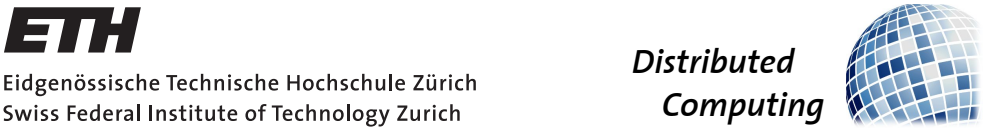


Newspaper 2.0

Master Thesis

Gianluca A. Vinzens

vinzensg@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Jochen Seidel,
Jara Uitto,
Prof. Dr. Roger Wattenhofer

July 1, 2015

Abstract

We have developed Newspaper 2.0, a personalised news app. We show how to track the user's reading behaviour *implicitly* to determine his interests. A recommender system uses those interest signals identified by the app to create user profiles and generate personalised recommendations. The user profile in combination with a confidence measure produces a unique news article ranking for each user. The ranking algorithm is a modified version of the algorithm used to determine the "hot" stories in Reddit. The recommender system applies semantic enrichment using ABC News and Freebase, and information filtering using natural language processing. Recommendations produced by the system are a mix of personalised news and trend news. We show how trends are determined with ABC News and the Reddit rankings. The recommender system is stored in both a relational database and a graph database. Results of comparing the personalised news recommendations with randomly picked news articles show that the personalisation improves the news article selection for a user.

Contents

Chapter 1

Introduction

The way people access news on a daily basis has changed over time. Reading news on a paper-printed version has become less popular over the last decade. Nowadays, most people use a browser or a smartphone to access news online [1]. News are not only read in the morning and evening, but throughout the day and even throughout the night [2]. This shift from printed media to online media has led news agencies to put more effort into online or mobile news. At 20 Minuten Switzerland in the so-called Newsroom, where the journalists and editorial agents are merged into one big room, 90% of the effort is put into online news and only 10% into printed media. News at 20 Minuten is primarily created for mobile devices and only then selected for the printed version. Other companies, such as Watson, do not even publish a printed version anymore, all the news is distributed online [3].

News agencies try to find patterns in their user's reading behaviour. They collect a vast amount of useful data from their users; either by asking them directly or by performing analytics on reading behaviour data [3]. Some companies also try to present the user with a user personalised snapshot of their news. Google and Yahoo have both launched a personalisation mechanism for their users [4, 5]. BBC just recently published an overhaul of their mobile app to offer the user more personalised news [6].

In many systems where the user is presented a personalised snapshot of the complete set of news articles, the user is forced to define his interests explicitly. This process of explicit user profiling is too time consuming for the user and leads to a poor user experience [7].

The glut of news articles produced every day that are available at any time, creates an information overload. Since users rarely use a single news source to inform themselves, retaining an overview becomes a challenge for the interested reader. Social media websites offer ways to organise the vast amount of available news. Often, news agencies have their own account to post news on Twitter and Facebook, and users are presented the news that are interesting to their friends. Being retweeted on Twitter or shared on Facebook is important to news agencies, because it helps them to distribute their articles and reach a much larger audience [8]. Moreover, the community of social media users distribute and create news themselves. For instance, community members on Reddit post articles published online, vote on their importance, and discuss their content. Nevertheless, social media alone are not the ideal way to quickly find interesting news. Friends on Facebook do not necessarily

have the same interests. On Reddit, the user has to manually select interesting news channels, his[1] taste is not determined automatically.

To help the readers find interesting news in this information overload, we have developed Newspaper 2.0, a personalised news app. A simple *user interface* (UI) with a clear structure lets the user navigate through news articles, one at a time. The order in which the articles are presented is tailored to each user's interests. A unique user profile is created and maintained for each registered reader in order to generate personalised news recommendations. Whenever the Newspaper 2.0 app is used to read news, the user interactions on the phone are registered and used to update the user profiles.

The contributions of this thesis are as follows. First, we present a design and implementation of an **Android News App** to display news articles and track the user's reading behaviour *implicitly* to determine his interests. Users can choose from a large range of different news apps. A good user and reading experience is essential for a successful news app. Second, we propose an implementation of a **Recommender System** to use the interest signals identified by the app and create user profiles to generate personalised recommendations. We show how content information can be extracted from news articles using ABC News[2] and Freebase[3], how it can be enriched with natural language processing, and how this information in combination with Reddit[4] generates meaningful recommendations. All news data is scraped from Reddit through its publicly open API.

In Chapter 2 we discuss related work on existing Android apps for news reading and different recommender system approaches, including a short description of Reddit. The design choices of the Newspaper 2.0 app and the process of identifying users' interests while tracking their behaviour are discussed in Chapter 3. Chapter 4 gives a detailed explanation of the inner workings and concrete technologies used for the recommender system. An evaluation of the recommender system is presented in Chapter 5. We conclude with a discussion and propositions for future work in Chapter 6.

[1] Whenever we write 'he' or 'his', we mean 'he or she' or 'his or her'.
[2] http://abcnews.go.com
[3] https://www.freebase.com
[4] https://www.reddit.com

Chapter 2

Related Work

2.1 Android Apps

In 2008, the Google Play Store, formerly known as Android Market, offered its first apps to users running the Android platform. At the time of this writing, the number of available apps in the Google Play Store was over 1.5 million [8]. Only 32.5% of all apps are paid Android apps, while 67.5% – more than twice as many – are free to download [9]. Today, more users access the Internet through a mobile device than through a fixed Internet connection. Four out of five Internet users own a mobile phone. People spend roughly 34 hours a month on media using their phone, of which 89% is through mobile apps and only 11% through the mobile web [10].

Most news apps available in the play store are developed by a news agency to extend their offer from printed version to mobile access. In the case of 20 Minuten Switzerland, news is explicitly created for mobile devices, whereas the printed version is secondary [3, 11]. New news articles are written throughout the day and all users are presented with the same selection of news articles. Which articles to show at what time is derived from user surveys and user interaction analytics [3]. In Newspaper 2.0, we interpret user interaction data on our app and feed it to a recommender system to create user profiles and generate personalised recommendations. The process of selecting the news for each user is completely automated.

Other news apps are produced by companies that *only* offer online news content. One such representative is Watson News. They write news themselves, but only publish it online or through mobile apps [12]. Slightly different is the approach of Yahoo News Digest [13]. Yahoo News Digest is also only made available online, but Yahoo does not write all the articles themselves, it aggregates the best news articles from different news agencies. The news are updated once in the morning and once in the evening, with 7 to 10 articles to read. The articles selected are the most interesting of the day and all users receive the same articles. We collect our news through Reddit, which also provides an aggregation of news articles from different news agencies. Unlike Yahoo News Digest, we create user personalised snapshots of the collected news articles with an infinite supply of news articles, which are updated throughout the day. Every user receives a different set of news articles specifically tailored to his interests.

To set apart from the unpersonalised news apps where all users see the same news, a new trend of personalised news apps has emerged. BBC just recently announced they would release a personalised news app [14]. With the new app, the users have the option to add specialised feeds of their choice in addition to the already offered sections [6]. Apart from being a standard news app like many others, the News Republic app allows users to specify their interests by choosing from a selection of sections [15]. The app then provides the user with a digest of the 12 most important articles from among those sections accessible on a separate dedicated screen. What both apps have in common is the fact that the user *explicitly* has to define his interests. Google News takes this one step further and, additionally to letting the user specify his interests, it *implicitly* learns the users' preferences from their "click behaviour" [16, 4, 17]. Both News Republic and Google News aggregate news from different sources. Similarly, we identify user interests implicitly from the user's reading behaviour for an enhanced reading experience and aggregate news from different news sources.

Social Media has affected people's habits of reading news. Today, 28% of the time spent online is spent on social networks [18]. Many people not only use social media to keep in touch with friends and relatives, but also to consume news. As of 2013 over 60% of U.S. citizens used Facebook, of whom 50% used facebook to read news (Figure 2.1) [19]. On Facebook the news is both shared by the community and directly by the news agencies. For news agencies like The Huffington Post, Facebook is an important tool to reach their news readers [20]. As of 2013, Reddit was used by only 3% of U.S. citizens. However, 62% of all Reddit members used the website to read news (Figure 2.2) [19]. For Newspaper 2.0, we choose Reddit as the news source and add user personalisation to facilitate finding interesting news for the readers.

2.2 Recommender Systems

A recommender system is an engine for a storage system that tries to predict the interests of its users. When the users interact with the storage system, they rate items stored either explicitly or implicitly. For each user, the recommender system generates and maintains a user profile. The goal is to present each user a different ranking of a collection of items, in our case news articles, depending on his interests. Recommender systems can be implemented in various ways. A very common and wide spread approach is a combination of *Collaborative Filtering (CF)* and *Information Filtering (IF)*. CF only takes historical data into account, while IF analyses and matches items depending on their content.

Collaborative Filtering

The CF approach was first mentioned by Goldberg et. al. [22]. The core idea of CF is to match users with similar ratings and generate recommendations using that knowledge. CF comes in two flavours: user-to-user CF [23] and item-to-item CF [24, 25]. User-to-user CF is the most straightforward approach. Users are matched depending on their rating behaviour: if two users have a similar rating behaviour, then they are likely to have a similar taste in general. Item-to-item CF matches items depending on the user ratings: if two items are rated the same by many users, then

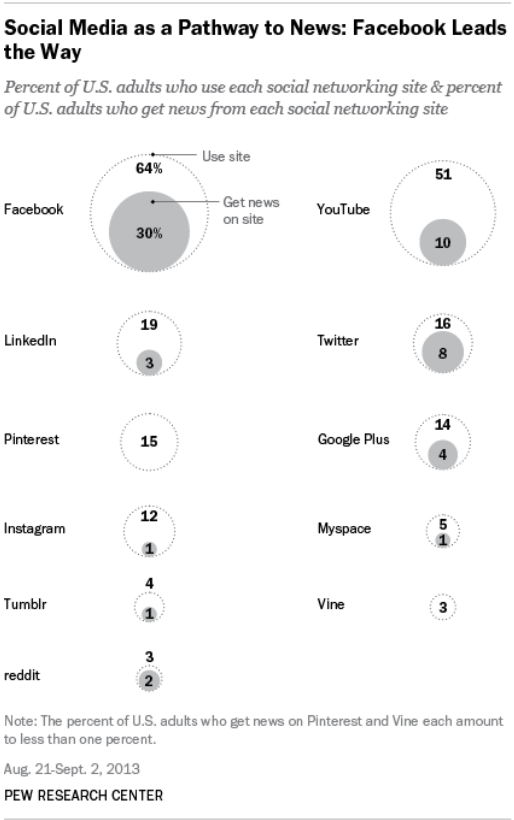


Figure 2.1: U.S. citizens consume a lot of news through social media. The leading social network is Facebook with 30% [19, 21].

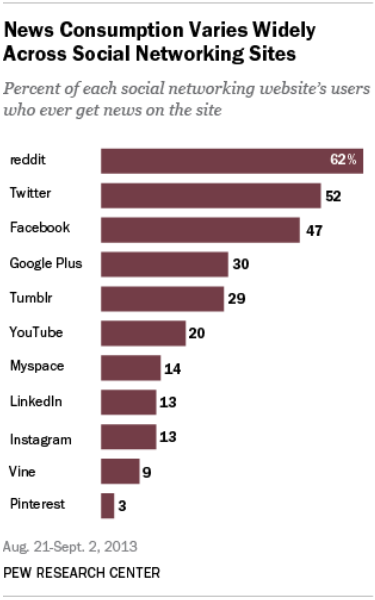


Figure 2.2: Almost two thirds of all Reddit users use the website to read news [21].

the items tend to be similar. This is different form content-based filtering, where the similarity between items is deduced from their content rather than their similarity in user ratings. One requirement for a collaborative filtering system to produce valuable recommendations is a large user base. We build a completely new system with no initial user base. Thus, as long as the number of users interacting with our system remains small, we do not include a collaborative filtering approach in our system. In the future, when more users use Newspaper 2.0, we can extend our recommender system with a CF approach as explained in Chapter 6.

