Athena: Mining-based Interactive Management of Text Databases

Rakesh Agrawal, Roberto Bayardo, and Ramakrishnan Srikant

IBM Almaden Research Center
San Jose, CA, U.S.A.
ragrawal@acm.org, bayardo@alum.mit.edu, srikant@us.ibm.com

Abstract. We describe Athena: a system for creating, exploiting, and maintaining a hierarchy of textual documents through interactive mining-based operations. Requirements of any such system include speed and minimal end-user effort. Athena satisfies these requirements through linear-time classification and clustering engines which are applied interactively to speed the development of accurate models.

Naive Bayes classifiers are recognized to be among the best for classifying text. We show that our specialization of the Naive Bayes classifier is considerably more accurate (7 to 29% absolute increase in accuracy) than a standard implementation. Our enhancements include using Lidstone's law of succession instead of Laplace's law, under-weighting long documents, and over-weighting author and subject.

We also present a new interactive clustering algorithm, C-Evolve, for topic discovery. C-Evolve first finds highly accurate cluster digests (partial clusters), gets user feedback to merge and correct these digests, and then uses the classification algorithm to complete the partitioning of the data. By allowing this interactivity in the clustering process, C-Evolve achieves considerably higher clustering accuracy (10 to 20% absolute increase in our experiments) than the popular K-Means and agglomerative clustering methods.

1 Introduction

People and organizations are amassing more and more data in the form of unstructured or semi-structured textual documents, due mostly to the flourishing of e-mail and world-wide web usage. Systems that manage text databases typically provide support for keyword queries and manual arrangement of documents into a hierarchical ("folder") structure. Benefits of a folder hierarchy include the ability to quickly locate a document without having to remember the exact keywords contained in that document, and the ability to easily browse a set of related documents. Unfortunately, the task of maintaining coherent folders is time consuming, and often unused or under-used as a result. For example, many if not most e-mail users simply allow their messages to accumulate in the inbox where these messages remain until they are deleted. As a result, the benefits of a well-maintained folder hierarchy are never realized. Outside the e-mail context,

web search engines such as Yahoo! [Yah] and several e-commerce sites with large product catalogs provide and maintain hierarchical categorizations to enhance the usability of search results and to facilitate browsing.

In this paper, we present a system henceforth referred to as Athena[1] that simplifies the process of hierarchical document organization through text mining algorithms. Athena is currently implemented on top of Lotus Notes [Lot] to support the management of e-mail and discussion databases, though the concepts apply equally well to other document repositories including collections of web pages or text files. Important functionality of Athena includes:

- *Topic Discovery*: Decompose an unorganized collection of documents into groups so that each group consists of documents on the same topic.
- *Hierarchy Reorganization*: Reorganize a hierarchical collection of documents into another hierarchy.
- *Hierarchy Population and Inbox Browsing*: Use the information contained in the current populated hierarchy to organize documents in Inbox by topic and/or automatically route new documents. Documents that fit in multiple categories can optionally be routed to (or displayed under) several categories.
- *Hierarchy Maintenance*: Identify misfiled documents in a node of the hierarchy and detect concept drift within a node.

A more detailed description of the Athena functionality is given in [ABS99].

Classification and clustering are the two basic building blocks used for implementing the above functionality. These technologies have been studied extensively in the areas of statistics, data-mining, information retrieval, and machine learning. However, standard techniques performed poorly for our application. For example, Naive Bayes classifiers [Goo65] are recognized to be among the best for classifying text. We found that by specializing Naive Bayes for our application, we could reduce the number of errors by 30 to 60% (with 7 to 29% absolute increase in accuracy) when compared to a standard implementation. Our enhancements include using Lidstone's law of succession instead of Laplace's law, under-weighting long documents, and weighting author and subject.

Clustering algorithms applied in this domain must be fast to avoid taxing the patience of the user. We present a new linear-time interactive clustering algorithm, C-Evolve, for topic discovery. C-Evolve first finds highly accurate cluster digests (partial clusters), gets user feedback to merge/correct these digests, and then uses the classification algorithm to complete the partitioning of the data. By allowing interactivity in the clustering process, C-Evolve is considerably more accurate (10 to 20% absolute increase in accuracy in our experiments) than K-Means [Ras91] as well as agglomerative clustering methods [Ras91].

Paper Layout In the rest of this section, we discuss related work on systems for routing or clustering email or text documents. (We discuss related work in

[1] Automated Text HiErarchy maNAgement

clustering or classification algorithms in Sections 3 and 4.) We describe the technical details of the classification component in Section 2, along with experimental results justifying our design decisions. Section 3 goes into the technical details of the clustering component, and also provides an empirical evaluation of its efficacy. Section 4 concludes with a summary and directions for future work.

