Scalable Clustering of News Search Results

Srinivas Vadrevu, Choon Hui Teo, Suju Rajan, Kunal Punera,
Byron Dom, Alex Smola, Yi Chang, Zhaohui Zheng
Yahoo! Labs
Sunnyvale, CA, USA
{svadrevu,choonhui,suju,kpunera,bdom,smola,yichang,zhaohui}@yahoo-inc.com

ABSTRACT

In this paper, we present a system for clustering the search results of a news search engine. The news search interface includes the relevant news articles to a given query organized in terms of related news stories. Here each cluster corresponds to a news story and the news articles are clustered into stories. We present a system that clusters the search results of a news search system in a fast and scalable manner. The clustering system is organized into three components including offline clustering, incremental clustering and realtime clustering. We propose novel techniques for clustering the search results in realtime. The experimental results with large collections of news documents reveal that our system is both scalable and also achieves good accuracy in clustering the news search results.

Categories and Subject Descriptors

I.7.m [**Computing Methodologies**]: Document and Text ProcessingMiscellaneous

General Terms

Algorithms, Experimentation

Keywords

clustering, news clustering, news search, realtime clustering, query based clustering

1. INTRODUCTION

Clustering of search results provides a unified view on the search results by grouping the similar documents. This allows the user to examine all the related categories of the search results without requiring to go through hundreds of items. Clustering becomes more import in the context of a domain like news because there could be thousands of related articles to a given query. Most of these news articles are related however and if we group the articles in terms of related stories, then the number quickly reduces to a handful. It makes it easier for the user to browse the search results in terms of news stories, rather than individual news articles. Furthermore, news articles re-publish the article from a different source to provide comprehensive coverage on a particular topic. This further exacerbates the problem by increasing the redundancy in the search results. Instead of leaving the search result organization to the user, a clustered representation of search results provides an overview to explore a topic.

The news search also has another component of recency which introduces additional complexities. Although two articles are very related in terms of document content, if they are far apart in the time axis, then they might correspond to a repetitive event, such as *california earthquake*. Even though such events occur multiple times, they correspond to different clusters because they are totally independent events and the user may be interested in looking at them individually.

Most of the work on search result clustering focused more on salient feature extraction and utilizing this information to cluster the search results [19]. However the salient features do not provide the complete picture about the document. Sometimes most important information about the news article is buried deep into the document in the body. Later we provide experimental evidence that shows that the body of the document provides a great boost in the performance of the clustering system. Other approaches [7, 3] utilize standard clustering algorithms like hierarchical agglomerative clustering (HAC) algorithm and partitional algorithms such as K-means. In [18], clustering problem is modeled as phrase extraction from documents and then utilizing suffix tree based clustering algorithm to group documents that contain common phrases. However we show that the query information is vital in clustering the search results and also show how to utilize this information in the clustering algorithm.

Other research related to clustering the search results focus on utilizing various matrix factorization techniques [13, 12] such as singular value decomposition, non-negative matrix factorization [10], local non-negative matrix factorization [11] and concept decomposition [4]. These clustering algorithms focus on extracting quality descriptions for the clusters by finding the labels with matrix factorization on the word features from snippets of search results. Although the descriptions may be important in web search results clustering, they do not play a vital role in news search results as the most representative document can be used to show

WSDM'11, February 9–12, 2011, Hong Kong, China.

the summary of the cluster. Another algorithm called DisCover [9] clusters search results by maximizing the coverage and distinctiveness of the clusters.

Another line of work is related to using named entities to browse the clusters of search results [16]. However this work mainly focuses on extracting the named entities from search results organizing them into clusters using the named entities. Other related work [6] identifies discriminative, ambiguous and common terms by constructing a relational graph of terms and using them to cluster web search results.

It is clear that the named entities that occur in the body of the documents are valuable to produce quality clusters of search results. However it is expensive to process the entire documents to cluster the search results in realtime as it impacts the user experience. In this work we propose a scalable clustering technique that can provide a fast execution time without sacrificing the accuracy by utilizing the body features in terms of offline clusters that provide a prior on the document relatedness of search results. The offline clusters are obtained by a process that is run offline. We also handle incremental clustering for documents that arrive continuously.

In fact the offline clusters that are obtained by clustering the entire corpus provide a useful representation and the search results can be organized into clusters by using this information alone. Some of the existing work follows exactly these methodologies. However such organization would have a fixed granularity of clustering, which may not be desirable for all queries. Some queries can be broad and require a higher level of clustering hierarchy and some other queries can be very specific and require a fine level of clustering hierarchy. If the offline clusters alone are used to organize the search results, the granularity of the clustering cannot be adjusted according to query. However the solution we provide overcomes this problem by applying a clustering algorithm on the search results in realtime by utilizing the offline clusters as features.

