



Nüwslì

An Intelligent Mobile News Reader

Master Thesis

June 2010 – November 2010

Student: Mihai Calin

ETH – ITET Master Studies

Supervisor: Prof. Dr. Roger Wattenhofer

Co-Supervisor: Prof. Juliana Sutanto

Advisors: Elia Palme / Michael Kuhn / Samuel Welten

Abstract

The way people read the latest news has changed through the Internet but the way people find interesting news is still by browsing headlines. Systems capable of recommending news to users just start to become popular for online news providers, but for mobile users they are still at the start, especially for the Swiss market. This thesis describes the implementation of such a news providing service that uses several news publishers as content providers and is able to recommend news to users, all through an iPhone application. A detailed introduction into state of the art recommender systems implements a prototype and suggests improvements based on the results of the test phase. A highly modularized server structure is presented, which supports the content provisioning of the iPhone application.

Chapter Overview

Introduction

Kicks off the discussion about current web based or mobile newsreaders and the need for a recommender system that would make suggestions to users about articles they would be interested in.

Related Work

Presents main advantages and disadvantages of solutions that are already on the market. Both web based or mobile newsreaders are presented and also their ability to recommend news based on the users interests is analyzed.

Solution Debate

Consists of a non-technical discussion about the main features a mobile newsreader has to provide and what reasons encouraged the main architectural decisions. The principle of the server is briefly presented together with the main features of the mobile application.

Architecture

While the Solution Debate chapter sketches the structure and functionality of the application, the Architecture chapter goes into details and presents the technical solutions used for constructing the system. Split in two parts, the chapter presents the main problems that arose while implementing the server and client sides.

Recommender

Presents state of the art recommender systems and algorithms through a theoretical approach before proposing a news recommending framework that is customized for the application's needs.

Conclusions

Takes a look back at the achievements of the current system, summarizing the main factors that can make this project successful.

Future Work

Debates about features that could help the system reach higher goals in terms of user satisfaction and academic importance. The improvement possibilities for the recommender system are an important factor in planning the evolution of the project.

Table of Contents

Abstract	II
Chapter Overview	III
Table of Contents	IV
1 Introduction	1
1.1 Nomenclature.....	1
2 Related Work	2
2.1 Web based News Readers.....	2
2.2 Mobile News Readers.....	2
2.3 News Recommender Systems	3
3 Solution Debate	3
3.1 Approach	3
3.2 Implementation	4
3.2.1 <i>News Acquisition</i>	4
3.2.2 <i>Presenting News</i>	6
3.2.3 <i>Personalizing Content Provision</i>	7
3.3 Known Restrictions.....	7
4 Architecture	8
4.1 Server Side Development.....	8
4.1.1 <i>Introduction</i>	8
4.1.2 <i>Stage1 Crawler</i>	9
4.1.3 <i>Stage2 Crawler</i>	10
4.1.4 <i>Stage1 Recommender</i>	11
4.1.5 <i>Provision Services</i>	11
4.1.6 <i>Stage2 Recommender</i>	12
4.1.7 <i>Database Structure and Logging</i>	12
4.2 Client Side Development.....	13
4.2.1 <i>Introduction</i>	13
4.2.2 <i>Implementation Details</i>	13
4.2.3 <i>User Interface</i>	14
4.2.4 <i>Personalization</i>	15
4.2.5 <i>Retrieving Content</i>	15
5 Recommender	15
5.1 Introduction to Recommender Systems.....	15
5.2 Related Work	17
5.3 Premises.....	19
5.3.1 <i>Data Type Analysis</i>	19
5.3.2 <i>Available Ratings Data</i>	19
5.4 Proposed Solution	20
5.4.1 <i>The Item Based Recommender</i>	20
5.4.2 <i>The Updates Recommender</i>	22
5.4.3 <i>The Random Recommender</i>	22
5.5 Solution Evaluation.....	23
5.5.1 <i>Introduction</i>	23
5.5.2 <i>Experimental Setup</i>	23
5.5.3 <i>Results</i>	25
5.6 Known Limitations.....	27
5.7 Future Development.....	27
6 Conclusions	28
7 Future Work	28
8 Bibliography	29

1 Introduction

While the Internet as information provider has evolved, much has not changed in the way people get informed about the latest news. Even if they go online instead of buying a newspaper, they still only go to websites of acquainted publishers and try to browse the information manually in search of an interesting topic. Wouldn't it be nice to have a system that actually proposes news that will be of interest to each user in particular, and not to the average reader? Several online news providers already implement recommendation algorithms to help the users find the information that suits their preference, but how about having such a system integrated in your iPhone and to access it from everywhere, at any time?

The solution presented by this paper can be described through the simple usage scenario in which a person gets into the bus on his/her way to work. While he/she has some minutes to himself/herself, he/she would like to read about the events of the current and past days, so he/she opens an iPhone application that displays a list of topics interesting to him/her. The key here is the short time he/she has at his/her disposal which is probably not enough to browse or search for interesting titles – they have to pop into the display automatically, awaiting to be read.

This thesis proposes a system consisting of two components (server and client side) that are able to bring news article suggestions on your iPhone. As there is no such solution for the Swiss market, this project has the chance to become a great success. To compute the suggestions, a recommender system is developed that combines information obtained from hundreds or even thousands of users and uses this information to find similarities between users. From here to suggestions there is just a single step: one will be interested in content that people with similar taste also found interesting.

The chapters of this thesis first present the current market for such a solution and then debates about ways of implementing an easy to use but effective application. After a thorough discussion about the technical implementation of both the server and the client side, a prototype news recommender algorithm is proposed and compared to other state of the art recommenders. The conclusion sections mainly presents ways to improve the recommender system through different strategies described in the literature.

1.1 Nomenclature

The following nomenclature will be used throughout the paper: “A *publisher* provides *articles*, in printed form or online, that write about *events*. If the same *publisher* writes a second *article* about an *event*, this is called an *update*. If there are *updates* to an *event*, the respective *articles* form a *news thread* about that *event*.”

2 Related Work

2.1 Web based News Readers

More and more people turn towards the Internet to keep informed about the happenings in the world. The publishers that have proved to be trusted providers for the printed or broadcasted media are also the top choice when it comes to online news: people that are used to read NZZ or watch the newscast on BBC will most probably want to read news from the websites of these publishers. The advantages of choosing already known publications are obvious: one is already used to the journalistic style of the publication, it's objective or biased point of view, it's location based news etc. Hence, on the Internet one can find small scale and geographically relevant publishers, rather than online news giants.

A ranking of eBizMBA¹, a publisher specialized in web businesses, shows that Yahoo is the leading online news provider, followed by CNN, MSNBC, Google News and The New York Times. The important news aggregators found in the top are Google News and Digg, the other publishers representing well-known news agencies.

Google News is an example of an aggregator that attracts users worldwide. However Google News is country-specific and hence provides the same geographically relevant news as other local newspapers. Moreover, the articles presented on Google News are a selection of articles published on websites of local publishers. For an event, Google tries to group together all articles related to that event and present the title and a few lines of text for each article. Clicking on an article opens the website of the publisher rather than a Google-version of it – copyright protection is one possible reason for choosing this strategy.

You can review later updates here:

- ▶ As it happened: Mumbai attacks - 28 November
- ▶ As it happened: Mumbai attacks - 29 November

Figure 1 - BBC News website, screenshot of a portion of an article

Updates to news threads are not as transparent on Google's News site as it is on sites like bbc.co.uk, where you have a clear overview of the timeline (Figure 1). Google's approach is to group together all articles

related to an event, regardless whether they are from different publishers or different updates by the same publisher.

2.2 Mobile News Readers

Web based newsreaders would strictly speaking also count as mobile newsreaders, because modern smartphones can easily display desktop-designated web content. Google News (among many) has a mobile-device optimized online version, which makes it a serious competitor to apps that specialize in delivering news to mobile users. On the iPhone platform, however, it has evolved to common practice to have a specialized application for accessing

¹ <http://www.ebizmba.com/articles/news-websites>

online content of different websites, rather than navigating to the website with the web browser.