Information Filtering

While collaborative filtering uses the user behavior to match items for similarity, IF directly compares the items' content. If computer processable content information is already available through metadata, items can easily be matched [26]. In most cases though, this content metadata is missing and has to be generated for each item before matching.

Items can be of different information type: texts, images, videos or sounds. Since we are processing news articles we only look at items composed of text strings, we call them *documents*. Most IF approaches come from the information retrieval domain: a user translates his information needs into a search query and sends it to an information retrieval engine, where it is processed and thereby produces a ranked list of documents.

A basic approach to extract content features from a document is to simply compute the *term frequency (TF)*, i.e., the number of times a word appears in the document, for every word in the document [27]. The most common words can then be used to represent the content features of that document. In order to improve the results one can make use of a natural language toolkit to remove stopwords and perform stemming on the words [28].

By using stopword elimination one can reduce the amount of non-relevant words appearing in the content features, but they are not eliminated. Words that appear in less documents are more meaningful and better suited to distinguish documents. TF-IDF (term frequency - inverse document frequency) solves this issue. The number of times a word appears in a single text is offset by the inverse frequency of the word in the complete document space [29, 30]. Just like for simple TF, the higher the TF-IDF score of a word, the better it reflects the content of a document. TF-IDF requires the complete document space to be available when computing the scores for the terms. As thousands of news articles are published every day, this would not be feasible for our recommender system. The TF-IDF scores would have to be recomputed when new articles are added to the system.

Not to rely on the knowledge of the complete document space, Reed et al. came up with a slightly modified approach: which they call IF-ICF [31]. Instead of using the inverse *document* frequency, the formula is modified to use the inverse *corpus* frequency. The corpus is a set of documents representing the complete document space and is chosen to meet the following requirements: The corpus has to (i) be of reasonable size; (ii) be relatively similar to the complete document space; and (iii) the growth of the unique term count has to be sharply reduced, i.e., the number of new terms found in documents, which are not part of the corpus, but part of the complete document space, becomes very small. If a term is not present in the corpus, it receives a corpus frequency of 0. Apart from not having to know the complete document space, TF-ICF is also significantly faster: from quadratic time to linear time in the number of documents in the complete document space. Finding such a representative document space, a corpus, for news articles is challenging. New terms are used in news articles almost every day: previously unknown people become famous, organisations are founded, or journalists invent new terms. In our implementation we present a different information filtering approach using a natural language toolkit to tag words in texts and extract content features. This approach does not rely on knowledge of texts of articles other than the one we extract content features from.

Hybrid and Unified

Recommender systems using the CF alone can prove ineffective for several reasons: cold start problem, sparsity problem, and grey sheep [32]. The cold start problem concerns the issue that the system cannot deduce any recommendations where no interactions between users and items has taken place. This can occur both when new users enter the system or new items are added to the collection. Users usually only interact with a small number of the items in the complete collection, which is known as the sparsity problem. A grey sheep is a user who's opinions do not consistently agree with another group of people, which makes it very difficult to generate meaningful recommendations.

One way to overcome the cold start problem in case a new user enters the system is proposed by Yahoo. Trevisiol et al. found out that users are very much exposed to news articles when they are performing other activities online such as social media or searching for information. They found out that the user behaviour is highly dependent on the referrer URL. Using that information they predict what page a user visits next. Users are matched depending on the external domain where they came from. Apart from not having to rely on the the user's history, users do not even have to log in to make use of the recommendations [5]. We lack the necessary data to build a referrer graph as described by Trevisiol et al.

A different approach is proposed by Liu et al. In addition to using a combination of CF and IF the user is presented a set of trend items. An item is considered a trend when it is rated positively by many other people [4]. In our implementation, we enrich the recommendations with trend news, which are not personalised, but the same for every user. The first time a user interacts with our system, he receives trend news only.

In the case where new items are being added to the collection we can augment the CF with IF to overcome the cold start problem [32]. P-Tango, a hybrid system, combines CF and IF and gives more weight to the better performing one, where the weights are user specific [33]. GroupLens uses filterbots that act as artificial users which rate new articles depending on the articles' content [34]. ProfBuilder uses an interface of two recommendation lists, one generated using CF, the other using IF. The result is a combined prediction [35]. Liu et al. propose to combine CF and IF to generate a ranking for the Google News articles. CF is based on the algorithm described by Das et al. [36]. The IF part is based on the content and the user's genuine news interests and general news trends [4]. Our implementation of the recommender system does not make use of a collaborative filtering approach and hence does not suffer from the cold start problem for new items.

As an alternative approach, in contrast to the hybrid systems, Popescul et al. propose a unified recommender system. To overcome the sparsity problem they get rid of the concept of documents and treat users to be reading words of the document, instead of the document itself. Since there is only a limited amount of words, in comparison to the number of possible documents, this drastically reduces sparsity [37].

2.2.1 Semantic Enrichment

Recommender systems based on CF and IF in its purest form, only the data and information generated by the proprietary system are taken into consideration. The vast amount of publicly available data generated every day by news agencies and through social networks or collaborative knowledge bases can be used to complement recommender systems with external data to semantically enrich the recommendation algorithms.

Abel et al. propose leveraging Twitter to improve user modeling and personalisation. Entities such as persons, events or products are extracted from Twitter messages using OpenCalais[1] [38]. Cantador and Castells explore the contextualisation of item recommendations. In order to do so they take advantage of ontologies[2]. The idea is that the user's preferences and concepts, derived from his activities within a given unit of time, can be semantically linked [39]. Another approach is proposed by Fossati et al. where they make use of a combination of natural language processing and the knowledge graph Freebase to find logical relationships between entities [40]. Freebase[3] is a community oriented database storing information on topics formed chiefly from member submissions. A Freebase topic represents a single concept or real-world thing. Currently, Freebase stores over 47 million topics, of which each has its own web page. Each topic is assigned a unique ID and belongs to a particular type, e.g., people, organisation, or location. A topic holds properties, such as date and place of birth for people, or headquarter and founder for organisations. These properties themselves are links to further topics, creating a logical network of topics. Google's Knowledge Graph is fueled to a limited extent by Freebase. The Freebase database can either be accessed online using a browser or through its publicly available Topic API[4]. One can query the database with standard search terms, while additionally restricting the scope of the topic's type. As a result, Freebase returns a set of topics best matching the query. A registered application using the API can make up to 100'000 requests a day. In Newspaper 2.0, we detect topic candidates in news articles and validate them with Freebase.

Popular News agencies, such as CNN, BBC, or ABC News provide semantics to news articles by partitioning them according to their content. The readers can choose news articles from different news sections. ABC News in particular provides a web page where news articles are listed under their main topic. An interested reader can choose one of these topics to reach a collection of articles covering news about that specific topic. Currently, ABC News provides almost 9'000 topics, of which about 4'200 are names of famous people and celebrities. Every day, ABC News adds an average of 5 new topics to the website. The most up-to-date and heavily discussed news topics are shown at the top of the website in a bar labeled with "now" (see Figure 2.3). The "now" topics are updated several times a day. Topics from frequently read news are listed under the "hot topics" (see Figure 2.3). The "hot" topics usually remain

[1] OpenCalais is a text processing API that extracts entities, topic codes, events, relations and social tags. http://new.opencalais.com

[2] An ontology is a formal definition of entities and their relationships.

[3] On 16 December 2014, the Knowledge Graph staff reported that it would be closing down Freebase over the accompanying six months. All the data will be moved to Wikidata. Google promised to give help to Freebase clients who need to make Freebase declarations suitable for incorporation in Wikidata. As of writing this thesis, the new API is not yet available.

[4] https://developers.google.com/freebase/v1/topic-overview

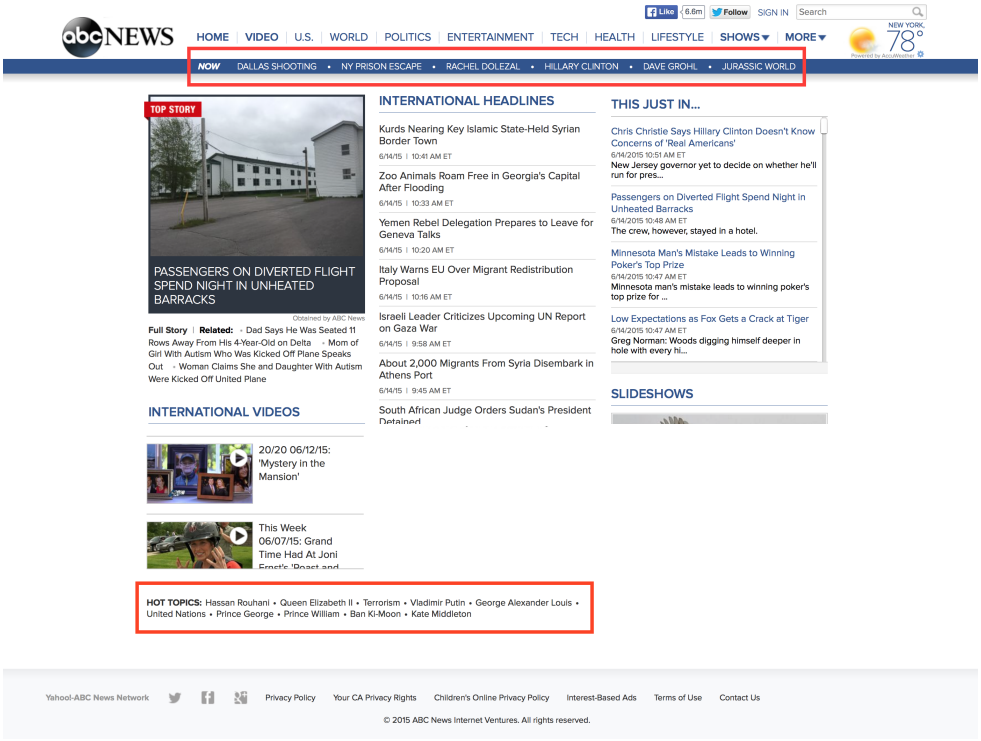


Figure 2.3: The "now" and "hot" topics on ABC News, highlighted with red boxes. The screenshot was taken on June 14 2015.

the same for much longer than the "now" topics. In our implementation, we use the ABC News Topics as an extension to the topics found using Freebase. The selection of "new" and "hot" topics is used to derive trend news.

2.2.2 Graph Databases

Graph databases persist data in a graph model instead of a relational model known from relational databases. A graph consists of vertices, edges, and properties attached to both the vertices and edges. A property can be a label or a key/value pair.

One way to process a graph is via graph traversals. Tinkerpop[5] is an opensource graph computing framework that offers tools to mutate and traverse graph databases, such as Neo4J, TitanDB, or OrientDB. The Tinkerpop framework provides several layers, including a webservice for the graph database called Rexster and a traversal language named Gremlin [41, 42]. Our recommendation system uses TitanDB to store the relationships between news articles and users. The bulbflow[6] library for Python lets us connect to a running Rexster server and execute Gremlin traversal code.

[5] http://tinkerpop.incubator.apache.org
[6] http://bulbflow.com

Ebay uses a graph database to store their recommender system. The products and users with their relationships are stored in the graph database, while the recommendations themselves are precomputed and stored in a relational database for faster access. They are regularly updated by a periodically running process [43]. The most widely used graph database is Neo4J [44]. The Neo4J website advertises the use of a graph database to produce recommendations as one of their use cases. According to their website, graph databases permit a flexible schema with high performance and good scalability [45]. Marko Rodriguez, the co-founder of Tinkerpop, emphasizes in his online blog how easy it is to implement a basic collaborative filtering for a movie recommender engine [46].

