CASE WESTERN RESERVE UNIVERSITY EST. 1826

Project October

Read news that you want to read

Authors:	Tom Dooner, Mika Little, Brian Stack
Project:	Project October
Date:	March 23, 2013

Contents

List of Figures

1 Abstract

In modern news aggregation services, such as Reddit, Slashdot, Digg, and Hacker News, longtime members report noticing a marked decrease in quality of discourse as the services gain mainstream attention. This gradual but irreversible decline heralds back to the newsgroup era, when new college students would log in for the first time come September, causing an influx of new users and subsequently diluting discussion upon the sites which they joined. As an example, after AOL opened newsgroups to the masses, the community standards continued to devolve, according to newsgroup veterans[5]. This gradual decline of content quality due to large numbers of new users came to be known as "Eternal September", being "Eternal" as nowadays, the influx of new users is no longer restricted to September, but now persists through all months of the year.

The inevitable loss of the longtime members further perpetuates the problem, and causes large numbers of excellent contributors to feel out-of-place in their own community. We aim to engineer a service that uses technological principles to avoid this, thus improving the user experience and allowing a large community to benefit from thoughtful discourse and interesting articles.

2 Project October: Hybrid Recommender System

Due to the ever-growing amount of information available online, the need for a highly developed personalization and filtering system is growing significantly. Recommender systems constitute a specific type of information filtering that attempt to present items according the interests expressed by a user[1]. Most web recommenders are employed for e-commerce applications or customer adapted websites, which assist users in decision making by providing personalized information [2], but the same techniques that suggest related items on e-commerce websites can recommend news articles to users as well. We believe Project October is the first attempt to apply recommendation techniques to social news aggregation.

2.1 Background

It is our hypothesis that the recommendation of news sources will provide a scalable community experience that can be tailored to each person's interests. Providing an automatic, customized, recommendation of articles will prevent the community from being diluted by new users and thus prevent the Eternal September phenomenon. Each user will have their own viewing environment curated for them, with different mindsets and preferences forming their own communities in which like-minded individuals can partake in discussion, eliminating the cross-contamination of user bases.

This proposal discusses an implementation of a hybrid collaborative and content-based filtering approach for a web-based recommender system ("October"). In particular, we will be linking various news sources and user submitted sources. The resulting network of user-item relations and associated content features is converted into a unified mathematical model, which is applicable to our underlying neighbor-based prediction algorithm. By means of various experiments, we demonstrate the influence of supplementary users as well as item features on the prediction accuracy of October, our proposed hybrid recommender. In order to decrease system runtime and to reveal latent user and item relations, we factorize our hybrid model via singular value decomposition (SVD).

For the development and evaluation of our proposed hybrid recommender system, we make use of various news outlets and user recommendations, importing them into a graph database. Both corpora are joined in a unified mathematical model, which describes the complex network of interdependencies.

2.2 Intended Audience

We intend Project October to be open to the public. At launch, we expect the user base to be somewhat technical in nature, being some of the Internet's power users.

3 Application

3.1 Frontend

The frontend will be a standard social news aggregator. Upon going to the homepage, new and logged-out users will be presented with a splash page that describes the nature of Project October and offers a registration link. Users can register for the application by selecting a username and password.

When logged in, users will be presented with their personalized content. The design is modelled after the front page of a newspaper – articles that are evaluated to be more interesting to a user will be placed in more prominent positions while articles that are deemed less relevant to the user's interest will fill the side columns, be located further down the page, and occupy less space. Users will be able to click on the headline (or associated image, if existent) to be taken to the original news source.

Each news article will feature a link to view the associated comments as well as an icon (located in the corner of the article text/image) displaying the credibility (“cred”) a given article possesses. This is analogous to upvoting in Reddit, except that giving an article cred on October will inform the recommendation engine of your individual preference, rather than directly impacting the weighting of an article by a predefined formula. Therefore, cred serves two purposes: 1. To inform other users of the quality of the article and 2. To inform the recommender system of the user's interests.

Users will be able to click a link on the homepage which takes them to an article submission form. The submission form will request a URL and a headline for that news story. Upon entering the URL, the news story will be scraped in the background using a web scraping tool which will attempt to extract properties about the page automatically – relying upon the ¡meta¿ tags and heuristics to determine the article's body text.

3.2 Backend

The backend will contain a recommender system, which will recommend articles for users with the following technique. When an article is submitted from the frontend (via the API – see Section 3.3), we will peform TF/IDF[7] on the full body text returned by the scraper. The weighted terms returned by TF/IDF will be stored in a document database with a custom-maintained inverted index of the terms to which articles contain them. This will allow efficient access of articles about arbitrary terms.

