Clustering for News Search Engines

Previously…

- We talked about the motivation behind vertical search engines, especially in the case of news
- A learning-to-rank approach of combining relevance ranking and freshness ranking was proposed in the form of demotion ranking,
- We introduced a different algorithm, the Joint Relevance Freshness Learning(JRFL), which attempts to use clickthrough stats and freshness features to learning a new model for ranking
- Today, I'll talk about a completely different method of presenting news articles in the form of clustering

News Clustering

- Clustering allows a quick unified view of search results by grouping similar documents together
- For news searches, it could be useful to group articles for a given query, since the number of related articles could be huge
- Clustering helps reduce the number of news articles to browse and can even solve the problem of redundancy for reposts
- Like before though, just because an article are related in content, there is a certain time component, or recency, to articles
- Going back to the earthquake example, we should make sure that articles about earthquakes at different times or places should belong to different clusters, since they are independent events

Working towards a good clustering method

- Previous work has shown that in order to get a good idea of an article, we need features deep within the document body
  - Includes things like phrase extraction
  - Might also be useful to take the query information into account
- On the other hand, quality descriptors aren't as useful in news search, since we only need one representative document for a given cluster
- One thing that is really useful for news articles is named entities and organizing clusters around them
- Processing each document for named entities in real time is slow
  - One way to get around this is offline clustering against the entire corpus

Architecture of the news clustering system

- Normally, we only need top-ranked search results for clustering
- What if user wants an article that is related, but not in top-ranked results?
- Use an offline batch processing of documents against the entire news corpus
- We'll also need a incremental cluster solution, since news is always updating
- First off, the offline clustering should be run on a regular basis, multiple times a day
- News corpus needs to be limited in scope
- When either the offline clustering or incremental clustering is done, we assign a cluster ID to each document
- Finally we can perform a real-time clustering using these cluster IDs to group the results and present to user

Architecture for news search clustering

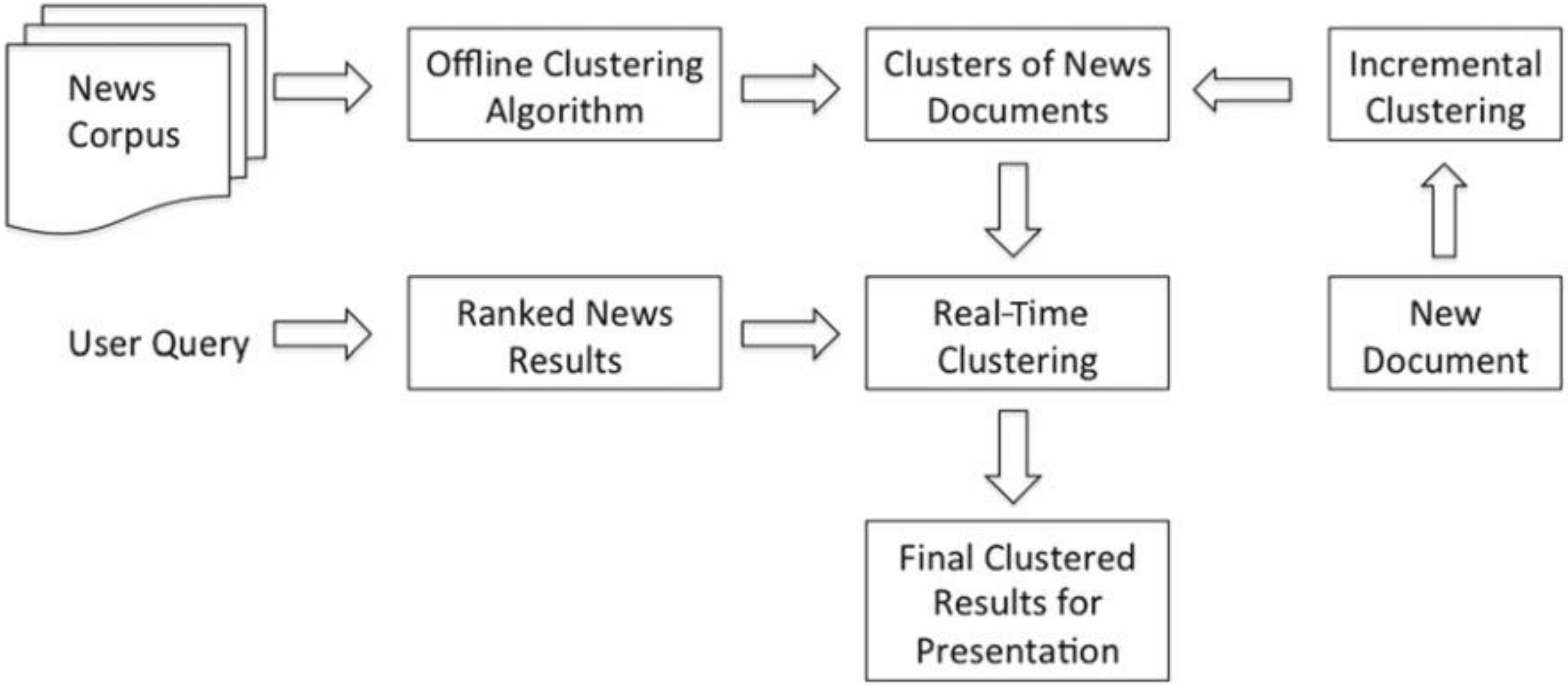

FIGURE 2.6

The architecture of the news search result clustering.