2.2.3 Reddit

Reddit is an opensource website [47], where community members post stories of interesting and discussion-worthy online content. Every story on Reddit belongs to a subreddit, a content section comprising user posts of a specific subject. Any member can create a new subreddit, which is independent and moderated by a team of volunteers. Subjects include among others news, gaming, motion pictures, music, books, wellness, sustenance, and photosharing. We are especially interested in subreddits that are specifically created to post links to news articles from websites of all sorts of publishers. News articles posted are not necessarily written by known news agencies, but can also be published by online magazines or even blog posters. Each subreddit can be viewed in different tabs, each containing a different ranking or snapshot of the same posts (see Figure 2.4). The most important are:

- **Hot:** Posts showing up first are ranked using a combination of recency and number of upvotes and downvotes.
- **New:** Most recent posts show up first.
- **Rising:** Posts that are starting to receive upvotes and downvotes show up first.
- **Controversial:** Posts that are getting both a lot of upvotes and downvotes show up first.
- **Top:** Posts with the highest score show up first.

Reddit stories cannot only be accessed through the browser. Any application can retrieve Reddit posts through a publicly available API[7]. We collect all news articles through Reddit using the Python Reddit API Wrapper (PRAW)[8] library for Python, which takes care of the underlying Reddit API calls. The “top” and “rising” Reddit rankings are used to derive trend news.

[7] https://www.reddit.com/dev/api
[8] http://praw.readthedocs.org/en/v3.0.0/

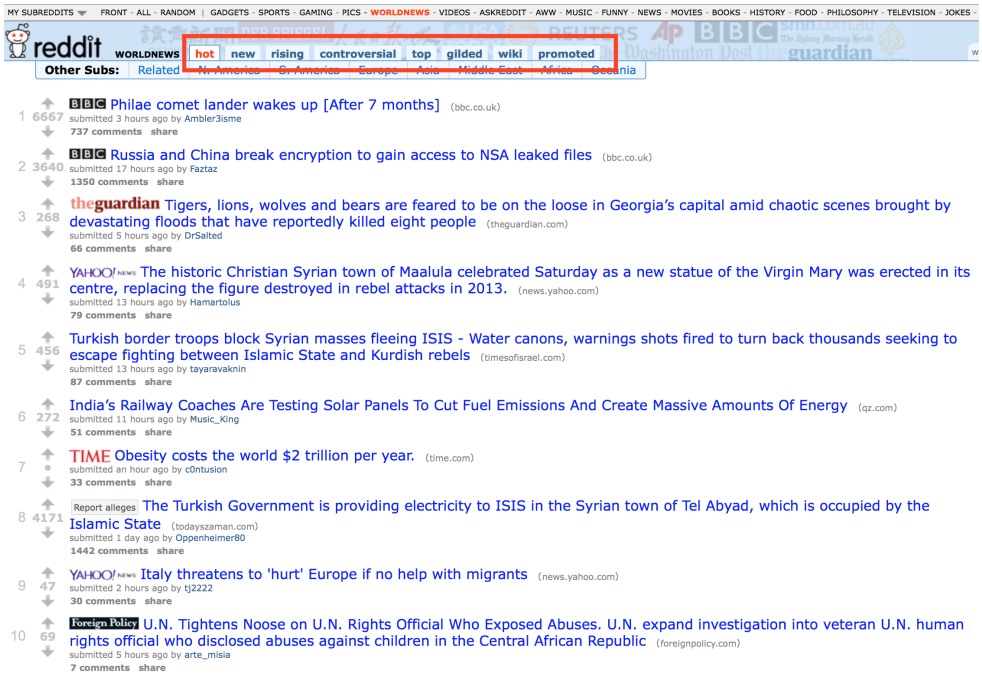


Figure 2.4: A screenshot from Reddit that shows the "hot" stories in the Worldnews subreddit. The red box highlights the tabs that provide different ranking stategies for the stories.

Chapter 3

Android App

This chapter starts by giving a detailed overview of the actual UI of Newspaper 2.0 in Section 3.1. The UI is motivated by a questionnaire conducted with 181 participants in Section 3.2. The personalisation of the news articles for each user requires user inputs. Section 3.3 explains how the user inputs can be identified. Finally, we conclude this chapter in Section 3.4, where we evaluate the implemented UI.

3.1 User Interface

When a user first launches the app the navigation drawer is opened up (see Figure 3.1). The navigation drawer can be pulled from the left of the screen at any time to jump from one screen to another. The user can navigate to the following screens: (i) the home screen; (ii) the archive tabs; (iii) a screen to choose a single section to filter the displayed articles; or (iv) log out and go back to the login screen.

The home screen is where the news articles are displayed (see Figure 3.2). At any time only a single article with image, title and the beginning of its text is shown to the reader. The user decides to either read the article by scrolling down and revealing the complete text or move on to the next one by swiping from right to left. A very similar navigation pattern is known from the popular *Tinder*[1] app, where users like or dislike other people's profiles through a simple swipe in order to connect with new and interesting people [48].

The article text shown to the reader is only a summary of the original article. By scrolling down the complete summary and three options to reach additional background information are revealed (see Figure 3.3): (i) the first button lets the reader load the original, usually much longer, article on the publisher's website in the browser. (ii) The second button redirects the user to the comment section of Reddit, where Reddit members discuss the currently displayed article. (iii) At the very bottom, a list of similar articles allows the user to choose further reading. To move between articles the user can simply swipe the article. Swiping from right to left displays a new article swiping from left to right returns to the previous article.

Additionally to reading the news article and selecting further information to read,

[1] https://www.gotinder.com

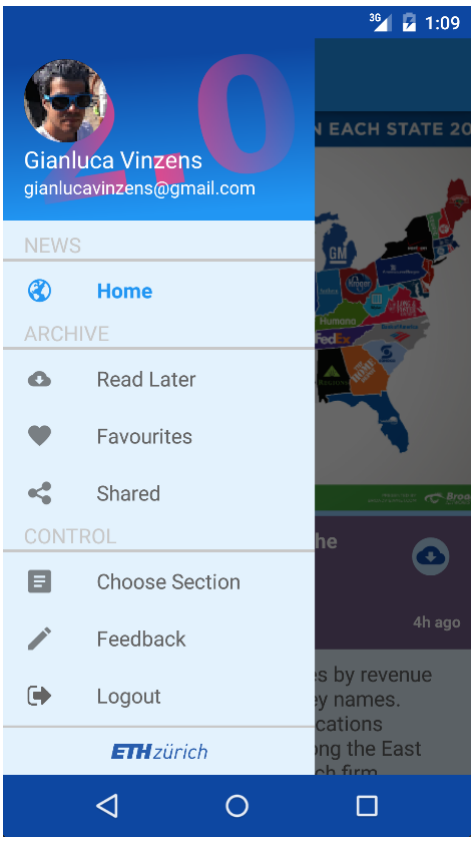


Figure 3.1: The navigation drawer can be pulled from the left and is used to jump from one screen to another.

Figure 3.2: The home screen is where the user reads articles and navigates between articles.

the home screen allows the user to add articles to his archive (see Figure 3.3). The archive itself is composed of three tabs, one for each archive type: an article can be (i) stored to read later; (ii) marked as a favourite; or (iii) shared within the app or via email. Articles in the archive are presented in a simple list (see Figure 3.4).

Every news article belongs to a news section. Different colors for different sections help the user to quickly identify his favourite news articles. The reader can choose a single section to filter the displayed articles. The sections are presented in an order that depends on the user's reading habits: the first three sections are the ones that the user read most articles from, the bottom three sections are the least interesting sections for the user, while the rest is simply ordered alphabetically (see Figure 3.5). A bar at the top of the home screen signals the reader that he chooses to filter the news for a single section only (see Figure 3.6). Clicking on it removes the filter.

News articles on the home screen are updated automatically. If the user keeps swiping, new articles are loaded from the server. When the app is closed, a background process updates the news every 10 minutes. This way, whenever the user opens the app, the most recent news articles are already loaded and available to read.

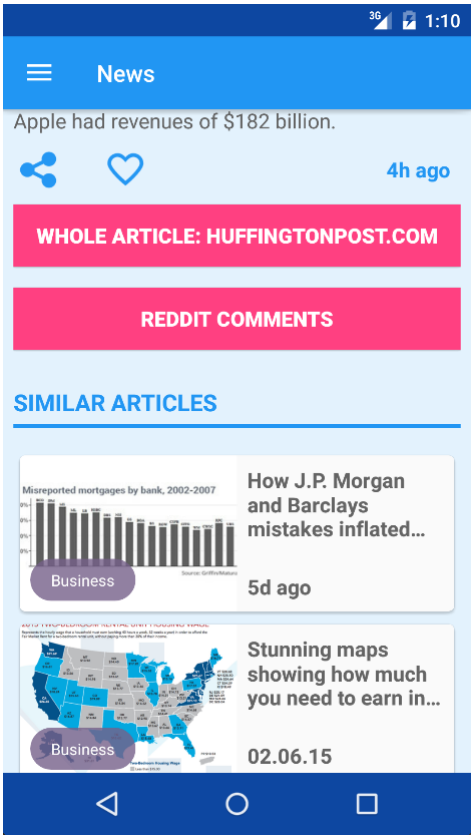


Figure 3.3: At the bottom of the home screen, the user can add articles to his archive or choose to read additional information by loading the original article or the Reddit comments, or select a similar article from a list.

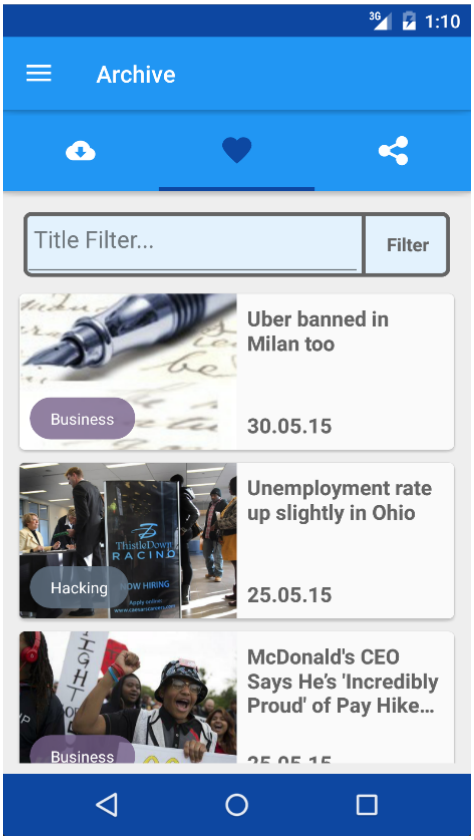


Figure 3.4: The archive consists of three tabs for articles to be read later, articles marked as favourites, and shared articles.

3.2 Questionnaire and Results

Good usability and a nice UI for the news app is essential for a good news reading experience. To figure out what people like and dislike about existing news apps, we conducted a survey with 181 people. Particular interest lies in the users' reading habits and their preferences regarding features and the UI in general.

The questionnaire consists of 4 main parts: (i) App Usage; (ii) App Features; (iii) App Usability and UI; and (iv) Favourite App. This section provides an overview of the results and analysis of the answers. The complete evaluation and result set can be found in the appendix (Appendix A.1).

