Fintan

An Algorithmic Approach to News Aggregation

Alyssa Harding, Brian Ngo, Brian Steadman, Nina Liong

Fintan seeks to provide a news aggregation service similar to existing services like Digg[1] or Reddit[2], but augment existing user voting systems with algorithmic ranking. In addition, the service clusters similar news entries from various blogs together, providing a diverse view on a single news topic. We seek to present this in an intuitive user experience that users of other news aggregation services will instantly be familiar with.

Project Goals

The original goals of our project were three-fold. We wanted to create a way to rank stories algorithmically in order to augment user voting, generate relevant clusters of stories, and create an intuitive user interface for our product. Of the original goals, we were able to satisfactorily achieve two of the three. In the following sections we will describe the segments of our implementation pipeline, the problems we faced, and possible room for improvement.

Pipeline: Retrieving Blog Data

The first stage of the Fintan pipeline was to retrieve blog data from an external API, Spinn3r[3]. Spinn3r is a service operated by the company responsible for Tailrank[4] (another news aggregation service), where they make their blog crawling engine available to the public. Spinn3r actively crawls over 10 million blogs on the web, and provides a lot of interesting blog metadata with each blog entry. Using their open sourced Java API, we wrote a program that automatically retrieves blog data and converts it into Hadoop-serializable formats for further processing. Although they provide a Java API to access their services, several data points were missing from their implementation. Thus, we implemented a few of those features and will be contributing our changes back to the Spinn3r product in the near future.

Pipeline: Clustering

This stage of the pipeline uses the most computational resources and is also where we spent a lot of efforts in optimizing. The clustering pipeline is actually made up of four distinct steps: data preparation, suffix tree clustering, cluster selection, and aggressive cluster combination.

In the first phase the blog data from the previous pipeline stage is prepped for clustering and shrunk down to reduce data requirements further down the pipeline. Blog IDs, which usually contain the URL of that entry are hashed to a unique integer. A table of the integer to URL mappings is written out to the file system to aid in re-associating IDs with entry data later in the pipeline. Furthermore, the text

[1] www.digg.com
[2] www.reddit.com
[3] www.spinn3r.com, free for educational use
[4] www.tailrank.com

of the articles is normalized, removing all markup language, white space, capitalization, and punctuation besides periods, which are needed to mark the end of a sentence.

The second phase involved the construction of suffix trees. Suffix trees may be constructed with O(n) space complexity, where n is the number of unique suffixes, and O(m) time complexity, where m is the number of words in the corpus. We implemented the O(n) space complexity but implementing O(m) time complexity was seen to be too difficult and not worth the time. However, we still ran into numerous memory issues on our Mapreduce nodes, and thus had to introduce further optimizations which will be discussed in a later section.

The purpose of the third phase is to extract the top 500 clusters from our numerous trees. Clusters are defined as any node in the final tree. The metric we use to score a cluster is the number of articles in a certain node multiplied by the number of words in its n-gram minus one (this was to eliminate depth 1 nodes, which contain many IDs but have little value).

In the fourth and final phase the top 500 clusters are aggressively combined to form the final groups. The clustering is performed by going through the top 500 scoring clusters in descending order and comparing them to other clusters. Two clusters are said to be similar if the number of matching documents between them is larger than half the size of the largest cluster. Any given cluster in this list may only be combined into a group once, after which it is removed from the list of top clusters.

Pipeline: Ranking & SQL Conversion

After the blog entries are clustered, we assign an initial rank to each entry within a cluster based on the blog's tier (derived from the metadata in the crawl stage) and the date it was posted. This rewards blogs which have good reputations and which post articles in the cluster early. Each cluster is given a score based on the ranks of the blogs comprising it. To bridge the backend and the frontend, we then run a Mapreduce job to generate SQL statements which (when run) will put each entry and cluster into our MySQL database.

Pipeline: Frontend

Our frontend was written in Ruby, using the Rails framework. We use MySQL as our data store and serve up pages with the Mongrel HTTP server. AJAX was also used when appropriate to create a better user experience. The aim of the frontend UI design was to provide a simple but also familiar experience when compared to other news aggregation sites. We made numerous design choices to improve usability and increase the visibility and effectiveness of our main content.

Algorithmic Choices

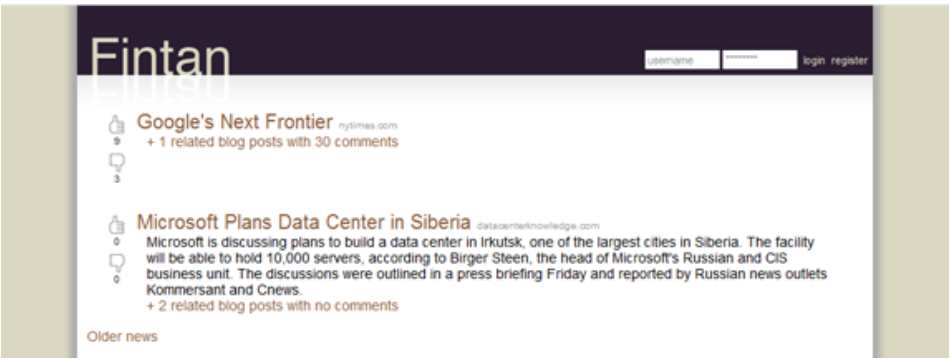
We made several choices in algorithms when implementing Fintan. To address the issue of how to cluster our blog entries, we decided on using suffix tree clustering instead of k-means clustering. Research done in information retrieval has shown that suffix tree clustering has higher average precision than any other text document clustering algorithm with O(n) runtime (Zamir & Etzioni, 1998). Ironically, because of problems we ran into with STC, we also implemented a K-Means clustering algorithm as well