Related Work There have been at least three previous proposals on the development of classification models for the purpose of routing e-mail, either in general [Coh96] [SK99] or for the special case of junk-mail [SDHH98]. Other systems provide agents that assist e-mail users by predicting an action the user is likely to take [Mae94] [PE97]. None of these proposals address the task of textual database organization outside of routing incoming documents.

Other related work includes the Scatter/Gather system [CDPT92], which uses on-line clustering to assist the user in browsing large collections of documents. While Scatter/Gather does not directly address the problem of creating and maintaining a lasting document organization, many of the requirements of our systems are similar, such as the need for fast, on-line algorithms capable of extracting useful information from text, and the need to involve the user in the process of applying these algorithms. We make a more detailed comparison of our system to Scatter/Gather in a later section. SONIA [SYB98] uses agglomerative text clustering to organize the results of queries to networked information sources, and Naive Bayes classification to organize new documents within an existing categorization scheme.

2 Classification Component

Athena's classification component is used for hierarchy reorganization, document routing, and identification of misfiled documents. We decided to base our classifier on the Naive-Bayes model [Goo65] for the following reasons:

- Naive-Bayes classifiers are very competitive with other techniques for text classification [CDAR97] [LR94] [Lan95] [PB97] [MN98].[2]
- They stabilize quickly [Koh96], which supports automated hierarchy reorganization with a limited number of examples.
- They are fast. They can be constructed quickly with a single pass over the documents, making them suitable for on-line model creation; they also quickly classify incoming documents [CDAR97].
- They are simple to update in the presence of document additions or deletions, making them easy to maintain.

[2] We also experimented with using the SPRINT decision-tree classifier [SAM96], but found it had low accuracy in this domain due to the small number of examples per class and the large feature space.

The basic Naive-Bayes classifier estimates the posterior probability of class C_i given a document d via Bayes' rule:

$$\Pr(C_i|d) = \frac{\Pr(d|C_i) \times \Pr(C_i)}{\Pr(d)}$$

We ignore $\Pr(d)$ since it is the same for all classes. $\Pr(C_i)$ is estimated by:

$$\Pr(C_i) = \frac{\text{Number of documents in class } C_i}{\text{Total number of documents in the dataset}}$$

To estimate $\Pr(d|C_i)$, we "naively" assume that all the words in the document occur independently to get

$$\Pr(d|C_i) = \prod_{w \in d} \Pr(w|C_i)$$

Let $n(C_i, w)$ be the number of occurrences of word w in class C_i (counting multiple occurrences), and $n(C_i) = \sum_w n(C_i, w)$ the total number of words in class C_i. Then the maximum likelihood estimate for $\Pr(w|C_i)$ is simply $n(C_i, w)/n(C_i)$. However, using this estimate would give a probability of zero for any word that does not occur in the class, and thus result in $Pr(d|C_i)$ being zero for any document d that contained a word not present in class C_i. The standard approach to address this problem (e.g. [CDAR97] [Mit97] [KMST98]) is to smooth the maximum likelihood estimate with Laplace's law of succession [Goo65] to get

$$\Pr(w|C_i) = \frac{n(C_i, w) + 1}{n(C_i) + |V|} \tag{1}$$

where $|V|$ is the size of the vocabulary (i.e., the number of distinct words in the dataset). The above formula is the result of assuming that all possible words are *a priori* equally likely (see [Ris95] for details).

Following [MN98], we use the multinomial form of the Naive Bayes classifier where each document is treated as a bag of words rather than a set of words to yield better accuracy.

2.1 Enhancing the Naive-Bayes Classifier

We first specialized the standard Naive-Bayes classifier by applying techniques such as under-weighting long documents and over-weighting author and subject [ADW94]. To over-weight subject (author) by w times, we treated the subject (author) as if each of the words in the subject (author) had occurred w times rather than just once. To prevent long documents from dominating the folder profile, we under-weighted documents with more then l words (for some threshold l) by treating each word as if it had occured only $l/n(d)$ times, where $n(d)$ is the number of words in the document. The weights and thresholds were determined by using a part of the training set as a validation set. These enhancements cumulatively yielded upto 3% accuracy improvements in our experiments.