Out of the 181 participants, 81.8% are male and 18.2% are female. With 20.4% under 25 years old, 42% between 25 and 44 years old, and 33.1% between 45 and 64 we have a decent number of participants from most age groups. Equally many

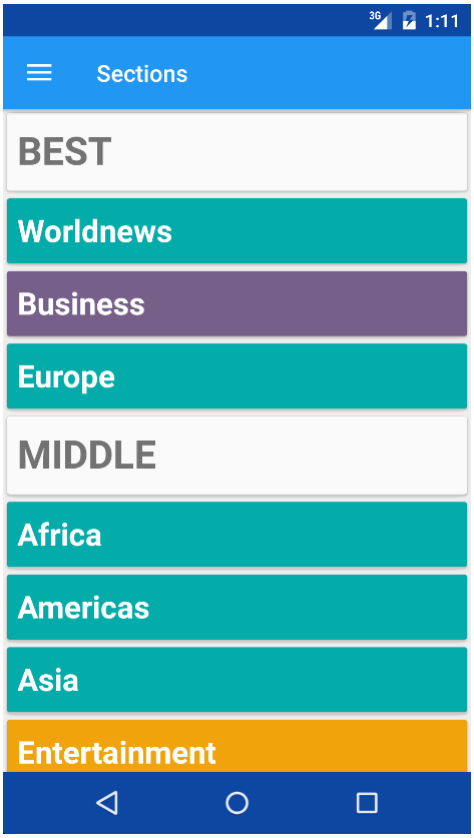


Figure 3.5: A user can choose to only display articles from a specific section. The sections are ordered by how frequently the user has read articles from the different sections.

Figure 3.6: A textbar signals the reader that he filters the news articles for one section only. Clicking it removes the filter.

work in a field best described with "Business and Financial" and "Computer and Mathematical", both 24.3%. Another 14.4% works in a field linked to "Education". The profession of the rest, 37%, is scattered among other fields.

3.2.1 App Usage

Only 3.3% of all the participants have no smartphone or tablet. From the rest, two thirds have more than one news app installed and almost one third have even more than 4 news apps installed. The smart phone and tablet are with 50% the preferred medium to read news. Only one fourth of all participants still prefer to read their news on print-media. With the easy access to news, people tend to check for news several times a day. More than half of all participants check the news multiple times a day — 15% even more than 5 times.

3.2.2 App Features

Contemporary news apps extend the functionality of news reading with multiple extra features. We are interested in what features tend to be more useful than others. A set of common features present in news apps are selected and each participant has to provide an importance score for each feature. We then ask them to select the three most important features from the set, which they do not want to miss in a news app (see Figure 3.7). Finally, we conclude with an open question, where the participants are asked to provide additional useful features known to them.

According to the results, people find a search function to look for specific articles the most useful. Many news apps automatically update their news several times a day. To be able to read interesting news at a later point in time, people would like to store an article to read it later on. In order to keep interesting articles in arms reach, participants want to create a private archive, where they store favourites and share articles with other people. For easier navigation through the news, each article should provide a list of similar articles and an article once read should be visually marked. News articles should be assigned to a section. It both helps to identify interesting articles when scanning over the news and to find a specific news article by explicitly choosing a section. Not important for the app's success are features like rating an article or providing a complete history of all read articles.

We are implementing a news app that focuses on user personalisation. The user should not have to search for interesting news articles, but the app automatically selects the most interesting news for the user. We choose not to provide a search functionality in Newspaper 2.0. Instead, the user can create his personal archive with interesting articles to read later, with his personal favourites, and with shared articles. Amongst others, extracting content information from articles is an essential part of the recommender system we are building to personalise the news. This information is directly used to find similar articles and present them to the user in the app. We only present the user one news article at a time. This is different from most common news apps, where articles are often presented in a list and the user then selects what he wants to read. Swiping lets the user navigate between articles. This navigation pattern does not require any articles to be marked as read or seen,

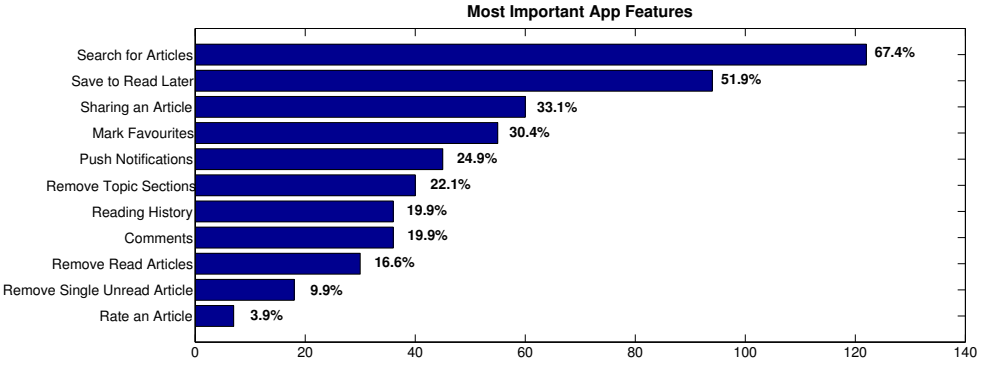


Figure 3.7: The aggregated results of the three most important features chosen by the participants.

interesting articles to the left are seen, whereas all articles to the right are new. In Newspaper 2.0, each news article belongs to a named section. The user has the option to filter for a single section only.

3.2.3 App Usability / UI

Good usability and an appealing UI are two key factors for a successful app. We let the participants choose from a selection of UI alternatives, including article specific parameters and different layout choices. Half of all the participants prefer having only a small number of articles available at any time (less than 20 articles). Only one forth want to be able to select their news from infinitely many texts. News articles should be updated whenever available; users want to see up-to-date news when they open up the app. From different layout alternatives 56% choose the simplest UI, the list, as their favourite. Colors can be a good tool to help the user navigate through the app, but should not be overplayed and act distracting.

The results show that a simple UI is preferred over a more complicated one. We choose an even simpler UI than the list by skipping the news selection phase entirely and directly present the beginning of the article to the user. Presenting the user one article after the other, but doing so in a user-personalised order, is a compromise between letting the user choose from only a small set of articles or infinitely many. The first articles presented are the most trendy and personalised ones. As the user keeps swiping for more news, the articles become less personalised and often less up-to-date. A user only interested in a few articles can stop reading after the first couple of articles. Someone preferring an infinite amount of articles can just keep swiping and reading less personalised news articles. A background process updates news every 10 minutes, such that the user is first presented with the most up-to-date news whenever he opens the app. Each section has a different color. This way the reader identifies his favourite sections with ease.

3.2.4 Favourite App

As seen in Subsection 3.2.1, two thirds of all polled people have more than one news app installed. Nevertheless one of these apps is usually used more frequently than the others. We ask each participant to select his favourite app and give us his reasons. By far the most favourite app is the *20min* app for 38% of the participants. It is followed by the *NZZ* app (12%) and the *Tages Anzeiger* app (6.7%) (see Table 3.1). A nice UI with a clear structure and sections is the top reason for choosing their favourite app. People prefer *20min* because it is always very up-to-date, offers push notifications, and the articles are usually short. On the other hand, NZZ enthusiasts enjoy reading articles with good quality and sufficient research. The common understanding is, that the app should provide a good variety of news and include local news for the reader.

We are using Reddit as the data source (see Section 4.1), which restricts our influence on the choice of news articles. We have no direct influence on the quality of the articles or the locality. In fact, most news is in English and focuses on the United States or Great Britain and there is practically no news available for countries like

App Name	People	Percent
20min	57	38.0 %
NZZ	18	12.0 %
Tages Anzeiger	10	6.7 %
SRF	6	4.0 %
Feedly	4	2.7 %
Blick	4	2.7 %
Reddit	4	2.7 %
Watson	3	2.0 %
Twitter	3	2.0 %
Others	41	25.0 %

Table 3.1: Ranking of favourite news apps.

Switzerland. Therefore, we limit our news coverage to English articles. On the other hand, Reddit definitely provides a good variety of news articles and the community is very active and produces up-to-date news. By providing the user with a summary of the original article we tend to have short articles. The interested reader can always choose to load the original news article or select more of the same topic through the list of similar articles.

3.3 Interest Signals

When the user opens the app and starts scrolling and swiping to read the news, interest signals are identified and sent to the recommender system for persistence. We define two types of interest signals: (i) *viewed signal*, showing user *interest* in the article and its topic in general; and (ii) *trashed signal*, showing user *disinterest*. *Viewed signals* can be of three different levels; from level 1, only expressing low interest, to level 3, high interest (see Table 3.2).

There are five cases we distinguish of how the users shows interest in an article: when the user (i) marks the article as a favourite; (ii) shares it with a friend; (iii) chooses to read the whole article in the browser; (iv) selects to read the comment section about the article on Reddit; or (v) when he uses the list of similar articles to find further reading. In all these cases we can be sure that the user is interested and therefore consider it a level 3 *viewed signal*; the highest level. For the remaining two viewed levels, level 1 and level 2, we analyse the user's scroll and swipe behaviour. Simply speaking, a user shows enough interest in an article to store a *viewed signal* when he looks at the complete article and remains on the article page for long enough. Determining that the complete article has been revealed can be tracked by the scrolling behaviour of the reader; if he scrolls down enough, eventually the complete article text is revealed. Seeing the complete text is a mandatory condition for both, level 1 and level 2, *viewed signals*. The two *viewed* levels can be distinguished by how long the user remains on an article.

Users starts showing interest in an article once they start scrolling down to read the text. We look at the time interval between starting to scroll down and swiping

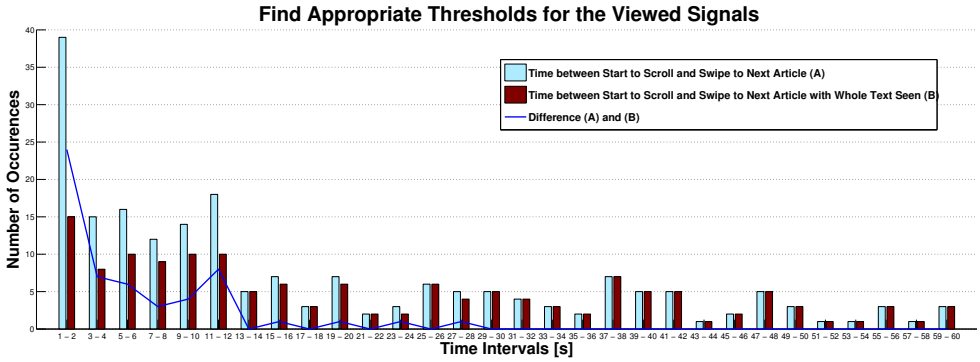


Figure 3.8: The time intervals between starting to scroll down and swiping to the next article and in how many of those cases the users actually scrolled down all the way to reveal the complete text.

to the next article (Figure 3.8). Next we check in how many of those cases the users actually scrolled down all the way to reveal the complete text (Figure 3.8).

As the time a user spends on an article increases, the occurrences of him starting to scroll, but not revealing the complete text decreases. If a user stays on the same article for more than 13 seconds, in all but four cases he reveals the complete text after starting to scroll. We assume the user has read, or at least skimmed over, the article after 13 seconds and as a result identify a level 2 *viewed signal*, standing for medium interest. Prior to reaching the 13 seconds threshold, in more than 45% of all cases, the user started to scroll, but stopped in the middle of the text and moved on to the next article. The text was not interesting enough to finish reading. Nevertheless, the reader must have been slightly interested, which made him start to scroll in the first place. We identify a level 1 *viewed signal*, standing for low interest, if the user starts to scroll and stays on the article between 3 seconds and 13 seconds. We choose 3 seconds as a lower bound, because this is where the ratio between starting to scroll while not revealing the complete text and scrolling all the way down drops below 50%. The lower bound lets us eliminate the cases, where the user mistakenly started to scroll and revealed the complete article text, even though he has no interest in the article.