The contributions of our work are two-fold:

- We provide a unified framework for scalable online clustering and detail the architecture of the system that describes each of the three components: offline clustering, incremental clustering and realtime clustering.
- We propose novel techniques to cluster search results in realtime by incorporating the query information in the clustering algorithm itself.

The paper is organized as follows. Section 2 presents the overall architecture of our system. Sections 3 and 4 and 5 discusses various clustering algorithms corresponding to offline, incremental and realtime clustering that we used in this work. Section 6 presents the experimental results for our algorithms and Section 7 shows the conclusions and future work.

2. ARCHITECTURE OF THE SYSTEM

To cluster the search results in realtime, we just need the top ranked search results. However, the user would like to browse all the related articles in relation to a particular news story, which may not be covered in the top ranked news results. To address this issue, we rely on offline clustering that clusters the entire news corpus. We utilize the offline clusters as an additional source of information in realtime clustering, which helps its performance as well. However this offline batch processing of documents, especially news articles does not work efficiently as news articles arrive continuously every second. To address this, we also incorporate incremental clustering solution to address the streaming data clustering problem. Figure 1 shows the architecture of our system that describes individual components that address these needs.

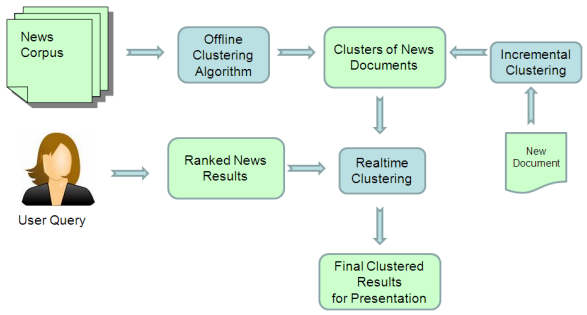

Figure 1: The Architecture of the news search result clustering.

The offline batch clustering algorithm will be run on a regular basis, multiple times a day on the news corpus, which we limit to a full month of news articles. This clustering algorithm assigns a cluster ID to every document that is present to the algorithm. The incremental clustering algorithm works in a streaming fashion assigning cluster IDs to documents that arrive in the interim time before the next offline batch clustering is run. This is a continuous process that assigns cluster IDs to documents based on the existing batch clustering algorithm output. Thus each document in the corpus at any given time will have at least a cluster ID from either the offline clustering algorithm or the incremental clustering algorithm. The realtime clustering algorithm, which is run when a query is issued, is executed at runtime utilizing these cluster IDs for each of the documents in the corpus. This realtime clustering groups the search results into clusters and provides the final presentation of clustered search results.

In the next three sections, we describe the offline clustering, incremental clustering and realtime clustering components in detail.

3. OFFLINE CLUSTERING

Our offline clustering system draws from the work of [15]. In this method, the similarity between all pairs of articles is first computed via Locality Sensitive Hashing (LSH). While we also use the LSH method to determine the pairwise similarities, we then construct a similarity graph on the corpus, wherein a pair of articles has an edge if the similarity between the two meets a threshold. A correlation clustering algorithm [1] is then run on top of the similarity graph to produce the final clustering. Note that it is essential to define a good similarity measure while constructing the graph. False edges causes different story clusters to be merged whereas missing edges causes stories to be split. Figure 2 shows the main components of our clustering system. In the following sub-sections, we describe in detail the design of each component.

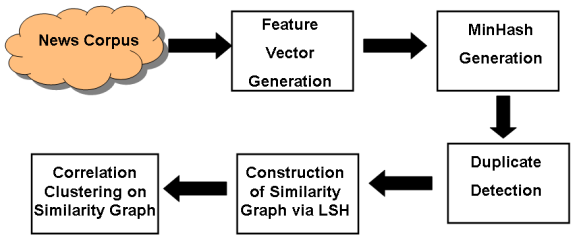

Figure 2: Overview of Offline Clustering System.

3.1 Feature Vector Generation

The good performance of our offline clustering system is heavily dependent on the underlying similarity function that is used to construct the similarity graph. As mentioned earlier, a poorly designed similarity function can either merge different stories or split a story into many clusters. Note that, we use the term *story* to represent the set of news articles that are about the same news story and hence should belong to the same cluster. News articles have several important information sources associated with it that can be used to define a custom similarity function. The different types of features that were extracted are:

- TF-IDF: A unigram based feature vector was constructed using the TF-IDF values for the words in a news article after stop-word removal and stemming
- Wikipedia Topics: Wikipedia topics were extracted from the news article using the technique described in [17]. The set of Wikipedia topics were then assigned an "aboutness score" which represents how important that topic is to the article. The ranked list was then used as features with the feature value corresponding to the aboutness scores.
- Part of Speech Tagging: The news article was also tagged with a Part of Speech tagger and unigrams were extracted from nouns, adjectives and verbs and used as features. Term frequencies were used as feature values.