as a Naïve-Bayes classifier in order to alleviate the issues we were having with memory. Lastly, in addressing the issue of ranking entries within clusters, we chose to create a custom ranking algorithm to initially reflect the blog's relevance and the date entries were posted, and then incorporate user votes.

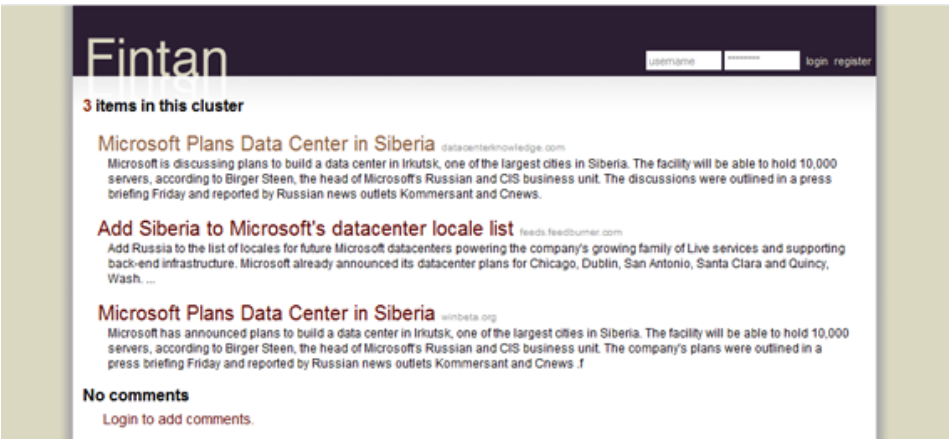
Screenshots and Use Cases

In order to keep the brevity of this report, we'll briefly describe what a user can do on the Fintan site and provide a few screenshots of the user experience. We encourage the reader to try a live demo at http://www.brianngo.net:3001.

The best clusters are displayed first on the Fintan front page. Each cluster has the best post's title as its headline. Clicking on the headline will take the user to the original site where the blog post is from.



The user clicks the link underneath the cluster to view the other posts.



If the user likes the cluster, he goes back to the front page and votes up, or down if he feels that the cluster is irrelevant.

Although not pictured here in the report, the user can also post comments on clusters and view their previous voting/comment history.

Surprises and Challenges

We had a large number of challenges that we encountered during the project:

- **Spam**. Not only was the high level of noise from blogs a huge issue, but a large number of the blogs we retrieved from Spinn3r were loaded with spam. Filtering out the spam would've been a complex project itself, so we attempted to do our best to work around it using naïve filters.
- **Cluster size**. We faced two major problems in clustering: data size limitations and quality of clusters. We had to make a decision on which to work on as the solutions to each required a non-trivial amount of development and both solutions could probably not be implemented by the deadline. We chose to try to increase the amount of data we could run through the clustering; reasoning that by the nature of our clustering this would also increase the quality of our clusters. We tried several strategies to fix this problem including removing stop words and truncating articles to the first few hundred words. We finally managed to solve our data problems by coming up with a splitting strategy for our suffix trees. Our splitting techniques for STC are described in the technical appendix.
- **Cluster quality**. However, this did not yield the improvements to cluster quality that we had expected. We found that while articles were being clustered intelligently it was not in the way we had anticipated. Rather than being grouped by content, similar writing styles ended up being the dominating relationship of our clusters. Because of this, many times the articles of a blog would be clustered with each other as the editors use similar language across posts and many blogs contain a signature which appears in each post.
  Interestingly enough, the same issue happened when we implemented a K-Means clustering algorithm to cluster blog entries together. Articles of the same writing style would consistently be clustered together.
- **Classification**. In an attempt to solve our cluster size problem, we attempted to write a classifier to classify blog entries into broad categories, essentially splitting our input data set into portions. We ran custom web crawls of selected portions of the DMOZ project and trained Naïve-Bayes classifiers on the content. However, the best accuracy our classifier was able to achieve was slightly below 60%. Even after attempting to augment our Naïve-Bayes with boosting or bagging, we weren't able to produce a good result set out of our classifier.
- **Ranking**. We expected to use PageRank to assign an initial rank to each of the blog entries that we encountered. PageRank, however, requires a densely linked corpus. When we examined our corpus, we discovered that the blogs rarely linked to one another and had to create another method of initially ranking the entries.