The Applestore provides dozens of apps for reading news on the go, both for iPhone and iPad. Most of them are publisher-specific applications that display the content of only one online newspaper (e.g. Fox News App [1]). Another category would be the RSS reading apps that also count as newsreaders because most news sources also provide RSS access to their content.

For Switzerland there are two noteworthy apps in the Applestore: [2] and [3]. Both choose a straightforward way of presenting articles, split into categories like Business, Sports etc. Upon selecting an article, the website of the publisher opens. One could speculate the two apps are actually RSS readers with hard coded RSS feeds.

2.3 News Recommender Systems

Many news providers generally implement an obvious or hidden recommendation system. Hidden simply means the system is not advertised to the user as a recommending system. Google News also has its alternative, called “Recommended for you”, which is further discussed in Section 5.2.

For mobile devices, the selection of recommending news readers is limited to a few solutions (e.g. [4]) and, to the best of my knowledge, none for the Swiss market.

Reasons for the lack of recommender systems could be the associated high computational costs and the high expectance from the customers on advertising a “Recommended for you” service.

Recommending news is currently a hot research topic; publications on recommender systems usually define news recommenders as a category itself. More about related work on news recommender systems can be found in Section 5.2

3 Solution Debate

3.1 Approach

When debating around an implementation of a mobile newsreader, the simple usage scenario presented in the introduction has to be recalled: a user enters the bus in the morning and would like to quickly get a personalized brief about today’s interesting news.

The most important feature of the proposed newsreader, and the one that is mostly missing from current applications (see Chapter 2) is a recommender system for news. The time each user has at his/her disposal is unknown, but rather very short, so the main headlines should pop into the display and be readable with as few actions as possible. To make this possible, a recommender system can be used to filter articles based on the personal profile of the current user and push out only articles that are relevant for him/her. By showing only relevant titles we make sure that no time is lost on browsing for information. The

solution is, intuitively, even better than searching; if he/she were to search, what would he/she search for? The recommender on the other hand brings proposals and the user only has to decide what to read first.

Of course users should have the possibility to browse for articles in case they want to read more than the recommended news. Most publishers use category based browsing (finance, sports etc.), method that has established itself as the standard in presenting news. Because users are already familiar with this approach and in addition it offers the possibility to discover topics that were not recommended, it is also available in the proposed application.

Regarding how the information should be presented to the user, an approach presented in Chapter 2 is using the phone's web browser to read news off a website (Google News approach). One of the advantages is that users do not need to install any software and updates are automatically pushed to all users. This solution however cannot use the persistent storage of the phone hence it cannot make content available offline. Also it cannot read the information of other sensors of the hardware like light conditions or movement detection. These are especially useful in extending the app for example through implementing advertising services or for providing location-based news or services.

Many of the already available solutions are self-supporting newsreaders that just download user specified or hard coded RSS feeds and display their content. A clear advantage of such a solution is the lack of dedicated servers. Depending on the amount of data, this can save a lot of money. However there are also major disadvantages: just imagine a publisher slightly changes the formatting style of articles in his/her RSS feed and the application is not able to read the feed unless updated. Obtaining user feedback and personalizing the content would also be impossible in such a scenario.

To overcome the mentioned issue, a server-client architecture is indispensable. With a much higher computing power, a server application can crawl for many news from different sources, aggregate, supplement, correct and personalize them and provide them to the user either upon request or recommended based on the user's profile.

In the best-case scenario, the presented solution to mobile news readers could change the way people get informed about what is new in the world. It would enable an instantaneous access to information that is proven, by the recommender system, to be more interesting. In the wildest dreams, it could even be an opponent to the number 1 time-killing activities on smartphones: updating Facebook status.

3.2 Implementation

3.2.1 News Acquisition

Most of current newsreader solutions only provide content from a single publisher. This way they get closer to the traditional model of providing articles: a physical newspaper contains only news from the respective publisher. A distinct approach could be mixing articles from most publishers and obtaining a more diversified news source. The advantages of such an approach are clear: a

publisher might not write about every event; hence mixing articles provides information about most events. However, when mixing content from different publishers, the application has to deal with the following non-trivial problem: what if two publishers write about the same event? Content would be duplicated.

As discussed in Section 2.1, Google News tracks duplicated news and groups articles together that appear to talk about the same event. Because of the complexity of determining if two articles deal about the same event, our server crawls Google News for articles.

The content provided by Google News is far from perfect for our purpose: images are rarely available and if so, only as a thumbnail and not in the original size; the text snippet from the news is often incoherent, mixing the publishing date or the author name between the sentences; the provided links often take the user to the desktop-browser version of the website, even if a mobile-browser version exists etc. A solution to all these problems is a system that, for each article, additionally crawls the website of the publisher directly; this is done through dedicated crawlers that are configured for each publisher. Only crawlers for the main Swiss publishers have been configured so far: NZZ, Blick, 20minuten, LeMatin, VingtMinutes, TIO etc.

There are two notable advantages of first restricting the application to the Swiss market; the first is the increased quality by using special crawlers that bring additional information about each news, like a full resolution image. The second advantage is the lower number of major Swiss publishers. Much more crawlers would have to be implemented to cover the important news agencies of Germany or France, for example.

Having four official languages, Google News has to be crawled for each language separately; news are however saved to the same database and multilingual users can opt in to read news in more than one language. Also, the proposed recommender system can make predictions in all the languages the user opted for.

The news threads are obtained from Google News by downloading data from RSS feeds. For each news thread, all articles are additionally crawled through standard website crawling. In a next step, the articles website is crawled for each supported publisher and the obtained information is used to update the (possibly mis-formatted) information obtained from Google.

The data on the server is accessible through web services that respond to request with XML formatted data. Enabling communication through XML allows other platforms to use the same services – an Android application for example. Only two services are needed to provide information for the current mobile application: get list of latest news thread IDs and get all fields of some news thread. Through the first of the two, the server decides what news to show on the phone, and in what order. The order of the recommended news is hence decided by the server and not by the client application.

Section 4.1 provides further details on the processes running on the server and the used technologies.

3.2.2 Presenting News

Referring again at the simple user scenario presented in the introduction, the application needs an intuitive and very minimalistic design. This however does not come at the expense of functionality – the application provides a configuration screen where users can opt in or out from the publishers that should appear and be recommended (e.g. if a user, for some reason, doesn't trust 20minuten, he/she can choose not to have 20minuten displayed to him). On the first launch, the application queries the language settings of the phone and automatically selects only those sources that match the language preference of the user. The user-interface is currently available in German, Italian, French and English and is changing automatically with the language preference of the phone.

In order to be very fast at startup, the application saves the downloaded news in the phones memory such that on the next startup it doesn't need to show a blank screen. This is especially useful if the phone cannot establish a network connection – the application can be used normally (read abstracts), the only difference is that no new articles can be opened in the browser.

Each time a user changes from one view tab to another, and also when he/she starts the app, a news fetch starts in background automatically, which downloads new recommendations for the user; each new article discretely slides into view as soon as it is downloaded. On selecting an article, the view modifies and shows a short abstract for the news and an image if available. The user can click on the image to open the article in an app-internal web browser and read the full story.

Tabs at the bottom of the screen enable the user to switch between the recommended news view, the categories view and the read news view. The categories view allows the user to browse for news belonging to a specific category (finance, sports etc.) in a similar manner as the recommended list. These lists also start preloaded with the articles that were fetched on the last update and new updates are automatically fetched in background.

For updating the currently displayed article list, the application launches a request to the server in which it states the type of the list it needs to update (recommended news or one of the categories). The server then decides what articles should be displayed and returns a list of news thread IDs. The application checks, which articles are already in the memory and only request those from the server that were not downloaded before. The high-resolution images of each article are also downloaded from the server only on demand.

The recommender system needs feedback from the user in order to improve the recommendations. For reasons detailed in chapter 5 - Recommender, the application only gathers implicit feedback from the user; in this case it is a log of his/her actions. Some of the currently logged actions are: application starts, applications ends, application downloads articles, user reads the abstract of an article, user opens the article in the browser etc. These actions are logged to the persistent memory of the application and are being sent to the server as a batch when closing the application.

3.2.3 Personalizing Content Provision

As presented in Section 3.1 the recommender system is the main feature that sets this application apart from its competitors. By making suggestions to the user, such a system is ideal for users that don't know what they could be interested in and want to discover something new.