When a user upvotes, downvotes, reads, or comments upon an article, the user's document will be modified to reflect that the user either cares more (in case of upvotes, comments, or reads) or cares less (in case of downvotes) about the terms associated with that article. When the frontend requests more articles via the API for that user at a later time, the new weights will be used, resulting in a different, more-tailored selection of news articles.

The backend and the frontend will share access to the frontend database, which will contain all persistent information about articles such as their headline, image thumbnail URL, and other associated metadata. The backend will have read-only access to the frontend database. The backend will have exclusive control of a document database which tracks information about users and article terms.

3.3 Frontend/Backend API

October will employ an API to promote separation between the frontend and the backend recommender system. This will provide a clear interface and facilitate easy simultaneous development of both parts of October.

To implement the API, we will use Apache Thrift, an Interface Description Language[6]. The essence of the API is simple, featuring primarily two types of calls:

Give n recommendations for user u This is the main output from the backend, returning recommended news stories or comments for a given user. Ancillary parameters will be added to this to facilitate the frontend placement of articles, e.g. the recommendation confidence and individual article weightings.

User u took action n This is the main input to the backend, allowing it to adjust recommendations according to user action. The parameters to this API call can be of many types. For example, "User commented on article $\#n$", "User gave cred to comment $\#n$", and "User visited link $\#n$" are all valid parameters for this API call.

These two calls manifest themselves in Thrift with a few simple object definitions and an interface. The abbreviated version `0.1.0` is attached as Appendix A.

4 Methodology

4.1 Project Management

We will manage the project using agile development methodologies with a focus on progressive elaboration and documentation for the purposes of this project.

4.1.1 Agile Development

In order to stay within the bounds of Agile Development while still maintaining the requisite documentation, we will be creating documentation at the end of each release. In order to schedule work, we have employed Pivotal Tracker, an online task management solution. From the queue, we will assign tasks as we finish previous tasks. The work naturally fits into two main categories – frontend and backend. Work relating to each category will be assigned to the same people until we make enough progress to cross between teams at will. At least until that point, Brian will work on the backend while Tom and Mika create the frontend.

4.1.2 Scrum

We will meet every Monday, Wednesday, and Friday at 3:55pm to have a short meeting to discuss the current progress, any stumbling blocks, and what work is coming up in the immediate future.

4.1.3 Release Schedule

Releases are bi-weekly and will include a team review as well as planning for the next release. Release dates will involve pushing the current state of the project to a continuous integration service (http://travis-ci.org), after which automated deployment will be executed.

4.1.4 Iterations

We plan to have weekly iterations. At the end of each iteration, team members will re-sync and make sure that we have a clear outline for the next iteration.

5 Software Design

5.1 Intended Audience

We intend Project October to be open to the public. At launch, we expect the user base to be somewhat technical in nature, being some of the Internet's power users, however, October should support the registration of any internet user.

5.2 Homepage Requirements

The are two cases for the homepage: logged in users and logged out users.

Logged out users should arrive at the homepage and be presented with a splash page of sample content and a description of October's features. There should be a registration link so that interested users can register for their own account.

Logged in users should, upon landing on the homepage, be presented with a display of news articles. The display will be pictures (when available) along with the headlines for each article. Each article should be accompanied with links to positively influence the article's recommendation rank, to negatively influence the article, and to comment on the article.

5.3 User Creation/Login/Display Requirements

Internet travelers should be able to create accounts on October easily by providing minimal information – simply their desired username, their email, and a password. After creating a user, an email should be sent to the user welcoming them to October and containing a confirmation link to confirm their email address.

When a user first uses October, we don't know anything about that user's interests. Users should be shown various sample topics until the recommender is confident enough in their browsing habits to hazard a recommendation.

Every mention of a user's name (i.e. the credit for who posted an article or comment) should be a link to a user profile page. More detail about the user profile page is given in Section 5.4.

5.4 User Profile Page Requirements

Each user has a profile page containing information related to that user. The profile page contains a list of comments that the user has made on posts and a list of articles the user has posted. All information on this page is public, as it is merely a summary of content available elsewhere on the public website.

5.5 User Profile Edit Page Requirements

In contrast to the profile page, which is a summary of a user's public activity, the profile edit page allows a logged in user to edit their own information. Settings such as their email address and password can be changed here.