In addition to the above feature vectors, we also made use of presentation cues associated with the article to emphasize certain phrases or unigrams such as the fact that a phrase appears in the title, or abstract, or is italicized etc. Thus, the different features mentioned above were assigned a score based on their presentation in the news article. The features from the three different channels were then combined through a simple aggregation of weights assigned to unigrams from each channel. The feature vector was then unit normalized before being used to compute the cosine similarity. Another important feature that was used is that of time. News articles typically have a time stamp associated with it. Given two articles published on days t_1 and t_2, the cosine similarity on the custom feature space was weighted by $\exp(-|t_1 - t_2|/7)$. The intuition behind this weighting function is that the closer the date of publication of the two articles, the more likely they are to be similar. Since we believe that a story cluster typically should not contain any articles that are apart by more than a week we decrease the similarity between such pairs even more.

While it is trivial to compute the feature spaces and compare all pairs of articles within a small set such pairwise comparison becomes computationally expensive for larger corpora. A corpus with 100,000 articles requires 10,000,000,000 such comparisons. However, once an article has been mapped into its feature space the chances of a pair of completely unrelated articles sharing any useful features is quite low. It is then unnecessary to explicitly compute the pairwise similarity between such pairs. We make use of the LSH to eliminate unnecessary similarity computations. An important component of the LSH method is the generation of Minhash signatures for each article.

3.2 Minhash Signature Generation

Minhash signatures are a succinct representation of each article computed such that the probability that two articles has the same Minhash signature is equal to the Jaccard similarity between the two [8]. In order to compute the Minhash signature for each article, we first represent it as a feature vector in the custom feature space described in Section 3.1. We then construct a length 100 Minhash signature using the technique detailed in [15]. However, the randomized nature of the Minhash generation method requires further checks to increase the chances of uncovering all pairs of related articles and removing articles that were brought together by chance. Thus, we resort to LSH to reduce chance pairings. Prior to performing LSH, the Minhash signatures can also be quickly used to detect exact duplicate.

3.3 Duplicate Detection

Given each article and its 100-length Minhash signature we use these signatures to identify articles that are duplicates of each other. If an article is a duplicate of another it is unnecessary to include both in the clustering process. One article per group of duplicates is sufficient to determine the cluster memberships of the rest. Given that a typical news corpus has a number of duplicates, because of different publishers re-printing the same story from a news agency such as AP, identifying such groups and using only the unique documents for clustering provides significant computational savings. If two articles have the same Minhash value in each of the 100 slots they are highly likely to be duplicates and are marked as such. Thus, the articles are grouped by their Minhash signatures and a representative is chosen for each group to participate in the clustering process. Once the representative article has been assigned to a cluster we then propagate that cluster membership to all of its duplicates as a post-processing step after clustering.

3.4 Locality Sensitive Hashing

Although the Minhash technique allows us to represent each article by a length 100 signature and enables duplicate detection, we still need a $O(N^2)$ comparison to determine the similarity between all pairs of articles. However, as mentioned earlier, documents that are unrelated are likely to have very a low similarity which value need not be explicitly computed. Locality Sensitive Hashing can be used to quickly eliminate pairs of articles that share very few features from the pairwise similarity computation. We use the method detailed in [15] for LSH. For each article we construct a shorter LSH signature by concatenating a smaller set (2) of Minhash signatures. This process is repeated 200 times. Thus, documents which contain at least a few words in common are likely to agree in at least one of the LSH signatures. Only those pairs of articles with the at least one common LSH signature need to have their similarity computed. The

particular settings of our LSH method, 200 LSH signatures of length 2, were obtained by offline experimentation on a smaller dataset. The current parameter settings enabled us to uncover $\sim 96\%$ of all pairs of similar documents as a full-blown pairwise comparison. Pairwise similarity is computed on all the documents sharing a LSH signature. Note that cosine similarity is computed on the unit normalized vectors represented in the custom feature space and not on the Minhash signatures. The cosine similarity thus computed is further weighted with the time information as explained in section 3.1. Only those pairs of articles whose similarity exceeds a user-defined threshold are recorded. A similarity graph is then constructed using the output of the LSH process. Each article is represented as a node in the graph and an edge exists between two nodes if its similarity exceeds the threshold. The edges are also weighted by the cosine similarity measure.

3.5 Correlation Clustering

The similarity graph is then fed into a correlation clustering algorithm based on the work in [1] to partition the graph into clusters. Correlation clustering is also a randomized algorithm that attempts to minimize a cost function based on the number of dissimilar pairs in the same cluster and the number of similar pairs in different clusters. We modified the original algorithm to allow the weight on the edges that are cut or formed, as clustering proceeds, to participate in the cost function. The algorithm is sensitive to the initialization data point. So we start multiple correlation clustering algorithms multiple times with different random seeds and identify the one with the lowest cost as the final clustering solution. An important characteristic of the correlation clustering approach is that it does not require the specification of the number of clusters. This feature is important as it is not easy to guess the number of clusters in an evolving corpus in which a major news event can trigger multiple stories over a few days.

