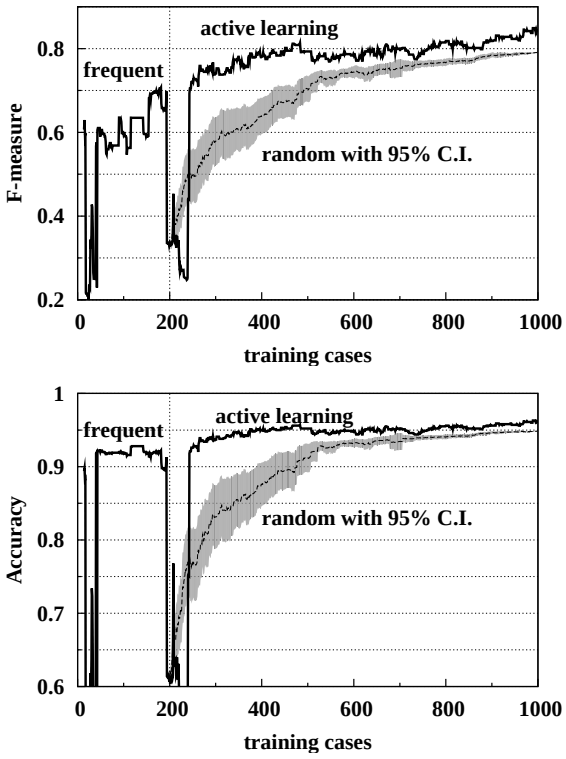


Figure 5: Active learning vs. 95% confidence interval under random sampling.

4. EXPERIMENTAL ANALYSES

In this section we answer a series of performance questions by experiments.

4.1 Statistical Significance

When looking at the performance improvement in the learning curve with active learning vs. random sampling in Figure 4, it is natural to wonder how much of the difference is due to the particular sequence of 800 randomly-drawn cases labeled. The typical method of assessing this question would be to perform many independent runs with different sets of randomly-drawn cases, but unfortunately, this would require many weeks of labeling effort and is infeasible. We can, however, address the similar question of whether the particular *order* in which the randomly-drawn cases were labeled is important. Clearly, the end result with the full 1,000 training cases will be identical, but it might be the case that a different order would have taken a path indistinguishable from, or perhaps superior to, the active learning curve throughout much of its run. To address this, we permute the randomly-selected cases and simulate building classifiers by adding training cases according to the permuted sequence.

Figure 5 shows the average learning curve under random sampling and its 95% confidence interval, computed over thirty such random permutations (always beginning with the initial 200 most frequent cases). Overlaid on this graph is the actual active learning curve we saw previously. From case 243 on, the active learning F-measure and accuracy are strictly above the 95% confidence intervals.[5] So statistically it is very likely that no matter what order the randomly-selected cases were seen in, the active learning strategy would have produced better results for all training set sizes once the active learning strategy had recovered from the poor performance of the initial training set. Indeed, looking at the Figure 5, it is clear that the particular random sequence we observed actually recovered more quickly than could have been expected, and Figure 4 understates the distinction between active learning and random sampling.

4.2 Non-monotonicity

Another question that arises is the degree to which the "cliffs" noted in Section 3.5 are to be expected, as opposed to simply being statistical flukes. In the experiment with randomly generated cases, the F-measure dropped 13.4% from 0.71 to 0.62 at case 331. In the 30 permuted runs described in section 4.1, 20 (67%) had a drop of at least that magnitude somewhere on their learning curves, 10 had at least two such drops, four had at least three such drops, and one had four. So a drop of that magnitude would appear to be expected, at least given the particular set of cases used.

Of the 30 cases, 24 (80%) had an F-measure drop of at least 10%, 10 had a drop of at least 20%, and three had a drop of at least 30%. The largest drop in a permuted run

[3]Note that the specific cases added will be different for the two curves from case 201 on.

[4]The F-measure is always strictly higher. The accuracy of the random curve ranges from 0.001 to 0.004 higher from case 731 through case 737, but otherwise the accuracy for active learning is higher.

[5]Even where they appear to touch on the graph, the active learning value is very slightly above.

ranged from 4.0% to 35.2%, with a mean of 16.9% ± 3.0% and a median of 15.7%.