	Small Class	Large Class
Vocabulary	10,000	10,000
Number of Words	1000	100,000
Number of occurrences of word w	10	1000
$\Pr(w\|c)$ w/o correction	1%	1%
$\Pr(w\|c)$ with correction	0.1%	0.91%

Fig. 1. Skew due to Laplace Correction

In search for ideas for larger improvements in accuracy, we examined the words that had the maximum impact on the classification of a document. These words can be found by looking at the ratio of $\Pr(w|c)/\Pr(w|c')$, where c is the class Athena chose and c' the class for which $\Pr(w|c')$ is highest. Investigation of some misclassified documents revealed that the probability estimate of some words was being highly skewed by the Laplace correction. Figure 1 explains this skew with an example. Word w has a maximum likelihood estimate of 1% in both classes. However, after applying the Laplace correction, it is considered 9 times more likely to appear in the large class as in the small class. Hence we suspected that the Laplace correction was creating a strong bias towards larger classes.

Lidstone's law of succession We then replaced Laplace's law of succession with Lidstone's law of succession. For positive λ, we estimate $\Pr(w|C_i)$ to be

$$\Pr(w|C_i) = \frac{n(C_i, w) + \lambda}{n(C_i) + \lambda|V|} \tag{2}$$

This class of probability estimates is due to the actuaries G.F. Hardy [Har89] and G.J. Lidstone [Lid20] at the turn of the century. The above estimate is a linear interpolation of the maximum likelihood estimate $n(C_i, w)/n(C_i)$ and the uniform prior $1/|V|$. This can be seen by rewriting (2) with the substitution $\mu = n(C_i)/(n(C_i) + \lambda|V|)$:

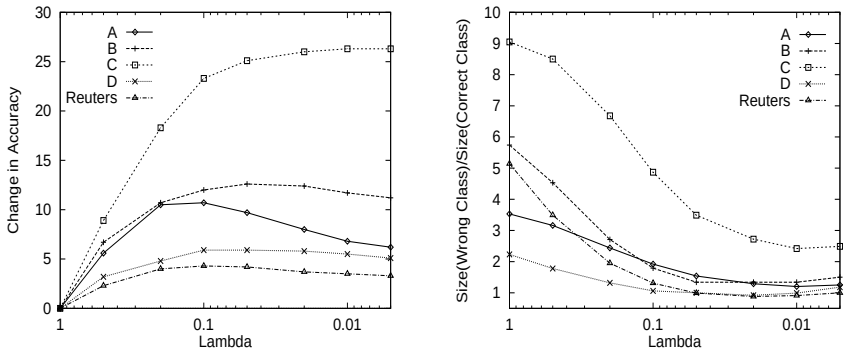
$$\Pr(w|C_i) = \mu\frac{n(C_i, w)}{n(C_i)} + (1 - \mu)\frac{1}{|V|}$$

Note that if $\lambda = 1$, the Lidstone correction is identical to the Laplace correction. In their attempt to improve the accuracy of Naive Bayes, Kohavi et al [KBS97] also experimented with Lidstone correction using the datasets available in the UCI repository.None of these datasets consists of textual data. They found that using $\lambda = 1/\text{total records}$, they could slightly reduce the average absolute error (by 1%, from 19.59% to 18.58%) compared to Laplace correction.

Empirical Evaluation We ran experiments on the datasets shown in Table 1. A, B, C and D refer to four email datasets, while Reuters is the single-category

Dataset	#Folders	#Docs	#Docs/Folder Avg ± Std. Dev.	#Words/Folder Avg ± Std. Dev.	Vocabulary
A	53	768	14 ± 25	4K ± 4K	15K
B	38	1,039	25 ± 34	10K ± 17K	23K
C	204	2,995	15 ± 23	5K ± 9K	39K
D	15	964	64 ± 47	18K ± 13K	15K
Reuters	82	11,367	138 ± 517	15K ± 36K	24K

Table 1. Dataset Characteristics



(a) Improvement in Accuracy (b) Size(Wrong Class)/Size(Real Class)

Fig. 2. Lidstone Correction

version of Distribution 1.0 of Reuters-21578[3]. Notice that three of the four email datasets have a very large number of classes, ranging from 38 to 204, and very few examples per class, ranging from 25 to just 14 documents. All our accuracy numbers were computed using 10-fold cross-validation.

Figure 2(a) shows the change in accuracy as we vary λ in the Lidstone correction, relative to the accuracy for the Laplace correction ($\lambda = 1$). The accuracy improved by 4% to 26%, with the largest gains on those datasets where the accuracy numbers were low. The number of errors was reduced by between 25% and 60% on these datasets (details given later in Table 2). The optimal value of λ varied between 0.2 and 0.01, depending on the dataset.