3.6 Evaluation

In order to evaluate the performance of the offline clustering system in terms of the quality of the clusters produced, editorial tests were conducted. We first constructed a corpus of news articles collected over a week. The size of this corpus is $\sim 700K$. Pairwise similarities were then computed between all pairs of articles in the corpus. ~ 5000 article pairs were sampled by stratified random sampling with the stratification done on the similarity measure. Pairs of articles were then presented to the editors and were labeled as follows:

- Must-link: Includes pairs of articles that are duplicates, pairs in which one article summarizes or paraphrases the content in the other and also those pairs covering the same news event but with different text.
- Maybe-linked: When the two articles are about the same set of entities but in two different news stories the pair is labeled as Maybe-linked.
- Cannot-link: When the two articles in the pair are about unrelated news stories it is marked up as Unrelated.

The clustering system was then run on the same corpus and we compute the fraction of times pairs of articles appears in the same cluster for each label. Note that this number should be high for must-links and low for cannot-links. The maybe-linked label is the gray area where it is not entirely wrong to show the two articles in the same cluster but for better user experience one would want to keep this number low as well. The results of the clustering system are shown in Table 1. As can be seen, the offline clustering system performs well.

Label	Editor Pairs	# in Same Clus.	%in Same Clus.
Must Link	1958	1504	77%
Maybe Linked	1327	293	22%
Cannot Link	1436	40	2.8%

Table 1: Performance of Offline Clustering System

4. INCREMENTAL CLUSTERING

The offline clustering phase produces a set of clusters of similar/relevant documents. These clusters are then taken as groundtruth for constructing a classifier for incremental clustering. The incremental clustering refers to the task of assigning a new document that has just arrived at the system to a cluster it is *most likely* to be associated to.

Since offline clustering is usually scheduled to run at the interval of a couple of hours, it is likely the case that news that has just broke after the offline clustering phase does not belong to any of the existing clusters. We describe here three simple classifiers for the purpose of incremental clustering with strategies to reduce the potential impact of this scenario:

Static A standard classifier that must be re-trained on the latest set of documents at a shorter time interval.

Semi-adaptive: A classifier that is capable of creating new class for news articles which are not close "enough" to any of the existing clusters. The closeness here is a value which requires careful tuning.

Fully-adaptive: A classifier that is not only capable of creating new classes but also updating itself to cope with the evolution of the news in the existing clusters. That said, the classifier is also able to remove classes corresponding to "submerging" stories. Compared to semi-adaptive classifier, this classifier is more flexible but more likely to suffer from sub-optimality as it is sensitive to the order of the news articles that arrived at the system.[1]

The three types of classifiers described above roughly cover the whole spectrum of incremental clustering. The choice of a specific classifier depends on the computational and time constraints we are confined to: As long as offline clustering can be carried out more frequently, static classifier is perhaps the best as it is simple to implement, easy to deploy, and requires no maintenance effort. Otherwise, semi- or fully-adaptive classifiers, which are harder to maintain, can be used instead.

[1]This cause of sub-optimality is also commonly seen in online learning.

5. REALTIME CLUSTERING

Realtime clustering is vital to adjust the granularity of clustering at the query-time. For example, the query 'earthquake' in a news search engine returns clusters of news stories where the each cluster represents news story discussing about an earthquake occurred in a particular location. Thus, all the results related to Chile earthquake may be grouped in to a single cluster. However a related query such as 'chile earthquake' might return detailed news stories that are all related to the Chile earthquake. In this case, the news stories may be discussing about the damages caused by the earthquake, donation related information and the incredible force of the earthquake, which depict various aspects of Chile earthquake.

Thus adjusting the granularity in realtime clustering is very important and we propose three novel techniques to handle this that are described in following sections. Each of the methodologies shows how to modify the similarity measure in a standard clustering algorithm that can be used to cluster the top documents retrieved by a news search engine. In our experiments we used hierarchical agglomerative clustering (HAC) to incorporate these similarity measures and compared the proposed similarity measures with the standard cosine similarity.

5.1 Meta-Clustering and Textual Matching

This approach relies on the output of the offline clustering output and also textual matching features among the query and the documents. The similarity measure in this clustering can be formulated as:

$$\text{sim}(q, d_1, d_2) = \sum_{i=1}^{K} w_i c_i + \frac{\text{bm25}(q, d_1 \cap d_2)}{\text{bm25}(q, d_1) + \text{bm25}(q, d_2)}$$

where

- K is the number of offline clustering algorithms
- w_i is the weight for the clustering algorithm i
- $c_i = 1$ if the clustering algorithm i puts d_1 and d_2 in the same cluster
- $c_i = 0$ if the clustering algorithm i puts d_1 and d_2 in different clusters
- $d1 \cap d2$ is the overlap of documents d_1 and d_2