For each request of recommended articles launched by a user, the server first computes a number that represents how many articles should be returned. This number will be chosen such that the sum of recommendations throughout one day (24 hours) is not more than 40 recommendations. In this list of 40 articles per day but not more than 10 at a time, three types of results are randomly mixed: *suggested articles*, *random articles* and *updated articles*.

Suggested articles are the actual result of a recommender system; in order to compute them, the system needs to know the users preference – which in this case is the list of articles that the user has read, meaning the news he/she showed interest in. Based on these articles, the user gets other articles recommended, which are similar to the ones he/she read. For computing the similarity between two articles, one could start by counting how many times both articles were read by the same user. If a lot of users showed interest in both articles, it means those two are very similar. A complete description of the recommender system is in chapter 5 - Recommender.

Random articles are placed in the list for two reasons: first, and most important, push articles to the user for which the system has no information. The recommender system cannot suggest an article if it no user has read that article; hence pushing out random articles gives new articles the possibility to start being recommended. The second reason for random articles is the possibility to measure the performance of the recommender system when compared to a naïve (random) grand truth.

Updated articles are articles that the user has not yet read, but they come as updates to articles that he/she read before. Because of limitations detailed in chapter 5 - Recommender, these results cannot be made part of the recommender algorithm, but are an important part of the recommendation: users that showed interest in an event, are likely to be curious about the evolution of that event.

Using a recommender system can help targeting information for the users so that they don't have to search for it anymore. In a further extension of the application, the same recommendation strategies could be used to target advertisements or location based events to the users.

3.3 Known Restrictions

Adopting a client-server architecture also brings a main disadvantage: high costs. As all the content provided to the users, including images, are sourced from the server, this could lead to high bandwidth requirements for a high number of users. The computing power requirements for generating recommendations would even rise polynomially, with increasing number of users.

The performance of the system, especially in terms of quality of recommendations, is highly dependent on the number of active users. As word of mouth is, sadly, the only free way of advertising for an iPhone application, it could take a while until a solid user base is obtained and good recommendations can be generated.

4 Architecture

4.1 Server Side Development

4.1.1 Introduction

When dealing with data crawled from the Internet, an application must be prepared to handle exceptions, especially when the content providers change their content. Another concern is the scalability of the server side, because the expected number of users can increase very fast when the application becomes popular. To assure stability in this volatile environment, the server side application is running several weakly coupled modules that are only connected to each other through the common database. The next sections separately describe each of the modules.

Using a modularized approach makes the whole system more fail-safe; if a crawler encounters an error, it crashes but all other crawlers are unaffected. An even more important advantage that comes from using modules is the increased scalability of the system. If, for example, the article-providing module becomes overloaded with requests, it can easily be duplicated on another machine and run in parallel with the original with small modifications. In the same manner, if an Internet data crawler is identified as such and blocked by a data source, it can easily be ported to another machine, another IP address, and be rerun without additional configuration on the server side. Each of the modules has a simple web interface that displays information about the current state, reads the errors of the console, if any, and allows for starting and stopping the activity.

The server side is actually a suite of four different applications (Figure 2): two crawlers (stage 1 and 2) that run as self contained java applications, another java application (stage1 recommender) that pre-computes results for the recommender system and a service (provision services and stage2 recommender) that is bound to an existing web application container – Tomcat 6 in our case.

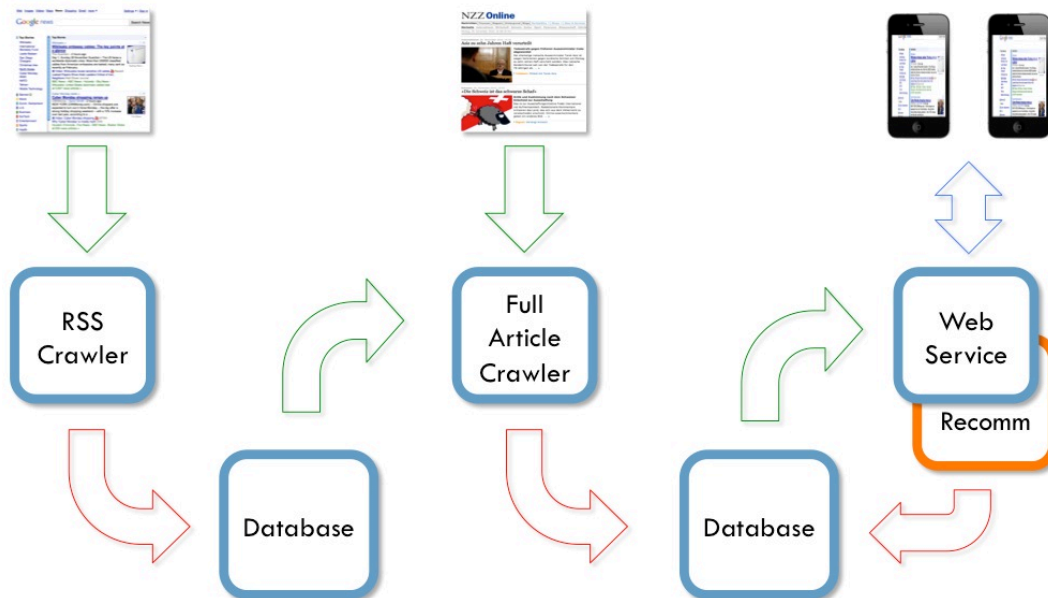


Figure 2 - The architecture of the server part

4.1.2 Stage1 Crawler

As discussed in Section 3.2.1 Google News has been chosen as main news provider for the system, mainly because it aggregates articles on the same subject published by different news providers.

Crawling articles from Google News is done in two steps. The first step runs an hourly crawler that fetches one RSS feed for each of Google's categories: International, National, Business, Technology, Entertainment, Sports and Medicine. The RSS feed always returns the latest 10 news threads of the respective category. In a second step, the crawler requests a list of all articles that are part of this news thread (meaning articles that write about the same event). If present in the database, the news thread is identified and articles that were not present at the last crawl are added to this news thread in the database. If the crawled news thread is not present in the database, it is added together with all its articles. The information stored for each article is title, short description, publisher, date and link. The link is also used as a unique key of the article while the date is the date of the crawl.

A web crawler is used to crawl all this information from both RSS feeds and web content. There are several open source web crawlers that can be used for this task [5]. Most of them are in a beta state, which makes them unsuitable for the production environment of this system. WebHarvester [6] is a Java library already at version 2.0 that can crawl information from any XML formatted content (so also directly from RSS feeds) and can be configurable through external XML files. This has the advantage that when a data provider changes the content, the java code must not be modified and recompiled to reflect the changes, but only the crawler's configuration files have to be updated. Data can be easily obtained from the HTML/XML code using XPath.

The RSS feeds managed by Google, however, offer information in a format that doesn't respect the RSS 2.0 standard [7]. What Google does is inserting all elements of an article inside the <description> tag of an RSS entry, encoded in a

HTML format (Figure 3). The information inside the RSS has to be parsed using the WebHarvester web crawler.

[Berlin und Paris einigen sich auf Nato-Kompromiss - WELT ONLINE](#)



[Hannover](#)
[Zeitung](#)

[Berlin und Paris einigen sich auf Nato-Kompromiss](#)

WELT ONLINE

Beide Länder machen Zugeständnisse beim Raketenschirm. Letztes Hindernis für das neue Strategie-Konzept ist die Zusammenarbeit mit der EU. Die Nato baut eine Raketenabwehr für Europa auf. Unmittelbar vor Beginn des Nato-Gipfels in Lissabon am Freitag ...

[Nato-Gipfel: Obama in Lissabon eingetroffen](#)FOCUS Online

[Neustart von NATO und Russland?](#)ORF.at

[NATO-Gipfel: Atomabrüstung und Afghanistan](#)Kurier

[STERN.DE -BILD -Spiegel Online](#)

[Alle 749 Artikel »](#)

Figure 3 – Screenshot of an entry in Google News' RSS feed

Another problem with Google's RSS feeds is that the <guid> field of each entry, which is supposed to be the unique identifier for an entry, changes over time. If for an event there is only one article, the <guid> takes the value of this article's link. If at a later time several publishers write about the same event, the <guid> changes into a Google internal unique number. This little change makes the updater think it deals with a completely new news thread and would insert it into the database as such. There is a routine on the server that currently checks for situations of this kind and fixes them.