Recall our hypothesis that the reason for the improvement in accuracy due to decreasing λ is the bias towards large classes produced by the Laplace correction. To verify this hypothesis, we need to measure the size bias of the classifier. For any misclassified documents d, let C_m be the misclassified class and C_r the right class. We use the geometric mean of $n(C_m)/n(C_r)$ over all misclassified documents as a measure of the size bias of the classifier. Figure 2(b) shows that the size bias decreases dramatically with decreasing λ, confirming our hypothesis. At $\lambda = 1$, the misclassified class is between 2 and 9 times larger than the right class. At the optimal λ values for the datasets, the ratio is between 1 and 2.5.

[3] Available from http://www.research.att.com/ lewis

Dataset	Number of Classes	Standard	Lidstone	Final	Reduction in Errors	Top 3 classes
A	53	**60.5** ± 5.7	71.5 ± 6.3	**72.4** ± 5.4	30%	**84.4** ± 5.5
B	38	**66.2** ± 5.4	84.8 ± 1.7	**86.9** ± 2.4	61%	**95.1** ± 2.0
C	204	**47.0** ± 3.0	74.4 ± 1.9	**76.1** ± 2.3	55%	**88.1** ± 2.7
D	15	**81.6** ± 4.2	88.6 ± 3.2	**88.9** ± 1.8	40%	**98.1** ± 1.6
Reuters	82	**79.9** ± 1.3	84.8 ± 1.0	**85.2** ± 1.2	26%	**94.8** ± 0.7

Table 2. Accuracy ± Standard Deviation

Note that dataset C, for which decreasing λ gave the most dramatic gains, also has the largest change in the size bias, decreasing from 9 to 2.5.

Given the sensitivity of accuracy to the value of λ, Athena uses an automatic procedure to select the optimal value of λ on a per-dataset basis. We use a portion of the training set as a validation set, and compute the accuracy of the classifier over this validation set for various values of λ to obtain the optimal value.

Recent work by McCallum et al. [MRMN98] uses both the uniform prior and a global prior (the frequency distribution over the entire dataset) to smooth the maximum likelihood estimate. Their algorithm uses expectation maximization to assign weights to each of these distributions on a per-class basis. This procedure typically requires a dozen or so iterations over the data, which can be expensive.

Summary of Results Table 2 shows the accuracy of the classifier for several versions:

- *Standard*: Standard Naive Bayes classifier with Laplace correction
- *Lidstone*: Classifier using Lidstone correction instead of Laplace correction, with the optimal value of λ determined automatically using part of the training set as a validation set.
- *Final*: Classifier using Lidstone correction that additionally over-weights author and subject, and under-weights long documents.
- *Reduction in Errors* between the Standard version and the Final version.
- *Top 3 classes*: Accuracy of the classifier when the classifier is judged correct if any of the top 3 choices were correct.

We observe large gains in accuracy due to Lidstone's correction, and small (but useful) gains in accuracy due to over-weighting author and subject, and under-weighting long documents. The increase in absolute accuracy ranged from 7 to 29% for the email datasets, with a 30 to 61% reduction in the number of errors. We also note that the classifier accuracy is significantly higher when considering the top three classes. These accuracies can be achieved by applications which make multiple recommendations for document routing (e.g. [SK99]), allowing the user to make the final selection. Additionally, in this application domain, many documents naturally belong to multiple classes. Multiple classifications of a document can be useful for informing the user of this possibility.

3 Clustering Component

The clustering component is responsible for topic discovery within an unorganized collection of documents. We have developed an interactive approach to clustering which involves iteratively presenting the user with a perusable number of related documents that suggest how a specified folder might be decomposed. We call such a set of documents a cluster digest. This term is borrowed from the Scatter/Gather system [CDPT92], which uses on-line text clustering to assist the user in browsing a document collection. In Scatter/Gather, cluster digests are presented to the user to determine which documents in the collection are worth browsing. Unlike Scatter/Gather, Athena does not produce a complete clustering before forming the cluster digests. Instead, it applies a novel algorithm that produces the digests directly. This results in response times suitable for on-line application, with equivalent or better results. The algorithm can also produce these digests incrementally, allowing the incorporation of feedback from the user into the clustering model before producing additional results, and avoiding any need for the user to specify a desired number of clusters apriori.

There are multiple proposals for text clustering, with many of the most popular and effective belonging to the agglomerative class [Ras91]. A drawback of this class of algorithms is they are at best of quadratic runtime complexity, limiting their usefulness in on-line applications such as ours. To overcome this limitation, Scatter/Gather uses a complex intermixing of iterative partitional clustering algorithms (e.g. K-Means) and an agglomerative scheme. The basic idea is to apply the inferior but fast partitional clustering algorithm until the document set has been decomposed to a point where agglomerative clustering can be used efficiently. Because we found that a complete clustering typically contains too many errors to be useful for reorganizing a document collection, we instead opted to develop a new algorithm that very quickly produces only the cluster digests for topic discovery, and leaves further document partitioning to the classifier. Our experiments reveal that this approach is considerably more accurate.