The first term in this equation is the term corresponding to the offline clustering weights. This is similar to ensemble cluster learning or meta-clustering that is a simple combination of various clustering solutions from the offline clustering assignments. The second term in the equation relies on the textual matching features among the query and the documents. This formation relies on the textual matching features among the query and the documents. We use BM25 [14] as a proxy for textual matching between the query and the document. Given a query q and a document d, BM25 score can be computed as

$$BM25F(d) = \sum_{t \in q \cup d} \frac{\bar{x}_{d,t}}{K_1 + \bar{x}_{d,t}} w_t^{(1)}$$

$$\bar{x}_{d,t} = \frac{x_{d,t}}{1 + B(\frac{|d|}{|avgdl|} - 1)}$$

where $w_t^{(1)}$ is the usual relevance weight of the term t in the document, which can often be the inverse document frequency, $x_{d,t}$ is the frequency of the term t in the document d and $\bar{x}_{d,t}$ is the weighted term frequency that can be further broken into fields which can, in turn have individual weights. The term frequency is normalized with the document length with $avdl$, which is average document length in the corpus, so that long and short documents are treated in the same manner. K_1 and B are parameters in this score, for which we used standard weights in our experimental setting.

5.2 Contextual Query-Based Term Weighting

In this *QrySim* methodology, we modify the weights of the term vectors to account for the context around the query in the document. We want to weigh the terms that appear closer to the query higher than the terms that appear farther from the query.

Our postulation is that terms that occur closer to the query terms are more important in realtime query-dependent clustering than those that occur far from the query terms. This can be validated from considering an anecdotal evidence. Consider a query 'earthquake' where the user might be interested in research about the recent earthquakes that happened, so the important terms for clustering in this case would be locations such as 'Haiti', 'Chile', 'Japan', etc, all of which occur close to the term 'earthquake'. However if the query is 'Chile Earthquake', the important terms for clustering might be different, such as 'Rescue', 'death toll', 'donations', etc. Finding the representative terms for each query might be difficult and a good approximation for finding them is to look for the context around the query.

We first construct the query position vector for each document, which lists the positions where the entire query occurs together in the document. We experimented with various n-grams of the query (unigrams and bigrams) as a proxy for the entire query to increase the coverage, but we found that using the entire query to build the position vector works well in practice. With this position vector, we parse each document and assign the weight for each term as the distance of the term from the closest query occurrence, i.e.,

$$F_t' = F_t * \frac{1}{\sqrt{d_{\min}}}$$

Where F_t is the original frequency/term weight of the term t and d_{min} is the closest distance of the term t to the query occurrence in the document.

With these new weights to each of the terms, we construct new term vector for each document and use them in computing the similarity measure for clustering, which weights the terms that appear closer to the query terms higher.

5.3 Offline Clusters as Features

Another way to utilize the offline cluster information into the realtime clustering is to use the offline cluster ids as additional features when we cluster the top n results. Before we add the offline cluster id information, we show how valuable the body information is to the realtime clustering. Figure 3 shows the comparison of clustering algorithms with and without using body features. It can be clearly seen that the body features provide a good boost to the performance of the clustering algorithm.

Although the body provides vital information to the clus-

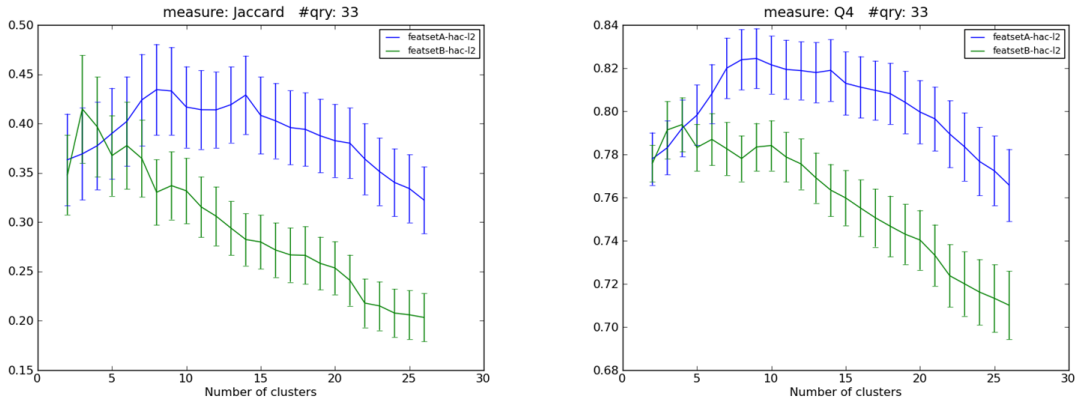

Figure 3: Usefulness of body features in realtime clustering. The figure shows the values of clustering evaluation metrics, Jaccard and Q4, averaged over results of 33 queries for various number of clusters (K=1 to 30). The green line corresponds to the feature set with only title and abstract features and the blue line corresponds to the entire feature set which includes body features, which is clearly superior.