The other significant part of the user profile edit page is a section which shows the user their own recommendation settings. For example, if the backend has decided the user is in a certain clique, the frontend's user profile edit page should convey that information. Sometimes, the recommendation algorithm might be less accessible than a easily-named clique, so those inferences cannot be displayed.

Furthermore, there should be a field where a user can submit additional categories they wish to voluntarily join. If there is such a category, they will be joined to it when they submit the form.

5.6 Article Posting Requirements

Logged in users should be able to click a prominent link on the homepage to submit a story. The submit story page will ask for the link to the desired story, and show the results of scraping that story's contents (see Section 5.9 for more about scraping). If the post looks good to the user, he or she can click a "Post!" button, which will persist the news story in all appropriate databases.

5.7 Article Page Requirements

The article display page is a page dedicated to a posted news article. Every news article will have an associated article page. The intent of the article page is to facilitate sharing of opinions around the posted news story through comments.

A user arriving on an article's page arrives at a page which contains the title of the news article along with a threaded comment list. Users can interact with comments in the same way as they interact with news articles – voting up, voting down, or commenting on them (in which case a child comment is created). Additionally, comments will have a field where they can be tagged with adjectives. These tags will be submitted through the API and reflect the ordering of comments for different users.

5.8 Frontend ↔ Backend API Requirements

Since the frontend and backend are divided with a Service Oriented Architecture (with Apache Thrift bridging the gap[1]), there must be an API between them.

The API should be as simple as possible to maintain the functionality of October. This is not a public API, so it needs to be robust to volume but we do not need to implement API endpoints for things that users might like.

The API is defined in a Thrift definitions file in the project-october-api repository on GitHub[**?**].

5.9 Web Scraping Requirements

When a user submits a link to a news article, October needs to know more about that article than its URL: the title of the article, any associated images, as well as the article's full text (for analysis purposes). We will employ a library which, given the URL to a webpage, will perform calculations and extract the main text of the article as well as make recommendations on what image might be the most relevant image on the page. The requirements of the web scraper are intended to allow the backend's NLP functionality to process the raw text of the article easily. It executes in three phases:

1. Document Cleaning
2. Content Extraction
3. Document Cleanup

Ultimately, this entails cleaning out junk and comments, grab the cluster of relevant text, attempt to determine the best image, and return these. Specifically the image extraction was troublesome, as Java's image functions were completely unreliable, which was resolved by using ImageMagick.

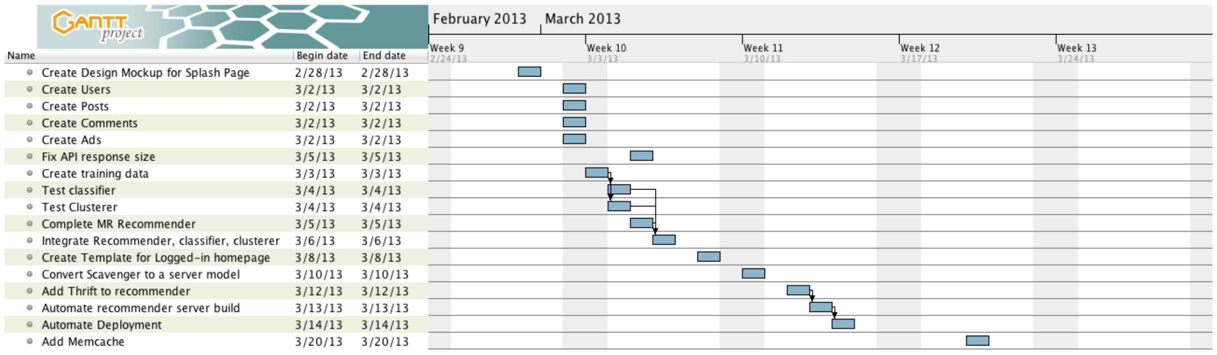
5.10 Backend Requirements

The backend is a recommender black-box which provides suggestions via an API to the frontend. The actual implementation of the recommendation service is arbitrary, only that it implements the services of the API effectively. For more about the implementation of the backend, consult Section **??**.