Our algorithm can be thought of as "evolving" one cluster digest at a time. After evolving a new digest, it is presented to the user who can discard irrelevant documents, move documents between the digests, and even discard a digest entirely. After these modifications are made, additional digests can be mined from the remaining documents at the user's request. In implementing clustering in this manner, the user need not specify up-front an exact number of clusters to discover.

3.1 Algorithm Details

The pseudo-code for the digest evolution routine appears in Figure 4. We call the algorithm C-Evolve(n), where the variable n specifies the number of cluster digests to evolve simultaneously. (Only the best of this set is returned.) This algorithm is called once for each cluster desired by the user.

Input variables:

A set D of input documents.

A set P of previously-discovered cluster digests (possibly empty).

An integer n specifying the number of clusters to evolve simultaneously.

An integer DIGEST_SIZE bounding the number of documents in a cluster.

Return value:

A cluster digest consisting of at most DIGEST_SIZE documents from D.

Score Function: See algorithm description.

Algorithm:

1. Remove documents within the digests of P from D.
2. Initialize a cluster C_1 with a random document from D.
3. For $i = 2 \ldots n$
   Initialize a cluster C_i with the document d in D that minimizes the following:
   $\max(\forall_{j=1 \ldots i-1}\ \mathrm{sim}(d, C_j))$
4. For $i = 1 \ldots n$
   Let cluster C_i' contain the top $\min(|C_i| + 1, \text{DIGEST\_SIZE})$ scoring documents d according to $\mathrm{score}(d, C_i, P)$
5. If there exists some $i = 1 \ldots n$ such that $C_i \neq C_i'$,
   (a) $C_i := C_i'$ for all $i = 1 \ldots n$
   (b) goto step 4.
6. Return the cluster C_i out of $C_1 \ldots C_n$ that maximizes the following:
   $\min(\forall_{d \in C_i}\ \mathrm{score}(d, C_i, P))$

Fig. 3. The C-Evolve(n) algorithm for evolving a cluster digest.

The pseudo-code makes use of a similarity function, *sim*(), that returns a floating point value between 0 and 1 (with 1 indicating maximum similarity). This function must accept either a single document or a set of documents for each of its two arguments. In our experiments, we use the Scatter/Gather approach for computing similarities: Similarity of a document pair is given by cosine distance between vectors of term frequencies, with term frequencies damped by the square root function. Similarity of two clusters is then given by the average pair-wise similarity between their documents.

Another function used by the pseudo-code, *score*(), accepts a document, a cluster digest C, and a set of cluster digests P. The return value of this function is maximized when the document is highly similar to cluster C, and highly dissimilar to those clusters within P. We use the method below for computing such a value, though C-Evolve can use other scoring functions with these properties.

$$\mathrm{score}(d, C, P) = \begin{cases} \min(\ \forall_{C_p \in P}[\mathrm{sim}(d, C) - \mathrm{sim}(d, C_p)]\ ) & \text{if } P \text{ non-empty,} \\ \mathrm{sim}(d, C) & \text{otherwise} \end{cases}$$

The first three steps of the algorithm perform initialization, which involves removing documents from the input set that already belong to previously-discovered cluster digests, and seeding the n clusters with documents that are dissimilar to each other. This seeding policy spreads the search for a good cluster digest across the document space.

Step 4 grows the clusters one document at a time until the maximum size is reached and clusters stop changing. By virtue of the scoring function, this step not only attempts to maximize similarity between documents within a given cluster, but also dissimilarity between the evolving cluster and previously discovered clusters. This ensures that, even though the algorithm is not performing a complete partitioning of the input documents, the evolving digests are likely to discover a topic that is not already represented by the existing ones. Note that this scheme is not entirely greedy – that is, a document added to the cluster may disappear from it in a later iteration. Even though a document may score highly during an early iteration, as the cluster evolves it may no longer score so highly. This feature ensures such documents are removed to improve the final result.

Athena employs C-Evolve with the parameter n set to 6. This setting was chosen because it yielded good results, and it is small enough to guarantee response times of a few seconds given a document collection of a thousand documents, a cluster digest size of 10-20, and a Java based implementation.