tering algorithm, the feature space increases dramatically if all of the body features are used in the realtime clustering algorithm. Since the realtime clustering algorithm needs to be executed at runtime after the query is issued, this poses latency issues as the clustering algorithm needs to compute similarity measure between the documents that operates on this huge feature vector. Thus, it is expensive to use all of the body features in the realtime clustering algorithm. To address this problem, we propose to utilize the offline cluster IDs as additional features to the online clustering algorithm. In computing the similarity measure for the realtime clustering, these offline cluster IDs can be used to determine the closeness of two documents based on whether they have similar offline cluster IDs. For this purpose, a document can have multiple offline clustering IDs from either different algorithms or with the same algorithm with different granular settings such as coarse and fine. We utilize the standard HAC for the realtime clustering algorithm with the standard cosine similarity with the regular term features and use Jaccard similarity to compute the similarity in the offline cluster IDs.

Thus the final similarity measure between two documents in this setting is as follows:

$$\text{Sim} = \alpha * \text{CosineSim} + (1 - \alpha) * \text{Jaccard}$$

Where *CosineSim* is the cosine similarity computed on the bag of words based term features and *Jaccard* is the Jaccard similarity measure between the vector of offline cluster IDs for two documents and α is the tradeoff parameter between the two similarity measures, which we set it to 0.5 in our experiments. The *Jaccard* similarity between two documents can be computed as

$$\text{Jaccard}(C_1, C_2) = \frac{C_1 \cap C_2}{C_1 \cup C2}$$

where C_1 and C_2 are vectors of offline cluster IDs of two documents that correspond to output from various clustering algorithms or the same clustering algorithm with various granular settings.

5.4 Performance Analysis

In this section, we provide a short analysis to estimate the number of computational operations required by using the entire body features as compared with using just the offline clusters as an approximation. We performed this analysis on an editorially evaluated data set containing 300 queries and approximately 25000 news documents. The clustering features contain words from three sections of the news documents, including title, abstract and the body. The number of unique body features is 20 times more than the features contained in both title and abstract. Thus we observe far greater number of unique features in the body, thus increasing the total number of operations when computing the similarity between documents if the body features are included in the computation. Hence we gain $20x$ savings in terms of number of operations if we do not use the body features and use the offline clusters as a proxy for these features.

6. EXPERIMENTS

In this section we present the experimental evaluation of our techniques on Yahoo! news corpus and search engine. The results we present evaluate the end product of the entire system that we presented, which is the realtime clustering. Thus we present several experimental evaluations to determine the quality of the clustering output on the search results given a query. In the following sections, we first describe the experimental data we used, present various evaluation metrics we used and finally show the results to evaluate various techniques we described.

6.1 Experimental Setup

We scrape the Yahoo! news search engine with a list of random queries sampled from the query log and collect the top 100 news search results returned by the ranking algorithm. Each of these search results correspond to an individual news article and our goal is to group these search results into clusters that refer to related news stories. Since the news articles are coming through feeds from news sources, we have access to the individual fields such as publication time, url, title, abstract and body of the news article. We ex-

Algorithm	Precision	Recall	F-Measure	Rand
Minhash Clustering	0.77	0.92	0.84	0.66
Subspace Clustering	0.83	0.90	0.86	0.69
K-Means (k=20)	0.89	0.93	0.91	0.63
K-Means (k=25)	0.95	0.96	0.95	0.71
Meta Clustering	0.95	0.96	0.95	0.71
Meta Clustering + Textual Matching	0.97	0.96	0.96	0.75

Table 2: Results with various simple offline clustering algorithms and the realtime clustering algorithm which includes meta clustering algorithm.

Metric	OrigSim	QrySim	Gain	T-test p-value	% times OrigSim is better
Q_4	0.724	0.731	0.96%	0.06	62%
Jaccard	0.258	0.260	0.74%	0.07	62%
Rand	0.564	0.571	1.27%	0.02	96%

Table 3: Realtime Clustering Results with QrySim similarity measure that boosts the weights to the terms that occur close to the query term over the standard similarity measure (OrigSim) with equal weights to all terms.

tract several features from these news articles including simple term features, unigram and bigram features and named entity based features. To extract named entities from the text of news articles, we use a dictionary based maximum entropy based tagger [2].

Each of the search results for a given query are editorially grouped into related clusters by incrementally going through them one by one. For example, first document is assigned to cluster A. If the second document is similar to cluster A, it is assigned to cluster A and if it is not similar, then it is assigned to a new cluster B. In this manner, all the top 100 results are grouped into clusters. We define a cluster to be a news story. For example, the documents in the same cluster are referring to the same news story. If the story is a developing story, then we require that the related news articles to be in the same cluster, unless they talk about significantly different news story. With this criterion for clustering the news articles, all documents within the same cluster correspond to same news story from various news sources such as New York Times and Wall Street Journal.