Offline Clustering

- In order to perform the offline clustering, we use *locality sensitive hashing*(LSH) between pairs of articles
  - A good similarity score is crucial to the success of the entire clustering algorithm
- Features for similarity:
    - Term frequency-inverse document frequency(TF-IDF)
    - Wikipedia topics – extract Wikipedia topics from article and assign an “aboutness” score
    - POS tagging
    - Also used presentation cues, such as words that are **bolded** or *italicized*
    - Time – compare the timestamps of two articles, t_1 and t_2, and take cosine similarity, weighted by $\exp(-|t_1 - t_2|/7)$
      - Assume news article are not further apart by more than a week
- Using this, we construct a similarity graph on the corpus using cosine similarity, where two articles share an edge if the similarity meets a threshold
- We also run a correlation clustering based on this similarity graph to get a final clustering

Minhash Signature Generation

- An alternative way to compare similarity is to compute Minhash signature of each article
- We assume that articles that have the same Minhash signature to be the same
- Using this, we can detect duplicates before the LSH procedure
- We can “cluster” duplicates together and select one representative to pass into LSH
- The cluster ID that it gets is assigned to all the other duplicate articles

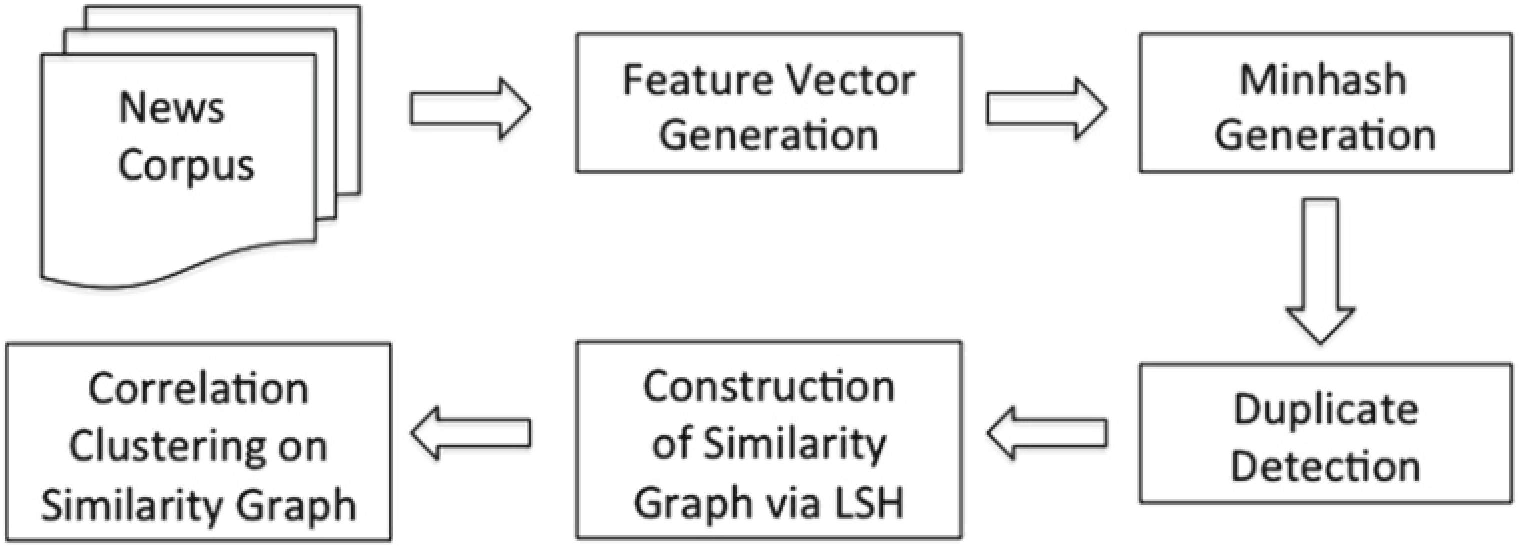

FIGURE 2.7

Overview of the offline clustering system.

Incremental Clustering

- Our offline clustering provides a base to work with
  - We already have a set of clusters
- For incremental clustering, we are assigning a new document to a cluster that it is most like to be associated with
  - However it is possible that the new article might not belong to any cluster
- We define 3 different classifiers for incremental clustering:
  - Static – standard classifier that must be retrained on the latest set of documents
  - Semiadaptive – Creating a new cluster for articles that are not “close enough” to the existing clusters
  - Fully adaptive – Not only able to create new classes, but also able to remove classes via merging
- Best solution depends greatly on how the offline clustering is already implemented

Real-Time Clustering

- So far we’ve outlined a way to group articles together into a class, or a story
- Say that we were to search “earthquake”, then we would should get a group of clusters with each cluster representing a different evet
- What if you were to search “chile earthquake”?
  - Instead of getting all earthquake stories, you might only want stories related to a particular Chile earthquake, the magnitude, news articles about damage, etc.
- Real-time clustering attempts to adjust this granularity by using the query as well
- Three methods of handling this task are explored

Meta Clustering and Textual Matching

- This method relies on our offline clustering output as well as matching text from the query and the documents
- The similarity measure used for this is:

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

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

Contextual Query-Based Term Weighting

- Here we assume that we have term vectors for each document which represents its context
- We want to weigh terms that are closer in proximity to the query terms higher
- Generally, we use the full query instead of bigrams or unigrams

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

- Using the new weights, we construct a new term vector for each document and use that to compute the similarity measure

Offline Clusters as Features

- Due to the computational load of real-time clustering, we need a fast and cheap solution
- One way is to leverage the work we've already done
  - Use the cluster IDs assigned from offline clustering

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

- Here, CosineSim is the cosine similarity on the bag of words for a document pair
- Jaccard is the Jaccard similarity measure between the vector of offline cluster IDs for two documents
- α is a tradeoff parameter between the two similarity measures, but just set to 0.5