There is no simple way to obtain a tight bound on the iterations performed by C-Evolve(n) since the clusters can continue to evolve after reaching the desired size. In practice, however, the number of iterations performed after the cluster digest has reached the desired size is typically small, usually well under 5. This amounts to a linear average-case complexity assuming the cluster digest size and the value of n are bounded by constants. If desired, the number of iterations can be hard-bounded by a small constant to guarantee linear complexity in the worst-case. If the user desires a larger cluster digest, e.g. one that is proportional to the collection size instead of bounded by a constant, the procedure can be modified to grow the cluster more than a single document at a time. We have found that slow cluster growth is most helpful during the early iterations of the procedure, so growth can be accelerated during later iterations in order to guarantee linear complexity without significantly compromising the results.

3.2 Evaluation

In this section, we compare C-Evolve with more traditional clustering methods including K-Means and Hierarchical Agglomerative Clustering [Ras91]. Our first suite of experiments evaluates how well C-Evolve performs when finding cluster digests compared to these other (appropriately modified) techniques. Because agglomerative and K-Means clustering fully partition the input documents, in order to produce cluster digests, we modified them to return the most central documents of each cluster. The second suite of experiments evaluates how well C-Evolve, when applied interactively with the classifier from the previous section, compares to these other clustering methods when fully partitioning the input documents.

Agglomerative clustering algorithms work by placing each document in its own cluster, and then iteratively merging the pair of most similar clusters until

the desired number of clusters remain. In our implementation, cluster similarity is given by average pair-wise similarity between the documents from each cluster. We use the same document similarity function (cosine distance) and document representation (term frequencies damped by the square-root function) for all clustering algorithms to ensure a fair comparison. This algorithm is exactly the "reference" clustering algorithm used in Scatter/Gather [CDPT92].

K-Means clustering seeds each initial cluster (the number of which equals the number of desired clusters) with a single randomly chosen document. A pass is made over the input documents and each document is placed moved into the cluster that is most similar to it. This process repeats until the clusters stop changing, at which point they are returned.

We used e-mail collections to evaluate these algorithms, obtained from co-workers who diligently file their e-mail, along with the standard Reuters-21578 benchmark data[4] in order to show the results apply more generally. (Table 1 shows some of the characteristics of these datasets.) For each data-set, we selected at most 100 documents from each folder/category to prevent folders with many documents (e.g. those compiled from an active mailing list) from excessively slowing the agglomerative algorithms[5] and overly skewing the input distribution. For the Reuters data, each document was placed into a "folder" corresponding to the first topic to which it is assigned.

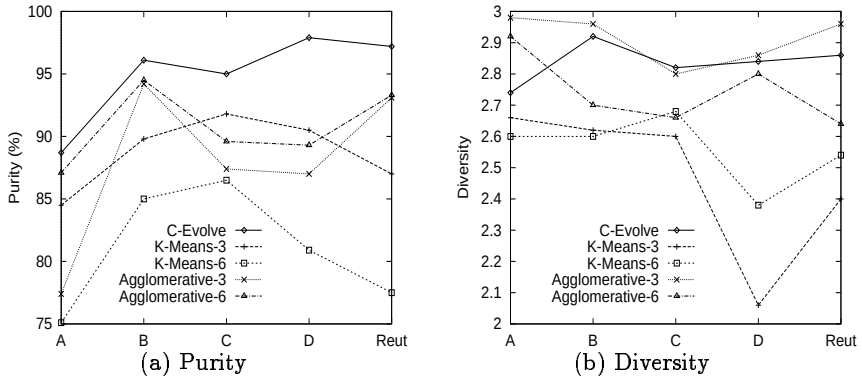
Evaluating Topic Discovery We assume that the organization provided by the folder hierarchy from each data-set is the "true" clustering of the data. For each data-set, fifty trials were performed, where for each trial, six folders were selected at random from a given data-set and the documents intermixed. Each clustering algorithm was then run on the resulting document collection and made to identify three cluster digests containing 11 documents each.

We chose to identify a smaller number of cluster digests than the number of true clusters in order to determine the sensitivity of agglomerative and K-Means clustering to the number of clusters identified – in real world applications, the true number of clusters is unknown, so the number of clusters the user chooses to identify is unlikely to be the same as the true number of clusters. To demonstrate that knowledge of the true number of clusters does not necessarily improve the results, we also have agglomerative and K-Means clustering identify six cluster digests and return the best three of the result. (The best cluster digests are those which maximize average pair-wise similarity).