Experiments

Throughout development of our clustering algorithms, we had a small test set of hand selected blog entries known to cluster to five different natural clusters. They ranged in topics from the latest tech news on Windows Vista to the NIE government report. We would run the small test set through our clustering algorithms and investigate the output to see how well it did with our informal metric. The metric we used was to see how many of the natural clusters our clustering algorithms were able to

identify. Sometimes the clustering algorithms would output more than the number of natural clusters that existed, but we would ignore the extra clusters and didn't count that against the algorithm.

We had two clustering algorithms that were implemented: suffix tree clustering and k-means clustering. Of the two, suffix tree clustering performed better and was able to identify more natural clusters than the k-means algorithm. However, STC also produced a lot of "noise" clusters. For example, with an input data set of 5 natural clusters, it would sometimes output upwards of 30 clusters, with at best 3 clusters being actual natural clusters (though usually they were missing an entry from the natural cluster or had an entry from another cluster – if the majority of the posts were of the same natural cluster we counted it as a success).

With k-means clustering, we could specify the number of clusters it should output, thus reducing the noise. However, it frequently failed at identifying the correct number of natural clusters and very often clustered blog entries according to writing style rather than content. The success rate of k-means was much lower, with only 1-2 natural clusters identified at best.

After analysis of our experiments we concluded there are several ways to improve both algorithms. In regards to the STC algorithm, we believe that using methods that disallow a blog clustering with itself would help. Also, giving higher weightings to proper nouns in the suffix tree construction would help us get past the writing style barrier. In regards to the k-means clustering algorithm, we believe that using a different distance measure would help immensely. We didn't have time to implement a tf-idf distance measure but that may be a great choice considering our input content.

Conclusion and Future Work

We are incredibly proud of our work on Fintan and the amount we have accomplished in the quarter we are able to work on it. Working with data from the blogosphere is incredibly challenging, and we've come a long way from the beginning of the project. Each of the group members has learned an incredible amount and it was an invaluable experience for all of us. Had we more time, we would invariably focus on improving our clustering algorithms, implement a solid classifier, and tackle the problem of addressing blog spam. Lastly, we would love to transition our code into a more cohesive environment, where the backend processing done by Mapreduce could easily communicate with our frontend framework. Currently the project runs in many separate pieces due to security issues with the educational Mapreduce cluster as well as our "production" server. In the end, we believe this project was a great success in many aspects.

Appendices

Included here are more detailed technical descriptions that we left out of the main report, as well as logistical details of our project.

Technical Appendix

Splitting suffix trees

A big difficulty we overcame was figuring out how to appropriately split up the trees so that they could be held resident in memory. Two to three days of blog entries amounted to approximately 200mb of data. The final suffix tree representing this data amounted to 300GB of data, or a 1500x increase. This final tree needed to be held completely in memory during the reducing phase. We finally observed that when two trees are merged that branches starting with a different first word would never combine and therefore are independent of each other. Seeing this we implemented a splitting mechanism on the tree where the hash codes of the first word in each branch was used to determine which reducer it should go to. This breakthrough allowed us to stop running on trivial <2mb data sources, and move onto the full data sources by splitting all trees into more than 7000 smaller trees.

Group Responsibilities

Group members worked together on a lot of the aspects of the project, but we attempted to have each member "own" or be responsible for a portion of the project. Here is how the responsibilities break down.

- **Brian Steadman**: Wrote almost the entire suffix tree clustering code.
- **Alyssa Harding**: Helped on STC, wrote the ranking & conversion part of the pipeline.
- **Nina Liong**: Frontend coding and designed the database schema.
- **Brian Ngo**: Project management, data retrieval code, k-means & classifier code, experiments.

Credit

We would like to give credit to the Spinn3r team for providing a great service. We also incorporated some code from the WEKA project for more aggressive stop word identification in our k-means clustering implementation. Lastly, we used Mallet for our Naïve-Bayes classifier.

Obtaining and Using Fintan

A demo of Fintan is up at http://www.brianngo.net:3001. You can also obtain a copy of our entire codebase via SVN at http://svn.brianngo.net/fintan. Use the username and password of "uwcse" and "zrxlx" to gain access. Almost all the parts of Fintan are designed to run separately, so there is no single command to run them all. We frequently ran our jobs within the Eclipse environment, and we advise that you do the same if you are interested in running any of the Mapreduce processes. Before running them please make sure that all of the paths are correct in the related job files. We have the paths hardcoded in the code since editing program arguments in Eclipse isn't as convenient as it should be. A

word of caution: running a job can take upwards of four hours for clustering a small data set. Getting data from the Spinn3r API can also take several hours, as well as saturate your internet bandwidth (also, please don't run it more than once as this might alert the Spinn3r team that the UW CSE API key is being abused). For any questions on running the Mapreduce jobs, contact brianngo@cs.

Running the Ruby on Rails server is much simpler, provided that you have the correct version of Ruby (1.8.6), Rails (1.2.5), and Mongrel (latest) installed. Additionally, you'll need MySQL and set up the databases to be accessed by the user specified in the Rails database schema file. After that it's as simple as running the `script/server` command from the frontend directory.