The Stage1 module is an independent java library that can run on any computer that has a network connection, without any additional configuration. The crawled information is used to populate two database tables: news threads (called topics) and articles (called articles). Because the crawler compares fetched data with the one existing in the database, several instances of this crawler can be started on different machines as long as they don't execute their crawling cycle in the same time (they have to be synchronized manually).

4.1.3 Stage2 Crawler

In Section 3.2.1 a second crawler has been presented that crawls the information directly from the publisher's website. Independent on the Stage1 Crawler, the Stage2 Crawler goes through the articles present in the database and for each publisher of an article it checks if there is a configuration on how to crawl information from this publisher's website. If so, details of the article are fetched directly from the website of the publisher and used to replace the information in the database. After this step, the updated articles are marked as "good to publish" and are ready to be sent out to users.

The Stage2 Crawler makes sure the information in the database is more accurate than the one provided by Google News. Also, compared to Google's short introduction, the Stage2 Crawler populates articles with the full body of the article text, which is helpful for doing text analysis and content-based

recommendation. Additionally the link to the article, that normally points to a desktop-browser formatted website, is changed into it's equivalent for a mobile-browser formatted website, because most publishers provide that. Just like the Stage1 Crawler, this crawler is also a Java application that can run on any machine with an Internet connection. Moreover, several instances of Stage2 Cralwer can run simultaneously without interfering.

4.1.4 Stage1 Recommender

Just as Stage1 crawler's output is used as Stage2 crawler's input, the Stage1 recommender is a self standing java application that computes a pre-stage of results, used by the Stage2 recommender to generate suggestions for users. The routine is scheduled to run every hour and update tables in the database. The connections to the database are pooled through an Apache DBCP [8]. More about the tasks and necessity of the Stage1 recommender are discussed in chapter 5 - Recommender.

4.1.5 Provision Services

The Provisioning System provides several services for accessing the data from the client side application. Two of them were already introduced in Section 3.2.1: Topics and Topic.

4.1.5.1 Topics

'topics' is a service used by client applications to query a list of news threads that should be used to update a news threads list on the mobile device.

The parameters that come with the request are the device-ID, the type of list he/she wishes to update and the publishers it wants articles from. The list type can be of two kinds: recommendation list or category list. In the case of recommendation list, a list of recommended items is generated by the recommender and returned (see Section 3.2.3). In the case of the category list, the server executes an SQL query in the database that returns the newest news threads that contain articles from the desired publishers and belong to the specified category.

The returned result is in XML format and consists of a list of pairs of news_thread_id and updated_number. The updated_number can help the client application see if a news thread changed since the last time it was downloaded. In the case of recommended items, the order of the results is relevant for sorting the news threads on the client device.

4.1.5.2 Topic

'topic' is a service used by client applications to retrieve all details and articles of a news thread. The parameter of the request is the news_thread_id that is to be retrieved.

The response is in XML format and contains the following details about the news thread: ID, publishing date, category, thumbnail and articles. 'Thumbnail' is a Base64 encoded .jpg image. The 'Articles' tag lists all articles of this news thread and for each it contains the following details: ID, imageID, url, non mobile url, publisher, title, description and publication date. The difference between 'url'

and 'non mobile url' is that the former is for the article on a website formatted for mobile-browsers while the latter is formatted for desktop-browsers.

4.1.5.3 Sources

'sources' is a service that provides a list of publishers the server is currently able to crawl. These are presented to the user who can choose which to accept and which to be declined when retrieving the results.

4.1.5.4 ArticleImage

'articleImage' is a service that takes an imageID argument and transmits the specified article image as binary code. Retrieving images in a dedicated service is offered such that images are only downloaded on demand by the client application.

4.1.6 Stage2 Recommender

When the 'topics' provisioning service receives a request for recommended items, the recommender is asked for a list of news threads.

The recommender itself has a modular structure composed of recommending modules and the underlining structure. The structure is executing the following tasks: determining how many items have to be recommended; initializing the recommenders and asking for suggestions; mixing the suggestions randomly and obtain the recommendations that are sent out to the user; logging the recommendations made to the user and how long it took the algorithms to generate them. The models are the actual recommenders and are classes that extend the NuesliRecommender class. Hence it is really easy to insert a new recommender system into the framework: all that needs to be done is extending the NuesliRecommender class and assign it to the list of recommenders to be used by the recommender underlining structure when asking for recommendations. Currently the application uses three recommenders: Suggested, Random and Updated. Because the recommender system is a plugin for the server it is described in a special chapter of the document (5 - Recommender); for a description of the other two, please recall Section 3.2.3.

4.1.7 Database Structure

The current design of the server side includes one database that acts as the bridge or the communication channel between all modules of the application. However, under heavy load conditions, the database could be split in three independent parts: one for saving the data generated by the Stage1 Crawler; a second for saving the data generated by the Stage2 Crawler and used by the provisioning service and a third used by the recommender system and to store the logs. The provisioning system was described in Section 4.1.5.

The database is currently composed out of the following tables:

- Topics –the different news threads (or groups of articles around the same event);
- Articles – all articles
- Users – a mapping between a userID and the deviceID of the user
- Logs – the usage logs received from the applications

- `Recomm_data_model` – a mapping of users and the items they read. Used by the recommender system.
- `Recomm_item_simil_loglikelihood` – the similarity between two news threads
- `Recomm_logs` – the recommendations sent out to the users
- `Recomm_time_logs` – the time each recommender algorithm took to compute recommendations.

4.2 Client Side Development

4.2.1 Introduction

The client side application is an iPhone application written in Objective-C using Xcode3 – an IDE currently running only on Mac computers. The application is also using iPhone specific libraries like CoreData (persistent storage handling library), CoreGraphics (library for enabling graphical effects) and others.

One major goal when designing the iPhone application was respecting a Model View Controller MVC architecture; this comes in handy when designing an iPad version, that basically has the same functionality but with a slightly different user interface. As a result of the MVC architecture, the application is centered on the Logic class, which acts as both the resources coordinator and communication channel between the View and the Model. Broadly described, the View adopts tasks as displaying and updating the data to be displayed, interacting with the user, logging the activity of the user, and handling the notifications display system. The Model, on the other side, assumes tasks as downloading new articles from the server, updating the persistent storage and notifying the view of such updates.

4.2.2 Implementation Details

For displaying data in list form, objective-c uses a programming paradigm where a single object (actually group of objects) is responsible for reading data from the persistent storage and displaying it; using this paradigm allows the programmer to only modify the data in the persistent storage, while the display is updated automatically. However, when programming an application that strictly follows the MVC architecture, it's difficult to implement the provided paradigm because the display belongs in the View part while the persistent data handling belongs in the Model part. Nevertheless, because the objective-c paradigm has a lot of advantages if implemented, a trick is used to overcome limitations: the group of classes (designed to be implemented by the same object) was split in two parts: one that handles reading the data from the persistent storage and one that handles displaying the data. Because both components would have to notify each other of different events, communication interfaces have been set up that enable a communication through the Controller, in our case the Logic class. Hence the paradigm suggested by apple is used while not harming the MVC architecture.

Starting from this trick, another idea emerged that makes the architecture even more simplistic: the part that handles reading data from the persistent storage, also handles downloading updates from the server. Hence, this strongly coupled group of two classes (one displaying the data and the other one provisioning the data from both the server and the persistent storage) becomes a self-contained

component that can be used in different parts of the application. Let's call this component a `ProvisioningNewsList`.

Section 3.2.2 presents the application as having three displaying tabs: the first presenting the recommended news, the second presenting a list of news belonging to a category and the third presenting the read news. The similarity between all these is that each one presents *a list of news*. From the software architecture point of view, these lists of news can be different instances of the `ProvisioningNewsList` presented earlier. Hence if the current list is supposed to present the recommended news, a `ProvisioningNewsList` object is instantiated with the parameters that define it as a recommended news list. Based on these parameters, the `ProvisioningNewsList` will already know what news to request from the server when updating and what logs to write when logging.