The offline clustering algorithms were run on the entire corpus of news documents within one month of time frame, comprising millions of news articles that correspond to US/English documents. The features for offline clustering algorithms utilize $600,000$ features comprising of unigram and bigram features and named entity based features. For realtime clustering evaluation, we editorially labeled top 100 results into clusters for a set of 300 queries. The features used for realtime clustering algorithm were chosen to be a subset of the features used in offline clustering algorithm, comprising $12,000$ features.

6.2 Evaluation Metrics

We utilize the following extrinsic metrics to evaluate our clustering algorithms that compare the performance of a given clustering solution to the editorial solution:

- **Precision**: If C is the number of clusters to be evaluated, L is the number of categories (from editorial judgments) and N is the total number of documents (100 per query), precision can be computed as the weighted average of maximal precision values.

$$\text{Precision} = \sum_i \frac{|C|}{N} \max_j \text{Precision}(C_i, L_j)$$

$$\text{Precision}(C_i, L_j) = \frac{|C_i \cap L_i|}{|C_i|}$$

- **Recall**: Recall, also referred as inverse purity focuses on the cluster with maximum recall for each category. Recall can be defined as

$$\text{Recall} = \sum_i \frac{|L_i|}{N} max_j \text{Precision}(L_i, C_j)$$

- **F-Measure**: Standard F-Measure can be computed from treating the precision and recall values equally.

- **$\mathbf{Q_4}$**: This is an information-theoretic validity measure that characterizes how well the cluster labels assigned to the objects by a clustering algorithm agree with their (manually assigned) class labels by the number of bits required to encode/compress the class labels of the objects given knowledge of their cluster labels. This code length corresponds to a special encoding scheme inspired by the *Minimum Description Length* principle[**?**, **?**]. The form of the measure that gives the bits per object to do this encoding is Q_3. The measure-form Q_4 is simply a normalized form of Q_3, designed to vary between zero (no information about class labels contained in cluster labels) and one (perfect clustering.)

$$Q_3(C,K) = \frac{1}{n} \sum_{k=1}^{|K|} \left[\log \binom{v(k)}{\{v(c,k)\}} + \log \binom{v(k)+|C|-1}{|C|-1} \right],$$

where $\binom{v(k)}{\{v(c,k)\}} = \frac{v(k)!}{\prod_{c=1}^{|C|} v(c,k)!}$ = multinomial coefficient

$$Q4(C,K) = \frac{\max_K Q_3(C,K) - Q_3(C,K)}{\max_K Q_3(C,K) - \min_K Q_3(C,K)}$$

$$= \text{normalized } Q_3(C,K)$$

These measures are refinements of the measures Q_0 and Q_2 derived in [5]. These refinements remove a form of redundancy known as *incompleteness* from Q_0 and Q_2. The details of the refinements are discussed in [**?**]. We chose to include Q_4 in our set of clustering-accuracy metrics because it satisfies a set of desirable properties, which are not all satisfied by more traditionally used metrics based on pairwise agreement/disagreement. These properties are listed in [5] and the fact that they are not all satisfied by traditional measures is demonstrated there as well.

- **Jaccard**: This is a simple Jaccard coefficient computed on the pairs of documents. It is computed as the number of pairs of documents that are supposed to be together and the algorithm actually put them together over all of the number of pairs of documents.

$$\text{Jaccard} = \frac{\text{SS}}{\text{SS} + \text{SD} + \text{DS}}$$

Algorithm Description	Avg Q4
Best Single Offline Clustering Algo	0.7340
Title + Abstract Features Only	0.7584
Title + Abstract + Best offline set of clusters	0.7801
Title + Abstract + Body Features	0.8157
Title + Abstract + Body + Best offline set of clusters	0.8208

Table 4: Q_4 values with Realtime Clustering algorithm with various combinations of features. The baselines include features with title and abstract and a single offline clustering algorithm. Although the combined feature set with all the features is the best one, the features with the offline clusters and title and abstract features is comparable to the one which includes body features.

Offline Cluster Sets	Avg Q4
(1,2,3)	0.78009
(1,2)	0.77686
(1,3)	0.77444
(1)	0.77418
(2)	0.77130
(2,3)	0.77036
(3)	0.76155

Table 5: Q_4 values with Realtime Clustering algorithm with various granularity settings of offline clusters as features. The baseline feature set includes just title and abstract features. The numbers 1,2,3 refer to different settings of the offline clustering algorithm at different granularity settings, specifically varying from coarse to fine representation of clusters. It can be observed that the best accuracy is obtained by combining all the configurations, and individual cluster IDs themselves provide inferior performance.

Where SS is the number of pairs that belong to the same algorithmic cluster and editorial class, SD is the number of pairs that belong to same algorithmic cluster and different editorial class and DS is the number of pairs that belong to different algorithmic cluster and same editorial class.