4.3 Influence of Most Frequent Cases

A further question is whether the first 200 most frequent cases should be excluded from the training set. Considering that the first 200 cases labeled might come from a different distribution than the test set, they may violate the assumptions of machine learning classification and lead to an unfortunate training bias. So we ask whether the learning curve would have been any better without these initial cases. Again, we resolve this by repeatedly permuting the case labeling order under random sampling, but now without the initial 200 most frequent cases labeled. We omit the graph for space considerations, but we found that the average learning curve performance with and without these first 200 frequent cases leads to roughly the same performance mid-way in the curve. So it would seem that these initial 200 frequent cases did not hurt the classifier. But neither did they help as much as 200 additional random cases might have. It is important to bear in mind that the from a business perspective, the test set performance does not account for the overwhelming importance of getting the most frequent cases correct, so whether or not they are added to the training set to develop the long-tail classifier, some mechanism will probably need to be used to ensure that they are handled correctly.

4.4 Comparisons with Baselines

It is also fair to ask whether programmed rules could do as well. A common alternative to a machine learning framework is to write some heuristic rules in software for making the automated decisions. Although it is easy to think up many heuristics, it can be extremely difficult to combine them well and debug them for greater accuracy. Prior to our approaching this whole problem, a programmer in our organization had already attempted the heuristic approach and ended up manually vetting and correcting many of its decisions for just the top 209 popular websites. So that we may have an apples-to-apples performance comparison with our system, we tested his heuristic rules on our 1000 validation cases. The rules achieved an accuracy of 0.87 on the cases and an F-measure of 0.44 (with a precision of 0.46 and a recall of 0.44).

We also implemented another natural rule that basically labels an instance as positive if the field name of the instance matches with any of the search field names corresponding to the labeled first 200 most frequent cases. This rule achieved an accuracy of 0.92 on the cases with an F-measure of 0.60 (with a precision of 0.69 and and recall of 0.53).

Thus, the machine learning approach with active learning does offer substantially superior performance, resulting in a classifier with an accuracy of 0.96 and an F-measure of 0.84.

4.5 Evaluation of Features

Recall that our approach for feature construction was mostly to generate many features that were easy to program, hoping the machine learning classifier could combine them successfully to produce an accurate classifier. This raises the question of which of the many features were actually useful. Table 4 shows the most successful features measured in the 3190 instances in the validation set according to the WEKA implementation of information gain (which involves a *minimum description length* discretization for each of our numerical features). Some of the strongest features were the average, maximum, and standard deviation of the number of whitespace characters in the column of strings. The third strongest feature is the prior on the field name of the instance. Generally, many of these top features have to do with the standard deviation of the count of some character class, indicating that heterogeneity in many aspects is characteristic of keyword search fields. By contrast, many of the least useful features (not shown) involved the maximum or minimum count of a special character or character class.

Table 4: Top predictive features by information gain

IG	Feature
0.1589	avg.isWhitespace
0.1585	stdev.isWhitespace
0.1491	prior on field name
0.1474	max.isWhitespace
0.1167	stdev.length
0.1151	stdev.lowercase
0.1146	stdev.alphanum_
0.1121	stdev.nonhex
0.1107	stdev.isLetter
0.0972	pct.lowercase
0.0907	prior given all case's field names
0.0884	max.lowercase
0.0819	pct.lower and uppercase
0.0784	max.nonhex
0.0782	pct.unique samples
0.0779	avg.wordsCapitalized
0.0778	max.isLetter
0.0790	stdev.wordLength
0.0747	pct.unique values
0.0736	stdev.hex
0.0725	stdev.words
0.0724	avg.uppercase
0.0696	avg.words
0.0693	pct.uppercase
0.0669	avg.nonhex
...	

5. RELATED WORK

Active learning with pool-based sample selection [8] has been used extensively in the supervised learning setting when there is a shortage of labeled data and a cost associated with acquiring labels. The literature on active learning is concentrated on two types of methods: committee-based methods [13, 10] and confidence-based methods [8, 14]. In committee-based methods, multiple hypotheses are learned and instances that lead to maximum disagreement in the committee are picked for labeling. In confidence-based methods, the confidence score of learned classifiers are used to determine the most uncertain instances and hence, considered to be the most informative ones. Implementing committee-based methods involves learning/updating all the classifiers after every round of labeling and having each classifier score every unlabeled instance to pick the next set of instances for labeling. In practice, this could affect the real-time performance of the labeling software. By using uncertainty sampling, having only a single classifier, the time to retrain and scan the entire pool rarely exceeded 30 seconds.