6 Project Management–Administrative Details

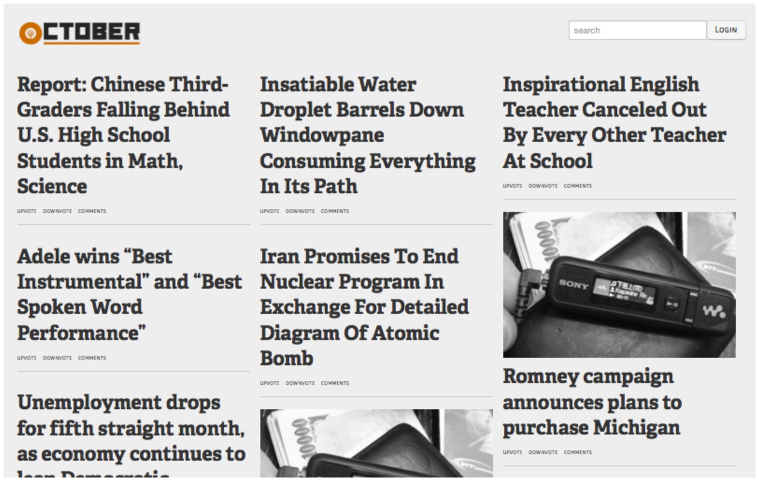
We have split the team into two parts, frontend and backend. With respect to the frontend, Tom and Mika have worked on the development and design respectively. The difficulty here is to determine how to represent the news in such a way that the user will be inclined to continue using the service. As for the backend, Brian and Raja have broken up the service architecture and the recommender tasks apart respectively. The difficulty here was to determine the best method of classification and clustering, while the graph traversal could be done via a classical method. To accommodate for the design changes in the frontend and the refinement of the recommender on the backend, the intended first release has been delayed by a 1.5 weeks. Despite this, the rest

[1]See Section 3.3 for more about Thrift



Name	Begin date	End date
Create Design Mockup for Splash Page	2/28/13	2/28/13
Create Users	3/2/13	3/2/13
Create Posts	3/2/13	3/2/13
Create Comments	3/2/13	3/2/13
Create Ads	3/2/13	3/2/13
Fix API response size	3/5/13	3/5/13
Create training data	3/3/13	3/3/13
Test classifier	3/4/13	3/4/13
Test Clusterer	3/4/13	3/4/13
Complete MR Recommender	3/5/13	3/5/13
Integrate Recommender, classifier, clusterer	3/6/13	3/6/13
Create Template for Logged-in homepage	3/8/13	3/8/13
Convert Scavenger to a server model	3/10/13	3/10/13
Add Thrift to recommender	3/12/13	3/12/13
Automate recommender server build	3/13/13	3/13/13
Automate Deployment	3/14/13	3/14/13
Add Memcache	3/20/13	3/20/13

Figure 1: Project October Gantt Chart



Report: Chinese Third-Graders Falling Behind U.S. High School Students in Math, Science

Insatiable Water Droplet Barrels Down Windowpane Consuming Everything In Its Path

Inspirational English Teacher Canceled Out By Every Other Teacher At School

Adele wins "Best Instrumental" and "Best Spoken Word Performance"

Iran Promises To End Nuclear Program In Exchange For Detailed Diagram Of Atomic Bomb

Romney campaign announces plans to purchase Michigan

Unemployment drops for fifth straight month, as economy continues to

Figure 2: Project October Homepage, pre-Alpha (Headlines courtesy The Onion)

of the tasks have been on schedule and the completion of these tasks place the projected alpha release within the next weeks. Ideally a beta release with the majority of bugs and a fairly accurate recommender service will be available in early to mid April.

7 User Interface

The user interface for the homepage and other pages will surely evolve over time.

The homepage is an especially crucial interface to get right. We mimic a newspaper feel with articles staggered in a rough column grid. This allows readers to be presented with many recommendations of different weights simultaneously. A pre-Alpha screenshot of the homepage is attached in figure 2.

The user interface for other pages will be in the same general content area, however there are no further screenshots to share at this time.

8 Testing & Evaluation

Testing and evaluation will be performed as we progress through the project. Currently, only very minimal testing is implemented.

9 Project Progress

The project is progressing satisfactorily. We are attempting to follow Agile Design Patterns, and we have been effective at meeting for short periods frequently. On the frontend, we have completed the user creation + login processes, mocked up the homepage, and connected the Thrift library. On the backend, we have implemented the API so it responds to requests from the frontend. The recommendation engine is still in-progress. We remain committed to maintaining team communication via scrum and continue working.

9.1 Discussions & Conclusions

Overall, the project is still on schedule. We still intend to complete a fully-featured product on-time.