In respect to the `ProvisioningNewsList`, the application can now be presented as a group of `ProvisioningNewsList` objects, each hosted by one of the tabs. Each `ProvisioningNewsList` then delegates tasks to other components of the model (for example the XML parsers) to populate itself with information from the persistent storage and from the server.

4.2.3 User Interface

Section 3.2.2 presents how news slide into view the instance they are downloaded. This is not a trivial task and this is because of how the iPhone OS is handling user interface requests. If a thread that downloads articles is running in the background, it triggers a view update after each article has finished downloading. The view, however, ignores these calls until the thread has completely finished execution. It is only then the user interface updates itself with all new articles. This context might be useful for most applications, but in the case of this application, the user's experience enhanced if he/she really sees the new articles popping in place when the progress bar suggests an article has been downloaded.²

4.2.3.1 Recommender View

The tab showing recommended items launches requests to the server to receive updates. Unlike the other news lists, the news threads shown are not sorted by date but by the order in which they were received from the server. Using this technique makes sure that when news are downloaded they are not used to replace the ones already present on the display, but they are added to the top of the list. On the other hand the server has the possibility to freely reorder the news already displayed, if at some point this is desired.

The actual steps when downloading news for the recommended list are: find out whether the news is already in the list of recommended items; if so, skip downloading and move that news to the top, marking it as unread. Of course the process is a little more complex: a news thread is not downloaded if it is already

² A trick is to force the update to execute on the main thread, using the method: `[NSObject performSelectorOnMainThread:aSelector withObject:arg waitUntilDone:wait];`

in the persistent storage (for example downloaded by a category news list view). It is however downloaded if the local copy is discovered to be outdated.

4.2.3.2 Categories View

The categories tab first presents a list of all categories in which news were classified by Google News. The top category is a mixture of all categories and is basically a way to browse news sorted by date. Each of these categories opens a `ProvisioningNewsList` that shows (and downloads in background) the latest news of that category.

4.2.3.3 Read News

The read news tab shows news threads that the user has read and that were not yet deleted from the application. On every exit, the application deletes all news threads that are old and only the latest 10 news threads are kept for each category; even if this sounds little, it amounts to 80 news threads that are in the application at any time.

4.2.4 Personalization

The application allows the user to select which publishers he/she would like to receive content from and which should be ignored. When the user changes these settings, the application does not transmit the values to the server but rather saves them and transmits them with every request for new articles. This method makes use of less connections and easier processing on both server and client sides.

4.2.5 Retrieving Content

Content like list of news threads and the fields of a news thread and its articles, are all received in an XML format, which is parsed and saved to the persistent storage. Upon saving, the application automatically notifies the lists that are currently displaying events that there is new content available; the view is hence updated automatically. This way, the background thread that downloads new articles must only save them to the persistent storage – the update of the view is handled automatically.

The process of retrieving content is optimized in respect to used bandwidth. News that are already in the persistent storage are not downloaded again and also pictures are not downloaded as long as they are not needed. When a user opens a news thread, the image of the first and second article are downloaded. If the user scrolls through the articles, only the image of the currently displayed and the next article are downloaded, to save data bandwidth on the mobile device. Images are retrieved from a dedicated service that returns data in binary form.

5 Recommender

5.1 Introduction to Recommender Systems

A recommender system is an algorithm that is able to recommend items to a user based on his preference history. In the general case, this is done by finding other users that have expressed similar preferences as the current user, and

recommending the current user items that have been preferred by the other users.

The idea of recommender systems has been in the computer science community since the '90s, the news recommender systems being an important part of the research right from the start; in 1992 Lorrie identifies three major advantages of e-newspapers: timely, richness and customization [9]. Coming back to 2010, research currently classifies recommender systems in three main categories: Cognitive, Collaborative and hybrids of the two.

Cognitive recommending systems usually work by extracting features of the items to be recommended and compare the features of items. If items have similar features, the similarity of features is assumed to approximate the real similarity between the two items. Hence a recommender system could look at the items a user has shown interest in and recommend other items that are similar to those. Decision tree learners, such as ID3 [10] are used to partition the data (news text in our case) into subgroups until those contain only instances of a single feature class. [11]

Collaborative recommending systems, on the other hand, look at the user interaction in deciding upon similarity between users. In order to recommend items to a user, collaborative systems are first computing a similarity between each two users; this value is easy to compute as the ratio between how many items both users preferred to how many items were preferred by both in total. Having computed a numeric similarity between users, a basic recommender could suggest items that were preferred by the most similar user and not yet seen.

Table 1 - List of existing recommender systems with a short description of each (source: [12])

Technique	Background	Input	Process
Collaborative	Ratings from U of items in I.	Ratings from u of items in I.	Identify users in U similar to u, and extrapolate from their ratings of i.
Content-based	Features of items in I	u's ratings of items in I	Generate a classifier that fits u's rating behavior and use it on i.

Both collaborative and cognitive filters have major advantages and disadvantages (detailed in the following paragraphs), making it difficult to recommend one over the other in a general sense. One of the most important advantages of cognitive (also called content based) recommenders over the collaborative counterparts is the ability to overcome the cold-start problem; because the quality of a collaborative filter is related to the amount of ratings the system knows, such a filter cannot say anything about a new user or a new item. The content-based approach can however deal with any new item because the way it's analyzed does not depend on how many ratings it has already received,

but on the system's ability to identify features of items. Another advantage of content-based filters is their computational performance: while a collaborative filter must keep track of new ratings and update itself, a content-based filter only has to analyze the data on its first appearance and, in a properly trained system, this remains constant in time.

Collaborative filters have grown in importance since their use for enabling web based services. Having many active users in a community makes gathering of ratings especially easy and collaborative systems perform very well providing a sufficiently high number of active users. Another advantage over the content-based alternatives, is novelty, the ability to recommend items of different fields of interest [13]; while content-based systems are limited in finding items whose content is similar to the user's item history, collaborative systems can go beyond that and let the user explore other fields of information. Collaborative filters are also easier to understand and implement than content based filters and can be applied to any content type (images, songs) for which obtaining features might prove difficult.

Collaborative solutions also have downsides, like the skewed correlations [13]; two users, both having accumulated only few ratings, can have the same items rated, making the system assume they have perfect similarity. This case also relates to the situation in which some universally popular items are rated by most of the users; an algorithm should thus weight the influence of a common item with the inverse of its overall popularity, in order to compute more accurate similarities between users.

Hybrid systems combine two or several recommender that work together to improve the quality of recommendations. In the case of the previously discussed cold-start problem, content based filtering could be applied to users that haven't accumulated enough ratings for doing collaborative filtering [14].

Table 2 - Short pros/cons view over collaborative filtering (source: [12])

Technique	Pros	Cons
Collaborative filtering (CF)	<ul style="list-style-type: none"> A. Can identify cross-genre niches. B. Domain knowledge not needed. C. Adaptive: quality improves over time. D. Implicit feedback sufficient 	<ul style="list-style-type: none"> I. New user ramp-up problem J. New item ramp-up problem K. 'Gray sheep' problem L. Quality dependent on large historical data set. M. Stability vs. plasticity problem

5.2 Related Work

Recommender systems received much attention in the period 2007-2010 because of Netflix³, an online movie rental company that has started a contest on improving its movie recommending system by 10%. Because of the competition,

³ <http://www.netflix.com/>

the topic of recommender systems, in general, moved into the spotlight and several service providers worldwide have started to implement a personalized recommendation system. The winning solution to the Netflix contest [15] can be considered a listing of the state of the art recommender algorithms available today. An important finding of the paper is that implicit feedback is much more important than initially believed: the fact that people *do* rate a movie, already says something about their preferences. Also the strategy of the winning solution is to implement as many recommenders as possible and weight them automatically through Adaboost [16], rather than tuning and improving each individual recommender.

Several other web based services implement recommender systems to enhance the user experience; Jinni⁴ is such an example that again implements a movie recommender system to suggest movies the user might be interested in. The online community actually considers Jinni to perform better than the price-winning algorithm from Netflix. Just like Jinni, some other web companies don't advertise their systems as Recommender Systems, probably because of the negative impact this could have on users' expectancy.