Viewed Level	**Constraints**
Level 1	After starting to scroll, the user scrolls down to reveal the complete text and remains on the same article between 3 and 13 seconds.
Level 2	After starting to scroll, the user scrolls down to reveal the complete text and remains on the same article for more than 13 seconds.
Level 3	The user explicitly shows interest by marking an article as favourite, sharing it, or choosing one of the options for further reading.

Table 3.2: The three different viewed signal levels.

In all other cases, when no condition for a *viewed signal* is satisfied, we conclude disinterest and send a *trash signal* to the recommender system.

3.4 Evaluation

To evaluate the app, we persist user behaviour data whenever someone makes use of the Newspaper 2.0 app. The details of the data collection of user interactions can be found in Section 5.1, which elaborates on the evaluation of the recommender system. We look at how often a feature has been used during one and a half months of beta testing with 15 registered users (see Table 3.3). During the beta testing period, 2602 articles where presented to the users, of which 565 (21.7%) resulted in a *viewed signal*.

The features read later, mark as favourite, and sharing all belong to the archive. In total, these features have only been used 30 times (see Table 3.3). The sharing feature is the most popular one amongst the archive features. This is most likely the case, because it is the most versatile feature. A user can share an article with his friends or family, but also send himself an email and thereby extract the article from the app. We have implemented the archive features, because they ranked high in the questionnaire. We conclude, that people not always know what they like and will make use of in a news app. Nevertheless, the archive is not completely ignored and certainly adds value to the app. Most contemporary news apps offer some sort of archive.

Features concerning further reading or background information find more favour with the users. The users have clicked on a similar article, the link to the whole article, and the link to the Reddit comment section for a combined total of 90 times. If a user finds great interest in an article, additional sources enhance the reading experience.

Even though the link to the whole article is the most used feature for further reading, it has only been clicked in 39 (6.9%) out of the 565 *viewed* articles. We conclude that the summary is usually enough for the reader to satisfy his interest.

Feature	**Usage**	
Read the Whole Article	39	Further Reading
Read the Reddit Comments	32	
Select a Similar Article	19	
Share an Article	13	Archive
Mark to Read Later	10	
Save as a Favourite	7	

Table 3.3: Number of times each feature has been used during the one and a half months of beta testing by 15 users.

Chapter 4

Recommender System

This chapter starts by showing how we scrape news articles from Reddit in Section 4.1 and explains the characteristics of a user profile in Section 4.2. Section 4.3 elaborates on how we create personalised news article rankings. Three different kinds of trend news are defined in Section 4.4. Section 4.5 describes the recommendation generation mechanism in more detail. A similarity score used to find similar articles is presented in Section 4.6. We finish the chapter in Section 4.7, which talks about the persistence layer of the recommender system. For an overview of the components see Figure 4.1.

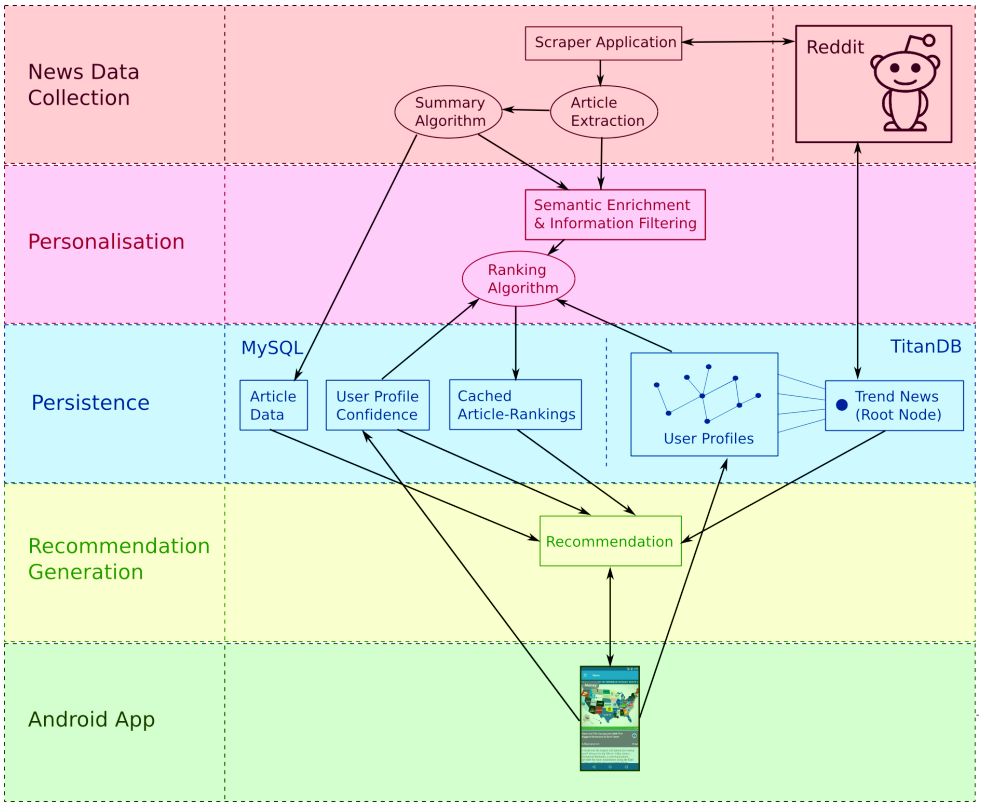


Figure 4.1: An overview of the recommender system and its components.

4.1 News Data Collection

All news articles are collected through the publicly available Reddit API. We restrict the collection of news articles to a total of 82 subreddits. The following list gives an overview of which of the 20 sections (left) available in Newspaper 2.0 is fed by which subreddits (right):

Worldnews:	Global Health, In The News, News, World Events, World News
Europe:	Europe, Ireland, Russia, Scotland, UK Politics, Ukrainian Conflict, United Kingdom
Americas:	Canada, Cuba, Politics, USA News
Asia:	Asia, India, India News, North Korea News, Pakistan, Philippines, Singapore
Middle-East:	Israel, Iran, Kurdistan, Levantine War, Middle-East News, Palestine, Syrian Civil War
Africa:	Africa, Southafrica
Oceania:	Australia, New Zealand West Papua
Worldplitics:	Geo Politics, World Politics
Sports:	Boxing, Football, Formula, Hockey, Olympics, Pro-Golf, Skiing, Soccer, Sports, Tennis
Business:	Business, Economics, Economy
Money:	Finance, Money
Environment:	Environment
Health:	Alternative Health, Health, Health Food
Technology:	Hardware, Tech, Tech News, Technology
Hacking:	Hackernews, Hacking, Intelligence
Science:	Earth Science, Everything Science, Science
Future:	Dark Futurology, Futurology
Entertainment:	Entertainment
TV:	Television
Music:	Album of the Week, Music, Music News, New Music
Odd:	News of the Stupid, News of the Weird
Food for Thought:	Food for Thought, Interesting Article, True Reddit, True True Reddit, Vignettes

Choosing Reddit as the source for the news articles has its advantages and disadvantages. The Reddit community is an active community posting news found on

most well known news agency websites such as CNN, BBC, or ABC News almost in real-time. Additionally to posting articles from actual news agencies, Reddit members also share articles from other sources such as magazines or blog posts. It is even possible to implement a bot, an automated process, which feeds a subreddit. The result is a variety of different news articles of different type and format. Reddit users reside in over 204 different countries and publish new posts round the clock. A registered member has the chance to vote posts up or down and discuss with other Reddit users. The various rankings of the stories provide a glimpse of what many people find interesting at the moment. On the other hand, we as passive users of Reddit have no control over what is posted on Reddit and have no influence on the quality of the articles. Even though the Reddit community is from all over the world, most news subreddits are in English. Further, the news are mostly focused on English speaking countries, such as the United States or Great Britain. For German speaking countries like Switzerland we cannot serve the user with regional or local news.

A scraper application written in Python periodically polls news stories from Reddit using the Python Reddit API Wrapper (PRAW)[1]. From the returned Reddit submissions we select the URL pointing to the actual news article, usually on a different website than Reddit. We can simply feed that URL to a library called *newspaper*[2], which downloads the HTML linked by the URL and extracts the article text and the top image. Articles without a top image are filtered out. Reddit does not restrict subreddits to only be of a single language. Therefore we take extra measure and check if the extracted news articles are written in English, using a Python library called *langid*.

Reddit members produce a lot of duplicate posts. Sometimes those posts are actually the same URL link, pointing to the same news articles. Other times, the same news agency publishes the same news article under two slightly different URLs or different news agencies publish the same news article. In order to get rid of the most obvious duplicate news articles, we only add a new article to the system, if both the URL and the title of the article are not already present in the system.

Before we add a news article to our recommender system, we produce a summary of the original news article. We use a slightly modified version of *textteaser*[3], a summary algorithm using natural language processing and machine learning. Textteaser lets us specify the number of sentences used for the summary. A longer text should also produce a longer summary. We find an appropriate value using an inverse Fibonacci function on the number of sentences in the original article (see Table 4.2). We use the inverse Fibonacci function, because its characteristic to increase more slowly for longer texts suites our purpose. The maximum number of sentences used for the summary is 10. Additionally, textteaser is not explicitly implemented for news articles, but texts in general. The first sentence in a news article is called the *lead* and is the most important structural element, which holds the story's essential facts. Hence, we extend the summary algorithm to always include the first sentence in the resulting summary.

[1] http://praw.readthedocs.org/en/v3.0.0/
[2] http://newspaper.readthedocs.org/en/latest/
[3] http://www.textteaser.com

# of Sentences in News Article	# of Sentences in Summary
1	1
2	2
3	3
4−5	4
6−8	5
9−13	6
14−21	7
22−34	8
35−55	9
> 55	10

Table 4.2: Inverse Fibonacci function used to determine the number of sentences in the article summary.

4.2 User Profile

A very important part of our recommender system is the user profile. It defines what news interests the reader has. Whenever a registered user interacts with the system, his user profile is updated. It is the core element used to generate user personalised recommendations for each reader.

People's interests can be split into two kinds of interests: the long-term interests and the short-term interests [4]. Long-term interests are the genuine interests a user has. They are part of his general interests based on his everyday activities, friends, and family. On the other hand, the short-term interests depend on current events: A person not interested in sports in general may very well be interested in the world cup, or a political vote may be interesting even though the user has not shown much affection to political issues.

People's interests are dynamic. The short-term interests are dynamic by nature; the emergence of a new event influences the person's interest. Similarly the long-term interests are not spared from developing over time; simply by becoming older, everyday activities and people surrounding a person change, and with them his overall interests.

We propose a recommender system where we present the reader a mixture of time-dependent *personalised news* and *trend news*. The personalised news reflect the users long-term interests and the trend news help to satisfy the short-term interests.

Trend news are a product of important events and fulfill the dynamic characteristic of short-term interests. For the long-term personalisation to become dynamic and do not get locked-in, the recommender systems learns to forget user interactions after a certain time. Additionally, recent user interactions are weighted higher.

Besides supporting the user in finding more interesting news, the recommender system directly changes and influences the user's interests through its recommendations

[32]; the user profile gets locked-in. When a user only receives news that are in sync with his profile, he has no chance of diversifying his interests. To a certain extent this is part of the nature of a recommender system. However, the mix of personalised news and trend news gives the user the possibility to diversify news reading. The trend news present the user a selection of news articles completely independent of his interests. In combination with the dynamic properties of the user profile, we reduce the user profile lock-in effect.

4.3 Personalised News

The goal of this section is to describe how we use ABC News Topics and Freebase for semantic enrichmant, and natural language processing for information filtering. The semantic enrichment lets us deduce general topics from news articles and the information filtering extracts additional content features, possibly missed without.