- **Rand**: This statistic relies on counting pairs of documents and their assignments into appropriate clusters and categories. Essentially it is the ratio of the number of pairs of documents that got correctly assigned into categories to the total number of pairs of documents.

$$\text{Rand} = \frac{\text{SS} + \text{DD}}{\text{SS} + \text{SD} + \text{DS} + \text{DD}}$$

6.3 Evaluating Meta-Clustering and Textual Matching

Table 2 shows the performance of various offline clustering algorithms applied directly on the editorial dataset for realtime clustering. It can be seen that the best offline clustering algorithm is K-Means with 25 clusters which achieves F-Measure of 0.95. A meta-clustering algorithm which is simply a combination of various clustering algorithms achieves the same performance as the best offline algorithm, while textual matching that utilizes the query information improves upon this result as it incorporates additional information in terms of query.

6.4 Results with QrySim

Table 3 shows results with *QrySim* method described in Section 5.2. HAC is used a clustering algorithm for these results. We compared the standard cosine similarity with the *QrySim* similarity measure where we emphasize the weights on the terms that are close to the query. The *QrySim* method clearly outperforms the standard similarity measure as shown in the results.

6.5 Results with Offline Clusters as Features

Next, we present experimental results by evaluating the efficacy of using offline clusters as features in the realtime clustering. In this setting, we used minhash offline clustering algorithm to cluster a big corpus of news documents and used HAC to cluster the top 100 search results by using the offline cluster ids as features.

The baseline feature set includes simple features such as bag of words based term features and we experimented with three variations of minhash clustering algorithm with various number of hash functions for the offline clustering algorithm as additional features. We also utilize the structured fields in the news articles such as title, abstract and body as individual features. The results are shown in Table 4.

We also experimented with various granularity settings for the offline clustering algorithm mentioned in Section 3 and their usefulness as features in the realtime clustering algorithm. The Table 5 shows the results with various combinations of such granularity settings as features in the realtime clustering algorithm. The results show that the redundancy in the clustering algorithms is helpful to achieve maximum accuracy in the realtime clustering. We can also observe a trend that the accuracy decreases as we go from coarse to fine granularity in the clustering algorithm.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an overall system for clustering news search results. We presented a system that involves clustering the entire news corpus with an offline clustering algorithm and handling the incoming streaming data with incremental clustering algorithm. The output from the offline and incremental clustering algorithms are then utilized for improving the realtime query based clustering. The experimental results indicate that the meta clustering that combines various offline clustering algorithms is as good as the best clustering algorithm, but the realtime query based clustering can be improved upon this by utilizing the query information. We also show how to utilize the offline cluster information in the realtime clustering by using them as features and this shows an improvement both in accuracy and the performance of the system.

8. REFERENCES

[1] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *MACHINE LEARNING*, pages 238–247, 2002.

[2] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1):39–71, 1996.

[3] K. Chakrabarti, S. Cauduri, and S. won Hwang. Automatic categorization of query results. In *Proceedings of SIGMOD 2004*, 2004.

[4] I. Dhillon and D. Modha. Concept decomposition for large sparse text data using clustering. *Machine Learning*, 1(42):143–175, 2001.

[5] B. Dom. An information-theoretic external cluster-validity measure. In *UAI*, pages 137–145, 2002.

[6] F. Gelgi, H. Davulcu, and S. Vadrevu. Term ranking for clustering web search results. In *Proceedings of Tenth International Workshop on the Web and Databases*, Beijing, China, 2007.

[7] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 76–84, New York, NY, USA, 1996. ACM.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. pages 604–613, 1998.

[9] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 658–665, New York, NY, USA, 2004. ACM.

[10] D. Lee and S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, (401):788–791, 1999.

[11] S. Li, X. Hou, H. Zhang, and Q. Cheng. Learning spatially localized, parts-based representation. *CVPR*, (1):207–212, 2001.

[12] S. Osinski. Improving quality of search results clustering with approximate matrix factorisations. In *In the Proceedings of the 28th European Conference on IR Research (ECIR 2006)*, London, UK, 2006. Springer Berlin.

[13] S. Osinski, J. Stefanowski, and D. Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Advances in Soft Computing, Intelligent Information Processing and Web Mining, Proceedings of he International IIS: IIPWM 2004 Conference*, pages 359–368, Zakopane, Poland, 2004.

[14] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *ACM international Conference on Information and Knowledge Management*, pages 42–49, New York, NY, USA, 2004. ACM.

[15] P. I. Taher H. Haveliwala, Aristides Gionis. Scalable techniques for clustering the web. In *In Proc. of the WebDB Workshop*, pages 129–134, 2000.

[16] H. Toda and R. Kataoka. A search result clustering method using informatively named entities. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 81–86, New York, NY, USA, 2005. ACM.

[17] S. G. Yiping Zhou, Lan Nie. Surface form resolution based on wikipedia. In *To appear In Proc. of COLING 2010*, 2010.

[18] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of SIGIR 1998*, pages 46–84, 1996.

[19] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *SIGIR*, pages 210–217, New York, NY, USA, 2004. ACM.