For each set of cluster digests provided by an algorithm for a given run, we compute two "goodness" metrics. The first is digest purity: we want each digest to contain only documents from a single one of the true clusters. Purity of a digest is therefore defined as the maximum number of documents within

[4] Available at http://www.research.att.com/ lewis/reuters21578.html

[5] Experiments were run by e-mail database owners on their personal machines in order to maintain their privacy. This required our experiments to complete over a single evening so that their machines were free for regular work during the day.



(a) Purity (b) Diversity

Fig. 4. Purity and Diversity Results for the Cluster Digests

the digest that belong to the same true cluster, divided by the total number of documents within the digest. Another metric is digest diversity: we want each digest to present a topic that is distinct from the others. Cluster diversity is therefore the number of true clusters covered by the dominant class from each cluster digest. This implies the maximum diversity score is 3 since we apply the algorithms to find only 3 digests in this experiment.

The results of the experiment appear in Figure 4. As can be seen, C-Evolve is superior to all other schemes with respect to the purity metric. With respect to diversity, it is superior to all but the agglomerative algorithm which identifies three clusters before returning the digests of each. Surprisingly, the algorithms which attempt to identify the true number of clusters do not fare any better than their counterparts which identify only three clusters.

Evaluating Interactive Clustering After a set of topics is discovered, Athena allows the user to populate these topics with additional documents by applying the classifier from the previous section. We evaluate how well this technique works when attempting to fully recover the true clustering by completely partitioning the input documents. For these experiments, we select 5 folders at random from each data-set and intermix the documents to form the algorithm input. Once again, fifty trials are performed on each data-set and the results averaged.

For this experiment, we have the K-Means and Agglomerative clustering algorithms attempt to identify the true number of clusters (5 in this case). For each cluster identified, each document is considered an "error" if its class does not match the dominant class of the cluster. Following [SYB98], we compute the accuracy of a given clustering as one minus the error rate.

We apply C-Evolve repeatedly until each folder is discovered (a folder is said to be discovered by a digest if the dominant class of the digest matches that of the folder). Typically, the number of digests identified in order to discover all folders exceeds the true number of clusters by only one or two. We compute errors

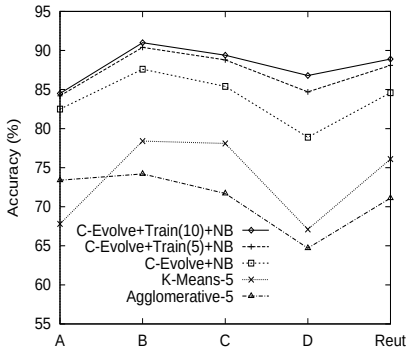


Fig. 5. Accuracy Results

for this approach by first summing up the total number of documents in each digest that do not belong to the dominant class of the digest. This error value is then added to the number of errors produced by the classifier in populating the clusters with the remaining documents. To simulate user-corrections, before applying the classifier, we remove any errors from the cluster digests and merge cluster digests that discover the same folder. We also evaluate the effect of having the system present the user with a small number of randomly-selected examples for manual correction of Athena's classification before the classifier is applied in order to broaden the training set. We tried both 5 documents per folder and 10 documents per folder. Each document presented to the user by Athena is counted as an error if the suggested class is not the folder to which the document actually belongs.

As can be seen in Figure 5, our interactive approach to clustering leads to substantial improvements in accuracy over standard clustering techniques. Manual correction of Athena's classification of a small number of randomly-selected examples before applying the classifier leads to further improvement. Note that we obtain these results even though we provide the traditional clustering techniques with knowledge of the true number of clusters. The fact that such improvements are possible suggests that a "true" clustering as envisioned by an end-user who organizes documents rarely matches the true clustering as defined by any statistical technique. By allowing interactivity in the clustering process, our approach allows the user to guide the algorithm towards a more desirable partitioning of the input documents.

4 Conclusions and Future Work

We addressed the problem of simplifying the process of hierarchical document organization and management through text mining algorithms. Using classification and clustering as the basic building blocks, our Athena system provides rich functionality for automatic creation and maintenance of hierarchical text databases. Using Athena, users can discover topics in their unorganized collec-

tion of documents and partition the documents according to these topics. They can reorganize a hierarchical collection into a different hierarchy by giving only few documents as examples. They can exploit the information contained in their current hierarchy to route new documents. They can also find misfiled documents and identify when some node in a hierarchy may need reorganization due to concept drift within a node.

The implementation of this application placed new requirements on the classification and clustering technology, making the use of standard solutions inadequate. For classification, we enhanced the basic Naive-Bayes algorithm with several features including the use of Lidstone's law of succession, under-weighting long documents, and over-weighting author and subject. These enhancements led to 7 to 29% absolute increase in accuracy on our real-life test data sets.