Having a strong impact on the way people get informed, news recommender systems have been implemented using different approaches but with mainly unpublished results. Several papers in the field talk about customized recommenders and present the results of the field studies but no industrial application has published its datasets on news articles recommendation.

A notable publication in the field is the report on the recommender system implemented by Google News as a side functionality of their news-aggregating platform [17]. The authors present a strictly collaborative approach to news filtering: two recommender systems that are combined in the end by mixing their results into a common list. The first recommender uses a user similarity based collaborative filtering that clusters the users into groups; to achieve this, two clustering algorithms are presented: a pseudo Jaccard Coefficient algorithm and a PLSI algorithm. The second recommender basically computes a time discounted similarity value for each pair of articles. According to the authors, the explanation for strictly using collaborative filtering is the goal to build a system capable of recommending not only news but also images or other type of content. Also the recommender should be able to work without training on content in any language.

News@hand is a news recommending web application that claims to solve several problems of current recommender systems [18]. Because of the very brief discussion of their algorithm, one can speculate the system uses content based filtering to enable recommendations. The empirical results of the News@hand system were not shared.

⁴ <http://www.jinni.com/>

5.3 Premises

5.3.1 Data Type Analysis

An extended description of the items to be recommended has to be done in order to decide upon a good implementation of a recommender. The current project focuses on recommending news articles to users. An article is usually composed of a title, a text body and possibly an image; hence content-based filtering is a possible recommending alternative (as chosen by News@hand, presented in Section 5.2).

The server currently downloads approx. 120 new news threads per workday and about 40 updates to news threads per day. As the system is only recommending news of the last few days, we can approximate a number of 500 items that (gradually) change every 3 days. In terms of user numbers, the system will target around 10.000 users in Switzerland. It is assumed each user reads around 5 news per day; this generates 150.000 ratings in 3 days – an expected number of 300 ratings per item at the end of the 3 days. This is equivalent to an average number of 150 ratings per item in the item's lifespan. Please note that these figures are only valid in the case of people randomly reading news and do not reflect the real situation in which people have preferences for news.

The recommender algorithm will be running in a continuous cold start as all news being gradually exchanged over a period of 3 days. This will prove to be a big problem of the recommender, a partial solution being described in Section 5.4.

5.3.2 Available Ratings Data

The way of obtaining rating information was briefly presented in Section 3.2.2 as implicit rating based on the activity log of the user. Using implicit ratings generally has a major disadvantage in recommender systems: if the interpretation is wrong, the recommender system is fed with wrong information. In an explicit rating system, the gathered feedback represents the opinion of the user more accurately. Particularly in the case of news recommender system, relying on the user to offer explicit feedback would reflect the position of the user to the article's content, rather than the quality of the recommendation itself. While such a behavior is desired when rating movies, it would not work for the news domain because users still want to get informed, even if the news are bad.

The activity that gets logged is: whether the user opened the description to a news; whether the user opened the news in the internal web-browser of the application; whether the user continued and opened the news in the phone's web browser (for example to copy the link of the article); whether the user has advertised this news to friends through Facebook, Twitter or mail; whether the user has reopened the article from the read news tab. Another approach on the logs can give information about the time spent by the user on reading the abstract and on reading the news.

When computing the interest of a user for an item, two types of 'rating' can be obtained: binary or gradual. Binary ratings are: he/she opens the description of an article; he/she opens the article in a browser. For computing gradual ratings, the system could use the following rules: the user opens the description

rating+=1, the user opens the news in the browser rating+=2; the user recommends the article to a particular friend per mail rating+=1, the user posts the article on Facebook or Twitter, rating+=2; user navigates to the read articles folder and re-opens an article rating+=2. For the first recommender, only the binary rating of opening the description is used.

5.4 Proposed Solution

5.4.1 The Item Based Recommender

Several papers ([13], [11]) of the literature suggest to use content-based recommenders in connection with news articles; Schafer [13] presents a list of features a domain has to have in order to be suitable for collaborative filtering.

1. There are many items
2. There are many ratings per item
3. There are more user ratings than items to be recommended
4. Users rate multiple items
5. For each user of the community there are other users with common needs or tastes
6. Item evaluation requires personal taste
7. Items are homogenous
8. Items persist
9. Taste persists

While applying the requirements for the news domain, it's obvious that point 8 will not hold, while points 2, 3 are only fulfilled in the ideal case of having thousands of users. Despite these requirements, some solutions like Google's recommending system [17] rely solely on collaborative filtering.

As presented in Section 5.1, collaborative recommending systems generally work by counting the number of items two users have in common and thereby computing a similarity value between the two users. How about inverting the terms user and item in this concept and analyzing the outcome? Two items have a similarity proportional to the number of common users over the total number of users [19]. The outcome is not the same, especially if the news turnover is drawn into the equation. When looking at users, we consider two users as similar if their whole history of reading articles is similar – this could however stretch over years. The more time users spend using the application (maybe years) the more unlikely it would be to find similarities with other users; when taken to infinity, this results in a constant similarity between any two users. However, when computing the similarities among articles, these similarities get erased as soon as the articles are too old to be recommended anymore. Not only does this help the system to remain dynamic to changes in user behavior, but it also helps in the case of the cold start user problem: if a new user is joining the community, it is enough for him/her to show interest in one article, and a few can already be suggested based on the similarity with that item. Vucetic argues that this strategy, mainly promoted by Amazon [20] with the title “users who bought X also bought Y”, is much faster (about two orders of magnitude faster) and has slightly increased accuracy compared to user similarity based recommender systems [19]. Karypis empirically tests the performance of item

based similarity on five different datasets and generates, on average, 18.8% more accurate recommendations than user-based equivalents [21].

It is noteworthy to mention that several publications, including [12] and [22], provide strong evidence that collaborative filtering generates better recommendations than cognitive filtering, on most datasets. Considering the additional advantages of collaborative filtering compared to content filtering (presented in Section 5.1) and also the results of item based algorithms, the approach of this system follows Google and Amazon's example of focusing on a collaborative recommending system and in particular on item based recommenders.

In order to have a fast response to recommending items to users, the recommendation system consists of two components: a routine, that runs every couple of minutes, updating the similarities between news articles, and a recommender that returns suggestions to specific users upon request.

The first stage of the recommender, the routine, reads all logs of users and updates a table in the database containing preference values of user X for item Y. In its first stage, the proposed algorithm only uses binary ratings, hence the presence or absence of a similarity is enough to compute the similarity between two articles as the number of users reading both items, weighted with the total number of users that read each item. If the similarities were numerical, as a second release of the algorithm proposes, a further weighting could be done considering these values. In computing the similarities of two items, several algorithms have been analyzed and weighted; the Tanimoto Coefficient [23], which comes close to what Google is using in its recommendation, proves to be less accurate than the LogLikelihood algorithm [24] that also does an additional weighting over all ratings in the space. The only downside of the LogLikelihood algorithm is its higher computational demand, but if this proves to be a bottleneck on the performance, the system can revert to using Tanimoto Coefficient without extra configuration.

Abhinandan suggests in his paper [17] to discount the value of a similarity with its age; by this, user reading both news three days ago is not as important as one reading both news now. The importance of such a measurement should be analyzed empirically by a future release of the recommender.

Using the presented algorithms, the recommender's first stage computes a table of similarities between any combinations of two articles, and saves this table to the database. In the first experiments, this routine takes under a minute to execute, but it is highly dependable on the amount of users and articles present in the system.

In a second stage, the runtime recommender, users ask for suggestions that have to be computed dynamically. The item similarities saved by the routine in a database table are loaded in memory and used to compute recommendations for the specified user. To compute these, first an article candidate set has to be determined; this is done by looking for articles that were read by users having at least one common article with our user. After having obtained a set of candidates, the algorithm computes the similarity of each of the candidates with

all items of the user's recent history and sorts the results by relevance (computed from averaging over the similarity values). Again Abhinandan suggests discounting the similarity values of the articles with their age, in order to push new content to the front. The results of such an approach will be analyzed empirically in an updated version of the recommender.

Both stages of the recommender were implemented using classes of the Apache Mahout⁵ project. The Mahout project combines several implementations of established recommender algorithms in a way that makes pre and post configuration easy to implement.