10 Appendix

References

[1] Gediminas Adomavicius , Alexander Tuzhilin, *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*, IEEE Transactions on Knowledge and Data Engineering, v.17 n.6, p.734-749, June 2005 [doi¿10.1109/TKDE.2005.99]

[2] Greg Linden , Brent Smith , Jeremy York, *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*, IEEE Internet Computing, v.7 n.1, p.76-80, January 2003 [doi¿10.1109/MIC.2003.1167344]

[3] Jensen, V. *Bayesian Networks and Decision Graphs*. Springer, 2001

[4] Arthur Asuncion, Padhraic Smyth, Max Welling. *Asynchronous Distributed Learning of Topic Models*. NIPS 2008.

[5] `http://en.wikipedia.org/wiki/Eternal_September`

[6] `http://en.wikipedia.org/wiki/Apache_Thrift`

[7] `http://en.wikipedia.org/wiki/Tf-idf`

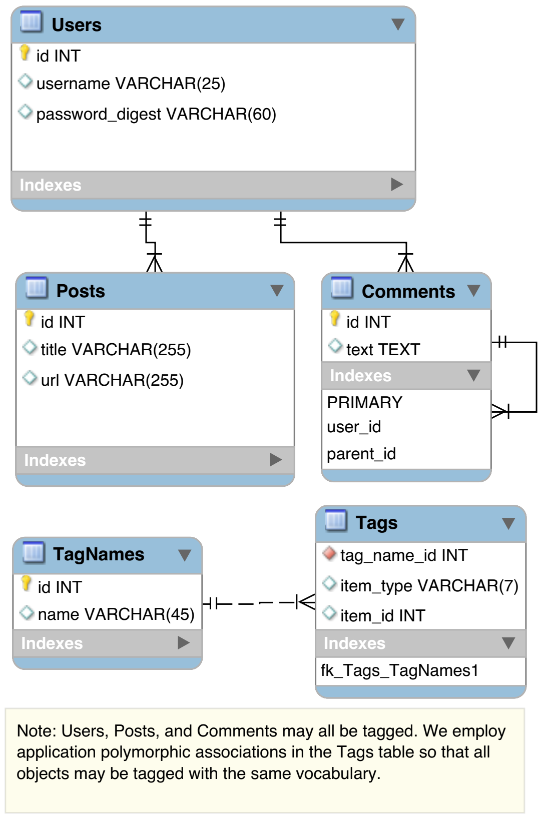
A Thrift API Definition

```
#
# Structs
#

/** A single post with its calculated weight.
 * @param post_id, the unique id of a post.
 * @param weight, the "importance" of the post to the querying user [0,1].
 */
struct Post {
    1: required i64 post_id,
    2: optional double weight,
}

/** A list of posts along with a confidence for the accuracy of the list.
 * @param confidence, the confidence in the results.
 * @param posts, a list of Posts.
 */
struct PostList {
    1: optional double confidence,
    2: required list<Post> posts,
}

/** The types of actions that can be performed in a triple (subject, verb, object) */
enum Action {
    READ,
    VOTE_UP,
    VOTE_DOWN,
    VOTE_NEGATE,
    POST,
    COMMENT,
    REPORT,
    TAG
}

#
# Exceptions
#

/** The queried object does not exist. */
exception NotFoundException {
}

/** There was an error processing the request */
exception EngineException {
    1: required string why,
}

/** Request took too long to process */
exception TimeoutException {
}

#
# Service
#

service Recommender {

    /** Test for connectivity */
    string ping() throws (1: TimeoutException te),

    /** Request a list of posts that are most appropriate for a user
     * @param user_id, the user that the posts are being requested for
     */
    PostList recPosts(1: required i64 user_id) throws (1: NotFoundException nfe, 2: EngineExcepti

    /** Alert the recommender that a user has actioned a post
     * @param user_id, the user that performed the action
     * @param verb, the action taken (this is from the Action enum)
     * @param post_id, the post that the action is being performed on
     */
    void user_v_post(1: required i64 user_id, 2: required Action verb, 3: required i64 post_id) t

    /** Alert the recommender that a user has actioned a comment
     * @param user_id, the user that performed the action
     * @param verb, the action taken (this is from the Action enum)
     * @param comment_id, the comment that the action is being performed on
     */
    void user_v_comment(1: required i64 user_id, 2: required Action verb, 3: required i64 comment
}
```

B Database Design

B.1 Frontend Database

Please see Figure 3.



Note: Users, Posts, and Comments may all be tagged. We employ application polymorphic associations in the Tags table so that all objects may be tagged with the same vocabulary.

Figure 3: Frontend Database Relational Diagram

B.2 Backend Database

C User Manual

D Programmers Manual