Active learning has been used for acquiring labels in several areas such as text classification [14], image/video classification [1, 15], speech processing [6] and information retrieval[16]. Discriminative methods like SVM-based active learning [14] have traditionally outperformed other methods in text-related domains and we also observed that for our task, SVMs outperformed several other base learners in cross-validation experiments not presented.

In Section 2, we introduced the distinction between cases (websites) and instances (fields of the website) in our setting and such a case-instance formulation is different from the traditional multi-instance learning (MIL) setting [3]. In the MIL setting, there are bags containing instances, and labels are assigned at the bag-level both during training and testing. The individual instances in a bag are not labeled, and a positive bag contains at least one positive instance and several negative instances. In our case-instance setting, the learning problem is in identifying the label of every instance, i.e., whether a query field is a search field or not. There is some recent work on multiple-instance active learning [12, 9] in which the authors develop active learning strategies for picking specific instances from positive bags for labeling towards improving the overall bag classifier.

There are other related fields like information extraction [2] in which the task is to automatically extract relevant information such as named entities from unstructured data and database schema matching [11] (especially, instance level matchers) which involves identifying specific targets such as zip-code/employee names. Our problem is different from these settings as our target concept is loosely defined and very diverse, as it could range from different shoe types to movies to printer models.

6. CONCLUSION & FUTURE WORK

In order to use this recognizer in production, we first apply the classifier offline to all cases. From this, it records all known keyword search fields with their associated website. Matching this against millions or billions of clickstream URLs is potentially a slow process if based on string-matching. We have instead developed an efficient hash-based recognizer, based on the ideas in [4].

The upshot of this research is that it is practical to accurately identify the keyword searches of sample users all across the web. This enables a wide variety of analyses on user search behavior, once primarily able to be conducted only with private search logs at Google or AOL. Furthermore, the classifier could be trained to look for other kinds of user input. For example, the recognizer could be trained to look for mailing addresses, which would be potentially useful for a catalog mail order company looking to partner with websites that regularly obtain people's mailing addresses. Such an analysis would not be possible from any single site's server logs.

While active learning for individual instances is well researched, active learning under the case-instances structure has not. We have begun to research some possibilities here, and we believe this area should have broader applicability.

7. REFERENCES

[1] B. Collins, J. Deng, K. Li, and L. Fei-Fei. Towards scalable dataset construction: An active learning approach. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 86–98, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] J. Cowie and W. Lehnert. Information extraction. *Commun. ACM*, 39(1):80–91, 1996.

[3] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.

[4] G. Forman and E. Kirshenbaum. Extremely fast text feature extraction for classification and indexing. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)*, pages 1221–1230, New York, NY, 2008. ACM.

[5] G. Forman, M. Scholz, and S. Rajaram. Feature shaping for linear SVM classifiers. In *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 299–308, New York, NY, USA, 2009. ACM.

[6] D. Hakkani-Tür, G. Riccardi, and G. Tur. An active approach to spoken language processing. *ACM Trans. Speech Lang. Process.*, 3(3):1–31, 2006.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[8] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1994.

[9] D. Liu, X.-S. Hua, L. Yang, and H.-J. Zhang. Multiple-instance active learning for image categorization. In *MMM '09: Proceedings of the 15th International Multimedia Modeling Conference on Advances in Multimedia Modeling*, pages 239–249, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] P. Melville and R. J. Mooney. Diverse ensembles for active learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 74, New York, NY, USA, 2004. ACM.

[11] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[12] B. Settles, M. Craven, and S. Ray. Multiple instance active learning. *NIPS*, 2007.

[13] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, New York, NY, USA, 1992. ACM.

[14] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, 2002.

[15] R. Yan, J. Yang, and A. Hauptmann. Automatically labeling video data using multi-class active learning. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 516, Washington, DC, USA, 2003. IEEE Computer Society.

[16] C. Zhang and T. Chen. An active learning framework for content-based information retrieval. *Multimedia, IEEE Transactions on*, 4(2):260–268, Jun 2002.