The system designed so far is using all the advantages of a collaborative filtering like novelty, coverage and the ability to recommend any types of items; even the user cold start problem was addressed by using item similarity instead of user similarity in determining recommended items. One major downside still exists, however, and this is the item cold start problem: the system has no information about items newly inserted in the system and hence cannot recommend them. Strategies of discounting the relevance by the age of an article could help with this limitation, as would randomly inserting new, never rated, items into the result. Shafer describes this problem as being especially troublesome in news recommending systems and suggests adopting a content-based recommender just for this case [13]. In order to do that, the performance of a content-based recommender has to be compared to the collaboration-based recommender, side by side, in a field experiment.

5.4.2 The Updates Recommender

A special particularity of the presented news recommender system is the tracking and detection of updates to news threads. Whilst a new article could or could not be of interest to a particular user, updates to the articles he/she reads are very likely to attract his/her interest.

The rules of the recommender are simple: if a person reads an article, it is not re-suggested to that person. Manipulating the logs, for example by marking the article again as not read would not work, because a special copy of the logs would need to be created for each user and article. One possibility is to look at updates as new articles, and not help the recommender with his/her decision. Another one is to ignore updates and regard the whole news thread as read, but this is surely not in the interest of the user. The chosen solution is to implement a second recommender, which recommends articles that were read by the user and have updates. The results of the Updates Recommender are then mixed together with the results of the item based recommender to generate one recommended articles set. Burke describes mixing as being a good way to bring results of a different recommender in the suggestions [12].

5.4.3 The Random Recommender

As discussed in Section 5.4.1 the biggest problem of the item based recommender is the cold start. Bringing random articles in the results first

⁵ <http://mahout.apache.org/>

allows unrated items to receive attention and also offers a way to measure the performance of the other recommenders that are use to populate the result set.

5.5 Solution Evaluation

5.5.1 Introduction

Section 5.4.1 introduces the first release of an item based recommender. Because the system has not been introduced to the market there is no dataset to test the performance of the recommender on; hence a different dataset has been chosen, from digg.com⁶.

Digg.com is an online portal that offers users the possibility to “digg” information online (videos, articles, blog posts) in order to advertise it to other users. If a digg.com user finds, for example, an interesting article somewhere on the web, he/she can copy the link into his/her profile (he/she ‘digg’ the link). His followers, other users of digg.com who chose to follow the activity of this user, will see this ‘digg’ and can read the article. If they like it, they as well can ‘digg’ it. Section 5.5.2.1 contains more information about the digg.com dataset.

The dataset from digg.com (from now on referred to as ‘digg_data’) is similar to the target of the presented recommender system (presented in section 5.3.1; from now on reffered to as ‘target_model’) in respect to the nature of items: large number of fast changing sources of information. Also the ratings are binary in both cases, whether a user has read an article or not. The information contained in digg_data is identical to the one that the proposed system crawls from its users to compute the recommendations. Although similar in essence, digg_data’s figures are far from target_model, as described in the next Section.

5.5.2 Experimental Setup

5.5.2.1 Data Set

Digg_data is data crawled from the digg.com website, through their API, from 24.11.2010 10:07 to 26.11.2010 21:29 and is a listing of all real-time diggs of the registered users of digg.com, in form of userID-itemID pairs with timestamps; such a pair shows that a certain user has digged a certain item at a given timestamp. A user cannot digg an item twice. In terms of target_model, a digg in digg_data is equivalent to a rating in target_model.

Table 3 - Comparison between figures of digg_data and target_model.

	Digg_data	Target_model
Diggs/ratings	562,207	150,000
Distinct users	39,694	10,000
Distinct items	453,558	500

Table 3 lists the figures of digg_data compared to target_model; not only does the digg_data contain much more information, but the ratio of users/items is inversed.

⁶ <http://www.digg.com/>

Table 4 –Distribution of items according to number of diggs (from digg_data). Column ‘No. of Items’ shows how many items have received ‘No. of Diggs’ diggs.

No. of Diggs	No. of Items	% total items
>0	453,558	100%
1	443,000	97.6%
2	5,164	1.14%
3	1,391	0.3%
4	720	0.16%
...	...	-
11	99	-
...	...	-
43-70	~10	-
70-140	~5	-
200+	1	-

Table 4 column “Initial Data” shows how the number of diggs is distributed: 97.6% of the 453,558 items only have 1 digg, 1.14% have two diggs and so on. Table 5 shows the same analysis from the users point of view. Note that the total number of diggs in digg_data is 562,207.

Table 5 – Distribution of users according to number of diggs (in digg_data). Column ‘No. of Users’ shows how many users have digged ‘No. of Diggs’ items.

No. of Diggs	No. of Users	% total users
>0	39,694	100%
1	14,900	37%
2	5,300	13%
3	2,300	7%
4	1,700	4%
...
10	430	1%
...
20	111	0.2%
...
43	1,161	2.9%
44	1,809	4.5%
45	1,679	4.2%
...
86	>10	-
200+	1	-

Because of computing time considerations, only randomly chosen 140,000 samples of digg_data are used in the next sections. Table x shows the important features of the sample of digg_data.

Table 6 – Comparison between figures of the initial digg_data, sampled digg_data, and the target_model.

	Initial digg_data	Sampled digg_data	Target_model
Diggs/ratings	562,207	140,000	150,000
Distinct users	39,694	23,019	10,000
Distinct items	453,558	116,837	500

5.5.2.2 Testing Procedure

In order to test the recommender, a feature of Mahout (introduced in Section 5.4.1) is used that generates performance measurements characteristic to Information Retrieval Systems [25].

The approach used to test a recommender is the following: for each user, the first n most popular items are removed from his rating history. Then the recommender is requested to recommend n items for the user. This approach is called the “. at n” performance. Different measurements are applied to the result: Precision, Recall, Fall-out and F-measure [25]:

- *Precision*: precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

- *Recall*: recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

- *Fall-out*: the proportion of non-relevant documents that are retrieved, out of all non-relevant documents available

$$\text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}$$

- *F-measure*: the weighted harmonic mean of precision.

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}$$

In the case of binary ratings, n ratings are randomly removed from the set, instead of the n most popular ones.

The result of the recommender is compared to other two recommenders: a random recommender and a popularity recommender. The random recommender just returns a random item on each recommendation request. The popularity recommender returns on each request items in descending order of their diggs count.

5.5.3 Results

For the Random column of the next tables, the value was computed using the formula:

$$\sum_{i=1}^n i \cdot p^i \cdot (1-p)^{(n-i)} \binom{n}{i}$$

where $\binom{n}{k}$ is the binomial coefficient and $p = 1/116837$.

5.5.3.1 Results at Rank 1

Table 7 - Information retrieval performance measurements of the recommender at rank 1 (on sampled digg_data).

Results in e-4	Item-based	Popularity	Random
Precision	57	1.85	0.08
Recall	8.31	1.85	0.08
Fall-out	0.0011	0.0085	
F-number	14	1.85	

5.5.3.2 Results at Rank 5

Table 8 - Information retrieval performance measurements of the recommender at rank 5 (on sampled digg_data).

Results in e-4	Item-based	Popularity	Random
Precision	34	1.12	0.4
Recall	2.21	1.12	0.4
Fall-out	0.0048	0.042	
F-number	4.15	1.12	

5.5.3.3 Discussion

The results show that the performance of the recommender is not as high as expected. However, compared to the performances of the popularity and random recommenders, the item-based collaborative recommender has a decent performance. Table x shows the performance increase in percent when using the item-based recommender.

Table 9 - The Performance at ranks 1 and 5 of the popularity and random based recommenders compared to that of the item-based recommender.

	Item-based	Popularity	Random
Precision at 1	100%	2.9%	0.1%
Recall at 1	100%	25%	1%
Precision at 5	100%	2.9%	1.2%
Recall at 5	100%	50%	20%

When looking at the item-based vs. random recommenders, the relative results show that the item-based recommender is much more confident about its first recommendation compared to the following ones; the random recommender increases its 'performance at 5' 200 times over its the 'performance at 1'.