4.3.1 Semantic Enrichment

We find potential topics in news articles and then check with ABC News Topics and Freebase, whether we found a valid candidate. Most words in English texts are written in lower-case. The capitalization rule[4] states: (i) Capitalise the first word of a document and the first word after a period; and (ii) capitalize *proper nouns—and adjectives derived from proper nouns.* We can directly make use from the fact that all proper nouns are capitalized. Proper nouns are names, places, organisations, or social occasions, in other words topics (see Appendix A.2).

We can find all capitalised words in a text using regular expressions. If two or more capitalised words appear one after the other, we select all n-grams of the sequence of words as potential topics. Every such identified candidate is checked with ABC News Topics and Freebase. To use ABC News Topics we simply scrape the ABC News Topics web page and extract the topics from the HTML code to store them locally in a database. ABC News only provides topics relevant to news articles. Freebase on the other hand stores all sorts of topics without any focus on news articles. When querying the Freebase Knowledge Graph, it is possible to confine the scope of the topic search. We choose to restrict Freebase queries in terms of the following news relevant types: (i) Organisations; (ii) Locations; (iii) Sports Teams; (iv) Music Artists; (v) Films; (vi) TV Shows; and (vii) Games.

Names stored in Freebase are not limited to known people, such as celebrities, sports athletes, or politicians. We do not want to consider every name as a topic, but only include names of widely known people. ABC News Topics provide over 4'000 well known people's names on their topics web page. Therefore, we only check for names included in the ABC News Topics and ignore names in Freebase.

Verifying topic candidates with Freebase is time consuming and limited to 100'000 requests a day. On average every news article includes 52 possible topics. Currently, we scrape on average 2'152 news articles every day. To make sure not to exceed

[4] http://www.grammarbook.com/punctuation/capital.asp

the limit inflicted by Freebase, we reduce the number of topic candidates before we query Freebase. First, we reduce the amount of candidates, by extracting the topic candidates from the summary of the article instead of the complete text. Apart from reducing the amount of potential topics to roughly the same number in every article, summarising the original article also focuses the extracted topics more on the core information of the article. The average number of requests per article necessary can be reduced from 52 to 20. Second, all topic candidates are first checked with our local database of ABC News Topics, *before* we query Freebase. Any topic detected in the ABC News Topics, and all n-grams derived from it, are removed from the set of potential topics. By first querying ABC News Topics, we further reduce the amount of topics checked with Freebase from an average of 20 to an average of 15 possible topics per article.

For each article we sort the topics according to their number of occurrences and only persist the 10 most relevant topics. This way we get rid of less important topics.

4.3.2 Information Filtering

Topic detection through ABC News Topics and Freebase finds 5 topics per article on average. Those topics are extracted from the article summary and are constraint to capitalized words. But there are other important content features in texts, which are not capitalised. We capture additional content features from the *complete* article text using the natural language toolkit nltk[5] for Python to tag every word with its word type and then merge words to logical terms.

From the natural language toolkit we first train a uni-gram tagger and then a bi-gram tagger both using the brown corpus[6] provided by the nltk library. The nltk library tags every word in the text. We have special interest in simple nouns (NN), proper nouns (NNP), and adjectives (JJ). Through a single linear traversal, the tagged terms are merged together using the following formulas as proposed in [49]:

$NNP + NNP \Rightarrow NNP$**:**	Merge two proper nouns.
$NN + NN \Rightarrow NNI$**:**	Merge two nouns.
$NNI + NN \Rightarrow NNI$**:**	Merge already merged nouns.
$JJ + JJ \Rightarrow JJ$**:**	Merge two adjectives.
$JJ + NN \Rightarrow NNI$**:**	Merge adjectives and nouns.

where NNI is introduced to tag compound simple nouns. Since compound terms hold more information and are better suited to distinguish articles or find similar articles, we only choose terms tagged with NNP and NNI and ignore simple nouns (NN). The term “presidential election” is more precise than “election”, or “security agency” is more specific than a simple “agency”. The chosen terms are then aggregated and sorted according to their number of occurrences. The number of content features we

[5] http://www.nltk.org

[6] A computer-readable corpus of texts prepared for linguistic research on modern English. http://www.helsinki.fi/varieng/CoRD/corpora/BROWN/

store depends on how successful prior topic detection was:

$$\text{number\_of\_contents} = \max(5, 10 - \text{number\_of\_topics})$$

where *number_of_contents* denotes the number of content features to be stored and *number_of_topics* denotes the number of topics extracted prior by using semantic enrichment. If the topic detection was successful and we found between 5 to 10 topics, we only store a minimum of 5 content features per article. If no topics were found we store up to 10 content features. Limiting the number of content features has the same reason as for limiting topics; we get rid of irrelevant content features.

4.3.3 User Profile Confidence

When a user reads news on the app, the app sends interest signals to the recommender system. We use that information to determine a confidence measure of how good the user's profile predicts his interests. The confidence depends on how many of the presented personalised news articles result in a *viewed signal* of any level within a single request session. A request session starts when one batch of articles is requested by the user and ends, when the next batch is requested.

We distinguish three different confidence levels: (i) low; (ii) medium; and (iii) high confidence. Analysis of user interaction data shows that approximately 15% of random articles are read (see Chapter 5). If a user signals interest on less than 15% of the personalised articles, we consider the profile to have a low confidence. Between 15% and 30% the profile receives a medium confidence and everything above 30% is considered a high user profile confidence.

The user confidence measure is used to compute an article ranking for each user and also to generate recommendations, when the user requests a new batch of news articles.

4.3.4 Ranking Algorithm

For every article entering the recommender system, we compute a ranking score for every user. The score tells us how relevant a certain article is for a specific user. A higher score means that the article is more important. The result is an article ranking, that is different for each user.

We propose to use a ranking algorithm very similar to the one used for the "hot" ranking in Reddit [50, 51]:

$$\begin{aligned} n &= \text{upvote} - \text{downvote} \\ \text{water-down} &= 45000 \\ f(n,t) &= \log_{10}(n) + \frac{t}{\text{water-down}} \end{aligned}$$

where t is the timestamp at which a new story was posted on Reddit and *f(n,t)* denotes the final score a story receives. The Reddit algorithm uses two parameters: (i) the logarithm of the difference between upvotes and downvotes, and (ii) weights it

with the recency of the story (water-down factor). The logarithm is used to dampen the effect of larger upvote-downvote differences. A higher score means that the article is more important. Including the time (recency) of a story in the algorithm makes sure, that newer stories receive a higher score when they have the same upvote-downvote difference. More intuitively, the stories are sorted in order of the time they were posted, but are moved forward in time by the upvotes minus the downvotes. Because of the logarithm, each additional upvote moves the post forward in time by a smaller and smaller amount.

In our version of the algorithm, we translate the sum of upvotes into the sum of weighted *viewed signals* and neglect the downvotes. When a user shows interest in a news article, he implicitly tells us what topics and content features he finds interesting, namely the topics and content features of that article. We can use this information to compute the sum of weighted *viewed signals* for a new article for a specific user. After we extract the topics and content features for the new article, we check for how many of those topics and content features a user has shown interest. A user can show interest in a single topic or content feature multiple times, i.e., when he reads multiple articles of the same topic or content feature. Depending on the confidence level of the reader's user profile (*conf_level*), what level of *viewed signal* an article receives (*viewed_level*), and how long ago the user *viewed* an article (*timestamp*), we derive a different weight. If no *viewed signals* for any article sharing topics or content features with the new article are present for a user, the article is not put into that user's ranking.

In general, *viewed signals* from longer ago weigh less than more recent ones. We distinguish between *viewed signals* (i) less than 1 day old; (ii) between 1 day and 3 days old; (iii) between 3 days and 1 week old; (iv) between 1 week and 3 weeks old; and (v) everything older than 3 weeks. For each confidence level we define a function *mon_decr(...)* that is monotonically decreasing in time and multiply it with a constant factor depending on the *viewed signal* level (*factor(...)*):

$$\begin{aligned}\text{weighted\_viewed\_signal}(\text{conf\_level},\ \text{timestamp},\ \text{viewed\_level}) = \\ \text{mon\_decr}(\text{conf\_level},\ \text{timestamp}) \cdot \text{factor}(\text{viewed\_level}).\end{aligned}$$

The monotonically decreasing functions are:

$$\text{mon\_decr}(\textbf{low},\ \text{timestamp}) = \begin{cases} 16, & \text{if timestamp} <= 1 \text{ day} \\ 4, & \text{if 1 day} < \text{timestamp} <= 3 \text{ days} \\ 2, & \text{if 3 days} < \text{timestamp} <= 7 \text{ days} \\ 2, & \text{if 7 days} < \text{timestamp} <= 21 \text{ days} \\ 0, & \text{else} \end{cases}$$

$$\text{mon\_decr}(\textbf{medium},\ \text{timestamp}) = \begin{cases} 16, & \text{if timestamp} <= 1 \text{ day} \\ 8, & \text{if 1 day} < \text{timestamp} <= 3 \text{ days} \\ 4, & \text{if 3 days} < \text{timestamp} <= 7 \text{ days} \\ 2, & \text{if 7 days} < \text{timestamp} <= 21 \text{ days} \\ 0, & \text{else} \end{cases}$$

$$\text{mon\_decr}(\textbf{high},\ \text{timestamp}) = \begin{cases} 16, & \text{if timestamp} <= 1 \text{ day} \\ 12, & \text{if 1 day} < \text{timestamp} <= 3 \text{ days} \\ 6, & \text{if 3 days} < \text{timestamp} <= 7 \text{ days} \\ 2, & \text{if 7 days} < \text{timestamp} <= 21 \text{ days} \\ 0, & \text{else} \end{cases}$$

We choose the following constant factors for the *viewed signal* levels:

$$\text{factor}(\textbf{1}) = 0.5$$
$$\text{factor}(\textbf{2}) = 1.0$$
$$\text{factor}(\textbf{3}) = 1.5$$

Viewed signals older than 3 weeks are ignored, they receive a weighted *viewed signal* of 0. This way, the user profile becomes more dynamic and can adapt to the natural changes in the users' interests. The time intervals are chosen more fine grained for more recent user interactions, hence, the system can more quickly react to a change of interest. For a medium confidence in the user profile, we choose the decreasing sequence of powers of two, starting with 16. For a low confidence and a high confidence we adapt the decreasing function: we keep the highest and the lowest weight the same, but multiply the two intermediate weights with 0.5 and 1.5 for a low confidence and a high confidence respectively. When a user profile receives a low confidence, the *viewed signals* stored in the profile do not reflect his interests. We hope to deduce the user's real interests from the new and most recent *viewed signals*. On the other hand, a high confidence in a user profile tells us that the *viewed signals* in that profile reflect the user's interest very well. We do not want to loose this information over time and weigh older interactions relatively high. The choice of constant factors is a mapping of the level number by multiplying it with 0.5. A single *viewed signal* can contribute up to 24 to the sum of weighted *viewed signals*. With 3 different confidence levels for the user profiles and 3 different *viewed signal* levels we have a total of 9 different *weighted_viewed_signal(...)* functions decreasing in time (see Figure 4.2).

A single topic or content feature could potentially contribute an infinite *viewed signal* weight. To limit the impact a single topic or content feature can have on the score, we define an upper bound of 48. This is equivalent to two of the highest weighted *viewed signals*.