We also developed a new interactive clustering algorithm. This algorithm first finds highly accurate cluster digests (partial clusters), obtains user feedback to merge/correct these digests, and then uses the classification algorithm to complete the partitioning of the data. In our experiments, this new algorithm resulted in 10-20% increase in absolute accuracy over k-means and agglomerative clustering. While strict classification and clustering algorithms have been well-studied in previous work, this interactivity dimension has received comparably little attention. We feel further research directed towards cooperation between on-line data-mining algorithms and the end-user will prove fruitful.

Acknowledgements We would like to thank Andreas Arning, Ashok Chandra, Dimitris Gunopulos, Howard Ho, Sunita Sarawagi, John Shafer and Magnus Stensmo for their contributions to the design of Athena.

References

[ABS99] R. Agrawal, R. Bayardo, and R. Srikant. Athena: Mining-based interactive management of text databases. Research Report RJ 10153, IBM Almaden Research Center, San Jose, CA 95120, July 1999. Available from `http://www.almaden.ibm.com/cs/quest`.

[ADW94] C. Apte, F. Damerau, and S.M. Weiss. Automated Learning of Decision Rules for Text Categorization. *ACM Transactions on Information Systems*, 1994.

[CDAR97] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases. In *Proc. of the 23rd Int'l Conf. on Very Large Databases*, pages 446–455, 1997.

[CDPT92] D.R. Cutting, K.R. David, J.O. Pedersen, and J.W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proc. of the 15th Intl ACM SIGIR Conf. on Research and Development in Information Retrieval*, 1992.

[Coh96] W.W. Cohen. Learning Rules that Classify E-Mail. In *Proc. of the 1996 AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[Goo65] I.J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. M.I.T. Press, 1965.

[Har89] G. Hardy. Correspondence. *Insurance Record*, 1889.

[KBS97] R. Kohavi, B. Becker, and D. Sommerfield. Improving simple bayes. In *The 9th European Conference on Machine Learning, Poster Papers*, 1997.

[KMST98] P. Kontkanen, P. Myllymaki, T. Silander, and H. Tirri. BAYDA: Software for Bayesian Classification and Feature Selection. In *Proc. of the Fourth Int'l Conf. on Knowledge Discovery and Data Mining*, 1998.

[Koh96] R. Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In *Proc. of the Second Int'l Conf. on Knowledge Discovery and Data Mining*, 1996.

[Lan95] K. Lang. News Weeder: Learning to Filter Net-News. In *Proc. of the 12th Int'l Conf. on Machine Learning*, pages 331–339, 1995.

[Lid20] G. Lidstone. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Trans. Fac. Actuaries*, 8:182–192, 1920.

[Lot] Lotus Notes. http://www.notes.net.

[LR94] D.D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *In Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–92, 1994.

[Mae94] P. Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, 1994.

[Mit97] Tom M. Mitchell. *Machine Learning*, chapter 6. McGraw-Hill, 1997.

[MN98] Andrew McCallum and Kamal Nigam. A Comparison of Event Models for Naive Bayes Text Classification. In *AAAI-98 Workshop on "Learning for Text Categorization"*, 1998.

[MRMN98] Andrew McCallum, Ronald Rosenfeld, Tom Mitchell, and Andrew Ng. Improving Text Classification by Shrinkage in a Hierarchy of Classes. In *Intl. Conf. on Machine Learning*, 1998.

[PB97] M. Pazzani and D. Billsus. Learning and Revising User Profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[PE97] T.R. Payne and P. Edwards. Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface. *Applied Artificial Intelligence*, 11:1–32, 1997.

[Ras91] E. Rasmussen. *Information Retrieval: Data Structures and Algorithms*, chapter Clustering algorithms, pages 419–442. Prentice Hall, Englewood Cliffs, NJ, 1991.

[Ris95] E.S. Ristad. A Natural Law of Succession. Technical report, Princeton University, 1995. Research Report CS-TR-495-95.

[SAM96] John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, Bombay, India, September 1996.

[SDHH98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-mail. In *Proc. of the AAAI'98 Workshop on Learning for Text Categorization*, Madison, Wisconsin, 1998.

[SK99] R. Segal and J. Kephart. MailCat: An Intelligent Assistant for Organizing E-Mail. In *Proc. of the Third Int'l Conf. on Autonomous Agents*, 1999.

[SYB98] M. Sahami, S. Yusufali, and M.Q.W. Baldonado. Sonia: A service for organizing networked information autonomously. In *Proc. of the Third ACM Conference on Digital Libraries*, pages 200–209, 1998.

[Yah] Yahoo! http://www.yahoo.com.