If the data structure is taken into account, the recommenders were tested on a dataset that has 233 times more items than expected, and double as much users (see Table 6). It's obvious that if this system reaches its target of 150,000 ratings

per day, the results of the item-based recommender will be much better even in its current basic form.

The results on the `digg_data` show that the item-based recommender is generating better results even than a popularity recommender and is a good candidate for the first release of the system. After the system gathers a more extensive dataset, improvements to the item-based algorithm will be tested and implemented.

5.6 Known Limitations

Section 5.4.1 has already detailed the greatest limitation of the current recommender: article cold start.

Using as feedback only a limited set of the logs, this first algorithm intentionally ignores other recorded user actions, thus making the recommendations less reliable than they could be. In [13], Schafer debates on the uncertainty in ratings, stating how it can be diminished: collecting multiple observations of variables that monitor the same action, and combining them into a single estimated rating, is the best way to cancel out errors induced by the uncertainty of implicit ratings.

The LogLikelihood algorithm choice was already introduced as a computationally intensive method and is under monitoring; in future, it could be replaced with the less expensive Pearson Coefficient algorithm.

5.7 Future Development

The literature review of recommender systems presents several approaches to recommender systems, each tuned for a niche purpose and usually accompanied by empirical results. But, what this really shows, is that not much is known about this field; there are neither exact recipes for a good system nor clear guidelines how a system should perform. Hence, most current solutions are custom and usually optimized on their respective dataset.

In respect to the current state of the art approach to recommender systems, the current algorithm also implements a generic recommender algorithm that enables a decent performance until a more extensive dataset is obtained. Section 5.4.1 has already presented some of the enhancements that would be implemented in the following release: cognitive filter against the cold start problem; using the logs to get better ratings; discounting articles and preferences with age. Before implementing any of these features, however, their possible effect has to be measured empirically on a valid data set. Hence, the next steps are to make a direct comparison between the proposed collaborative recommender and the cognitive recommender discussed at the end of section 5.4.1 and experiment on ways of mixing the results of both recommenders in a constructive manner.

Next, experimenting with several time discounting values will bring insight on how effective such a method is and how it could best be implemented. [26] propose a method of letting older ratings have less influence on the results; Burke, however, argues that “they do so at the risk of losing information about interests that are long-term” [12]. It still has to be explored if such a measure

would rather help or harm the current system, which anyway ignores articles and ratings older than a certain time.

Another approach that could enhance the performance of the system is presented in [27] and handles the case of determining user similarities; Billsus suggests the creation of two distinct users profiles for each user: a long and a short-term profile, and motivates this decision “a user's information need changes as a direct result of interaction with information”. His empirical studies provide evidence for the utility of this new approach.

6 Conclusions

The problem addressed by this paper is the worldwide shortage of news recommender systems for mobile devices, and the lack of such for the Swiss market. A system architecture has been introduced, composed of a server part that aggregates news stories of different newspapers and a client part that presents the news on the user's iPhone. Using Google News as an underlining news aggregator, the server fetches news from different sources and makes it available to any platform through web services.

The news recommendation system introduced in Chapter 5, runs on the server and, for the start, implements a generic recommender system based on item similarity for generating suggestions. Various improvement alternatives are presented through the literature review, together with empirical results of the proposed approach.

The client side application is a newsreader implementing a minimalistic, easy to use, design; it presents a self-updating list of recommended news but also offers the user the opportunity to browse for recent news by category.

Recalling the user scenario presented in the introduction, this system enables any user to keep informed about the latest news without having to search for articles that are interesting to him. The intuitive user interface together with features like recommended news, posting to Facebook, choosing the preferred newspaper when reading about a topic – all these create a fully featured yet very easy to use newsreader for iPhone users.

7 Future Work

Throughout the work, suggestions have been made on how the current system can be improved or expanded. As discussed in Section 5.7, the next update of the system has to address the recommender system. Sections 5.4.1, 5.5.3.3 and 5.7, all contain references to papers that address various features of recommender systems and how they can be improved. The immediate task will first let the system run and generate a valid dataset, on which to then test the relevance of these improvements.

As future development, the newsreader application should be ported to other devices, starting with Android based mobile devices, in order to increase the

number of active users. All versions will be able to use the generic interfaces of the same server thus helping out in obtaining user preference data.

On the server side, based on the success of the Swiss version, the application can be modified to work also in other countries. For this, only the Stage2 Crawlers presented in Section 4.1.3 have to be recreated for the popular news providers of the respective country.

A future development should move the application away from Google News and perform its own aggregation of articles into news threads. This will improve many of the current limitations imposed by using Google data, as for example the recognition and promotion of updates to articles.

From the theoretical point of view, the current system offers several ways to obtain relevant academic information; for example it is perfect for conducting field experiments regarding different recommender system alternatives. Different uses have to be found and implemented.

8 Bibliography

- [1] iTunes Preview. [Online]. <http://itunes.apple.com/us/app/fox-news/id367623543>
- [2] iTunes Preview. [Online]. <http://itunes.apple.com/ru/app/swiss-news-switzerland-live/id365545320?mt=8#>
- [3] iTunes Preview. [Online]. <http://itunes.apple.com/tr/app/switzerland-news/id395121468?mt=8#>
- [4] iTunes Preview. [Online]. <http://itunes.apple.com/us/app/smart-news/id382788079?mt=8>
- [5] java-source.net. [Online]. <http://java-source.net/open-source/crawlers>
- [6] sourceforge.net. [Online]. <http://web-harvest.sourceforge.net/index.php>
- [8] commons.apache.org. [Online]. <http://commons.apache.org/dbcp/>
- [7] Harvard Law. [Online]. <http://cyber.law.harvard.edu/rss/rss.html>
- [9] A. Lorrie, *The Electronic Newspaper of the Future: Rationale, Design, and Implications.*, 1992.
- [10] J. Quinlan, *Induction of Decision Trees. Machine Learning 1.*, 1986.
- [11] Daniel Billsus Michael J. Pazzani, "Content-based Recommendation Systems," *The Adaptive Web*, pp. 325-341, 2007.
- [12] Robin Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction*, pp. 331-370, 2002.

- [13] Dan Frankowski, Jon Herlocker, Shilad Sen J. Ben Schafer, "Collaborative Filtering Recommending Systems," *The Adaptive Web*, pp. 291-324, 2007.
- [14] Robin Burke, "Hybrid Web Recommender Systems," *The Adaptive Web*, pp. 377-408, 2007.
- [15] Michael Jahrer, Robert M. Bell Andreas Töscher, "The BigChaos Solution to the Netflix Grand Prize," 2009.
- [16] Wikipedia. wikipedia.org. [Online]. <http://en.wikipedia.org/wiki/Adaboost>
- [18] Alejandro Bellogín, Pablo Castells Iván Cantador, "News@hand: A Semantic Web Approach to Recommending News," Madrid, 2008.
- [17] Mayur Datar, Ashutosh Garg, Shyam Rajaram Abhinandan Das, "Google News Personalization: Scalable Online Collaborative Filtering," Alberta, 2007.
- [19] Zoran Obradovic Slobodan Vucetic, "A Regression-Based Approach for Scaling-Up Personalized Recommender Systems in E-Commerce," 2000.
- [20] Brent Smith, Jeremy York Greg Linden, "Amazon.com Recommendations," Amazon.com, Industry Report 2003.
- [21] George Karypis, "Evaluation of Item-Based Top-N Recommendation Algorithms," 2001.
- [22] G. Karypis, J.A. Konstan, J. Riedl B. Sarwar, "Item-based Collaborative Filtering Recommendation Algorithms," *Proceedings of the 10th international conference on World Wide Web*, pp. 285-295, 2001.
- [23] Wikipedia. wikipedia.com. [Online]. http://en.wikipedia.org/wiki/Cosine_similarity
- [24] Ted Dunning, "Accurate Methods for the Statistics of Surprise and Coincidence," 1993.
- [25] Wikipedia. Wikipedia. [Online]. http://en.wikipedia.org/wiki/Information_retrieval
- [26] A. Kobsa I. Schwab, "Adaptivity through Unobstrusive Learning," 2002.
- [27] Michael J. Pazzani Daniel Billsus, "User Modeling for Adaptive News Access," *User Modeling and User-Adapted Interaction*, pp. 147-180, 2000.