Analysis of user interaction data shows that the algorithm performs better without taking into account the *trashes* at all. For the analysis we choose all articles seen by a single user and compute the ranking for those articles (i) once including the *trashes* (as downvotes) and (ii) once where we ignore the *trashes* completely. We then compare the ranks of all *viewed* articles to select the better algorithm. Articles are ranked higher in 58% without including the trashes. When a user decides to

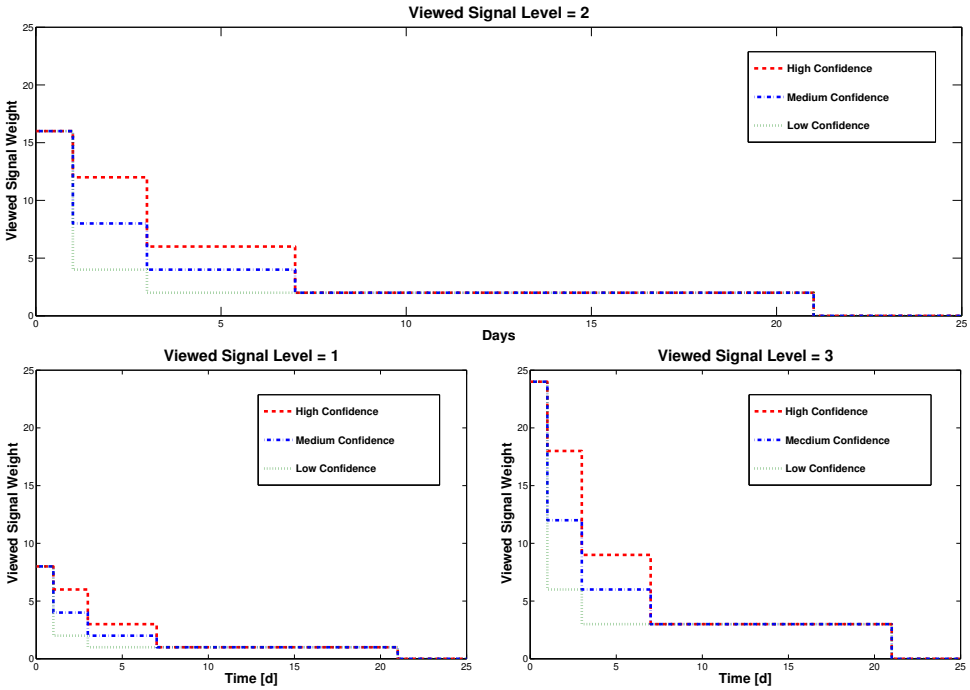


Figure 4.2: Viewed weights depend on how long ago the user viewed an article, the confidence of the user profile, and what level of viewed signal articles receive.

trash an article, it can have several reasons: (i) The obvious one is that the article content is not interesting for the user. (ii) Information reaching the reader is not restricted to the use of our app. A user may likely have read about the same topic on another media before. In that case, the *trash* simply means, that the user already knows enough about the article's topic to bother reading it again. (iii) The general mood and time of the day also play an important role in what a reader finds interesting. The latter two points do not signal a lack of interest in general. Unfortunately, there is no way to find out which reason led to a *trash* other than explicitly asking the user. We decide to completely exclude the *trashes* from the ranking computation.

The choice of water-down factor defines how large the sum of weighted *viewed signals* has to be for a news article to maintain a high rank while it becomes older and newer articles are added to the user rankings. In Figure 4.3 we see how increasing the water-down factor makes it more likely for older articles to rank high, i.e., the necessary difference between the sums of weighted *viewed signals* for old articles and new articles becomes smaller, in order that the older article still ranks higher than the newer one. The absolute ranking score is irrelevant for the user rankings, we only need to compare the articles' relative scores to each other. Reddit uses a water-down factor of only 45'000 (12.5 hours; the half hour is added to create a nice round value of 45'000). A very active community voting for stories produces big differences between upvotes and downvotes. These big differences, even with the logarithm applied, can outweigh the relatively small water-down factor for very active stories. But even with a lot of upvotes, a story on Reddit barely ranks on the front page for more than a day. In Newspaper 2.0, we use weighted *viewed signals* as

upvotes and ignore the downvotes. Even though a single *viewed signal* can contribute up to 24 times a single upvote (see Figure 4.2), the sums are a lot smaller than in Reddit. If we choose the same water-down factor as Reddit, a day old article can only rank higher than a new article with sums of weighted *viewed signals* of 0, if its sum of weighted *viewed signals* is greater than 80. For an article that is two days old, the sum would have to be greater than 7'000, which is impossible. Therefore, we choose to modify the water-down parameter to allow older news articles to be ranked higher as well. We choose the parameter to be 174'600 (48.5 hours; we keep the half hour introduced by Reddit). With this change, a day old article only needs a sum of weighted *viewed signals*, which is greater than 4 to rank higher than a new article with sums of weighted *viewed signals* of 0. For an article that is two days old, the sum only needs to be greater than 10, and for a three days old article a sum of weighted *viewed signals* of 31 suffices.

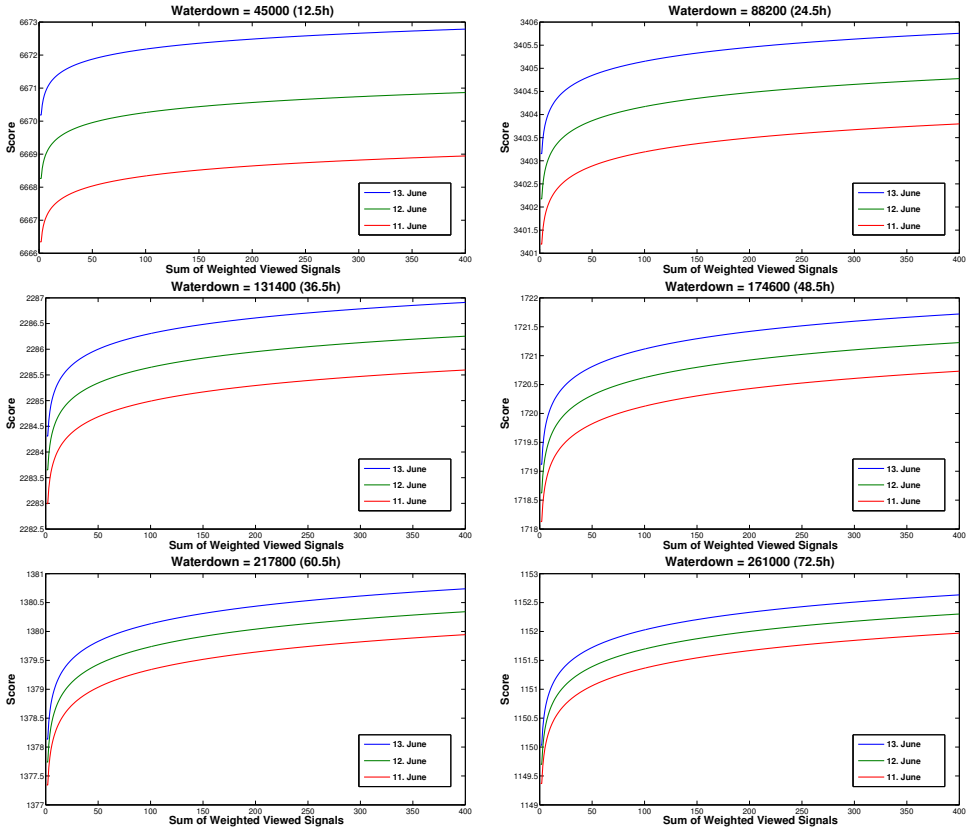


Figure 4.3: Reddit uses a water-down factor of 45'000 (12.5h). For our implementation, we changed the water-down factor to 174'600 (48.5h). This way, the score of an interesting article with a lot of viewed signals can remain high, even one or two days after it was published: New articles, which are not interesting for a user, are more likely to receive a lower score, even though their timestamp is larger. Note, the absolute score is not relevant, but the relative movement between days.

4.4 Trend News

A trend is "something that is currently popular or fashionable"[7], i.e., many people find interest in it. We use the notion of trend to detect news on important and noteworthy incidents. With more than 150 million active visitors a month from all over the world[8], there are enough Reddit stories on news to cover most important incidents happening worldwide. The community members upvote and downvote news articles, discuss, and present their opinions. Reddit provides different ranking strategies of user stories depending on what is "hot", "top", or "rising", in other words trendy, at the moment. We can make use of Reddit and ABC News to find trend news. We define three different kinds of trend news: (i) breaking news that are very recent; (ii) news on important events; and (iii) news on interesting topics in general.

Betternews[9] is a subreddit where only a single bot (user rotoreuters) is allowed to post news articles from different news agencies. Every 15 minutes new news articles are added to the subreddit. Currently, the bot collects news from 24 different news RSS feeds. The description of Betternews hints the reader to use the "rising" tab to find breaking news. Manual checks have verified this functionality. Most of the time the rising tab is empty, but every so often, stories appear and remain present for up to an hour. How exactly breaking news are identified and added to the "rising" tab is not explained. One solution could be to look for news, which are explicitly labeled as breaking news when adding articles to Betternews. After posting the breaking news story to the subreddit, artificial activity is created on the post, possibly generated by the bot itself. Reddit then automatically adds the article to the "rising" stories. We can directly reach those "rising" stories through the Reddit API.

To find the important events worldwide, we make use of the "now" and "hot" topics of the ABC News website. We choose the topics from the international tab, named "World", of ABC News. The topics can be parsed from the HTML code of the website. We then search Reddit using the Reddit API for the most recent ("new") posts covering those topics. The search is restricted to those subreddits that contain news from around the world, such as Worldnews and news from specific countries or regions (see Section 4.1). We include Betternews for a sufficient news coverage.

If trend news are only composed of articles on important events, the news coverage is often limited to politics or crime. There are other interesting news not falling into the important event's category. Such news can cover among others TV shows, music artists, or science. We use the Reddit community's opinion on stories to find interesting news articles in general. The "top" rankings of the subreddits mentioned in Section 4.1, excluded the subreddits used for news on important events, serve as the source of the articles. Stories appearing in the "top" ranking receive a lot of upvotes and only a few downvotes, whereas recency is less of a concern. To still find considerably recent posts, we choose the "top" ranked stories within the last 24 hours. We ignore the subreddits used for the important events, because they are already covered and would prevent us from finding more diverse news.

[7] http://www.merriam-webster.com/dictionary/trend

[8] In the month of May 2015, Reddit had over 167 million unique visitors from over 205 different countries. http://www.Reddit.com/about/

[9] http://www.reddit.com/r/betternews

4.5 Recommendation Generation

In Section 4.2, we have motivated the necessity of providing the readers a mix of personalised news and trend news. The exact ratio depends on the confidence of the user profile (see Figure 4.4): a *medium* confidence produces 50% personalised and 50% trend articles. If the confidence drops to *low*, we reduce the number of personalised articles to only 25%, whereas the rest, 75%, are trend articles. In case the confidence increases to *high*, we do the opposite and select 75% personalised articles and only 25% trend articles.

The reasons of mixing in trend news are two fold: (i) show the user important and noteworthy news of the day, and (ii) offer the user news articles of diverse topics, potentially very different from the personalisation. In Section 4.4, we have defined, that trends can be of three kinds: breaking news, important events, and interesting topics. For the recommendation generation step we consider breaking news to be part of the important events' news, but with a higher priority, i.e., they are chosen first. Hence, we only consider the two types: (i) important events; and (ii) interesting topics. Again, depending on the confidence of the user profile, we choose a different ratio of news about important events and news about intersting topics. When the profile confidence remains *medium*, we choose 60% news about important events and 40% news about interesting topics. We choose slightly more important events, because presenting news about important incidents is the core task of a news app. A *low* confidence of the profile means, that the user has not *viewed* many personalised news. This is either the case, when the profile does not predict the user's interests well enough, or when the user wants to read on other topics than just his main interests. In both cases we want to let the user choose from a larger variety of news articles, but still present enough recent news on important events. We select 40% of important events' news and 60% on interesting topics' news. On the other hand, when the user *views* many personalised articles and we find *high* confidence in his user profile, we produce 80% news about important events and only 20% news about interesting topics. The user is obviously interested in what we predicted, but because we reduce the total number of trend articles, we want to make sure, he sees enough news about important events. Diversification of the articles has a much lower priority.

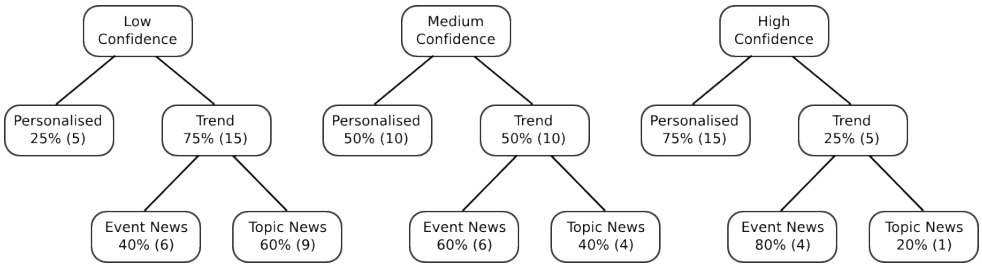


Figure 4.4: Depending on the confidence of the user profile we choose different ratios of personalised news to trend news. The trend news are again split in two different kinds; news about important events and news about interesting topics. The numbers in parenthesis indicate the exact number of articles falling in the respective category, assuming the recommendation generation produces 20 news articles in total.

The personalised news are selected from the user specific ranking of the articles, computed when the articles enter the recommender system. If we simply choose the first articles with the highest scores from the ranking, the chances are high, that we present the user many articles with the very same topics. To filter similar article, we take two measures: (i) first, before we add a new article to the user's ranking, we compute its similar articles (see Section 4.6). If the user holds any of those articles in his ranking, we compare the similar articles' ranking scores to the new article's ranking score. Only if the new article's score is higher, we actually add it to the user's ranking. If so, we additionally remove all similar articles from his ranking. On the other hand, if a similar article has a higher score than the new article, we do not add the new article to the ranking, but keep the similar articles. We have to keep all similar articles, because the similarity does not necessarily have to derive from the same topics and content features; we could remove articles form different topics. (ii) Second, when selecting the articles from the ranking, we make sure to include every article topic and content feature at most twice. This second filter only effects the articles delivered in one request. A next batch will again hold articles of the filtered topics and content features, but again only twice the same.

From Chapter 2 on related work, we learn that a recommender system can suffer from the cold start problem. The cold start problem can occur for both new users and new items, news articles in our case. To overcome the cold start problem for a new user, we can simply generate 100% trend news and ignore the personalised news the first time the user logs in. By construction of the recommendation generation process, the number of personalised news is increased, when the user profile is fed with more interest signals from the new users. A new article does not suffer from the cold start problem in our implementation. Recommendations are generated from the content of the news articles. Articles with no user interactions are automatically included in the recommendations.

4.6 Article Similarity Score

We recall from Chapter 3, that the Newspaper 2.0 app provides a list of similar articles for the users to find further reading. If we would compute the similar articles whenever a user reads an article, the processing would take too long. Hence, we precompute similar articles whenever we add a new article to the recommender system. Additionally, the stored information about similar articles is used to remove articles of similar topics from the personalised recommendations as explained in Section 4.5.

After we extract the topics and content features from a new article, we find those news articles in the system that share topics and/or content features with the new article: To do so, we calculate a similarity score between the new article and all articles already in the system. We define the similarity score (*sim_score*) between

the new article (a_{new}) and any article in the system (a_{sys}):

$$\text{same\_section}(a_{\text{new}}, a_{\text{sys}}) = \begin{cases} 1, & \text{if section}(a_{\text{new}}) = \text{section}(a_{\text{sys}}) \\ 0, & \text{else} \end{cases}$$

$$\begin{aligned} \text{sim\_score}(a_{\text{new}}, a_{\text{sys}}) = & 2 \cdot \text{common}(\text{topics}(a_{\text{new}}),\ \text{topics}(a_{\text{sys}})) \\ & + \text{common}(\text{contents}(a_{\text{new}}),\ \text{contents}(a_{\text{sys}})), \\ & + \text{same\_section}(a_{\text{new}}, a_{\text{sys}}) \end{aligned}$$

where *section* returns the section of an article, *topics* returns the topics of an article, *contents* returns the contents of an article, and *common* returns the number of matching elements. A topic is usually more significant in describing an article's text than a content feature. All topics are found in either ABC News Topics or Freebase, whereas the content features are extracted using natural language processing, which is more error-prone. Therefore, we weigh a topic twice as much as a content feature. We prefer articles from the same section in the similar articles list. Additionally, if two articles belong to the same section, they are more likely to talk about the same topic. Hence, we add 1 to the similarity score. All potentially similar articles are ordered according to their similarity score. If two or more articles have the same similarity score, we favour those with a larger timestamp (more recent). To eliminate similar articles with only a single topic or content feature in common, we only select those system articles that have a similarity score greater than or equal to 4. In total we only persist the 5 most similar news articles for a new article. The similarity relation is symmetric, i.e., it holds in both directions. Even though we limit the similar article retrieval to return at most 5 news articles, over time, a single article can potentially be linked to infinitely many similar articles.

4.7 Persistence

All the data collected and computed is persisted for later access and processing. Most of the data, i.e., the information about users and articles, the cached user rankings, ABC News Topics, and configuration parameters, are stored in a MySQL database. The component relationships of the recommender system, i.e., the article–topic and article–content-feature relationships, and the user profile with the interest signal information, are stored and processed in a graph database; in our case TitanDB.

A simplified version of the graph model we use for our recommender system is depicted in Figure 4.5. For every article scraped from Reddit, we create a new vertex in the graph model. Additionally, each topic and each content feature receives a separate vertex. Articles are then connected to the appropriate topic and content feature vertices.

The user profile is stored by creating a separate vertex for every user. When a user *views* or *trashes* an article, a new relationship is created between the respective user vertex and the targeting article vertex. By doing so the user profile is directly updated whenever a user's interest signal is stored in the graph model. The user's interst in topics and content features can be derived from the graph by following the edges from the user vertex to the articles he *viewed* and then again follow the edges from the articles to the topics and content features.

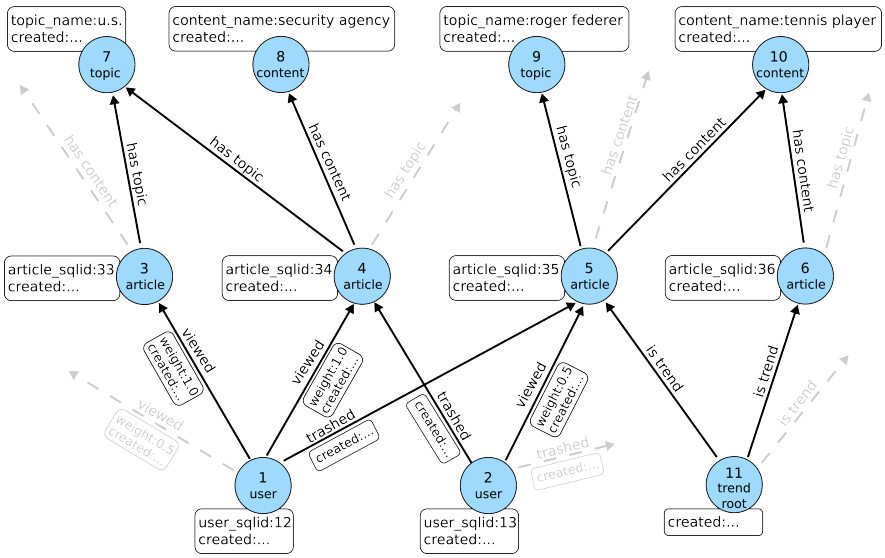


Figure 4.5: A simplified version of the graph model used for the recommender system.

For the trend news articles, we simply store a trend root vertex, which is connected to the currently trendy articles. The connections are updated periodically.

In Section 4.3 we have explained how interest in topics and content features are derived from *viewed signals* of the users and how that information is used to compute the sum of weighted *viewed signals* for new articles for each user. The following code snippets show how the Gremlin traversal language can be used to reach users, who have *viewed* articles with the same topics and content features as the new article, for which we compute the ranking scores.

A reference to the graph is stored in the variable *g*. The traversal code starts by finding the relevant new article vertex in the graph model:

```
g.V('article_sqlid',article_id)
```

We then find other articles with the same topics and content features:

```
.out('has_content', 'has_topic').in('has_content','has_topic')
```

The traversal continues by finding all users who have *viewed* one or several of those articles:

```
.in('viewed')
```

Finally, the weighted *viewed signals* are aggregated for every user:

```
.groupCount()
```

By using a graph database for our recommender system, we explicitly store the relationships between the vertices. Overall, this is a more intuitive model for the recommender system than using a relational database. Additionally, the graph database

lets us read and join data very efficiently. The edges explicitly store the join operations in the model, what makes the queries not only more efficient, but also more simple.

Chapter 5

Evaluation

In this chapter we evaluate the recommender system. To measure the quality of the recommendations, we compare the percentage of *viewed signals* for personalised news articles to randomly chosen news articles. We first explain how the user interaction data is collected in Section 5.1. Section 5.2 defines how we randomly choose an article to present to the user. Finally, we discuss the results of the evaluation in Section 5.3.

5.1 User Interaction Data Collection

The data has been collected over a time span of one and a half months of beta testing. During that time, the app constantly improved and changed its look and feel several times. In total, 15 people have registered for the app and at least logged in once to use the app. Of the 15 users, 4 use the app on a more regular basis of an average of almost 4 times a week.

We recall from Section 4.5, whenever a user requests new data we send a mix of personalised news and trend news. Apart from sending the user all the data necessary to display the news article in the app, we additionally provide the recommendation type, i.e., *personalised* or *trend* for each article. This information is not displayed, but sent back to the recommender system, whenever a user interacts with a news article. When the user opens up the app and starts reading news articles, we keep track of his actions and persist various user interactions by sending them to the recommender system server for persistence. A single user interaction has the following parameters:

User ID:	A unique user id to identify the user interacting with the app.
Article ID:	A unique article id to identify the article the user is interacting with. The article id is 0 if the interaction is not directly connected to an article.
Recommendation Type:	An article presented to the user can either be a personalised article, a trend article, or, for evaluation purposes only, a random article (see Section 5.2).

Interaction Type:	The type of interaction a user has performed on the app. The complete list of interaction types we are tracking will be presented next.
Value:	An additional value may be added to the user interaction in form of a string. This parameter is optional and not used for every user interaction.
Timestamp:	Every interaction is associated with a timestamp. Differences between timestamps can be used to find time intervals between interactions.

The goal is to track as much of the user's behaviour as possible. Not everything is used for the evaluation, but nevertheless it may prove useful in the future. We are interested in the following user interaction types:

Resume App:	A user opens the app for a new session or to resume a previous session.
Pause App:	A user stops reading news and closes the app.
Resume Archive:	A user opens up the archive. It can either be the read later tab, the favourites tab, or the sharing tab.
Pause Archive:	A user leaves the archive.
Viewed Signal:	A user shows interest in the article. The *viewed* level is stored in the value parameter. This interaction is stored either when the user moves to a next article or closes the app. Its timestamp can be used to find out how long the user spent on an article.
Trashed Signal:	When we do not identify a *viewed signal*, i.e., a user shows no interest in the article, we store a *trashed signal*. This interaction is also stored either when the user moves to a next article or closes the app. Its timestamp can be used to find out how long the user spent on an article.
Read Later:	A user chooses to read the article later.
Favourite:	A user marks the articles as a favourite to add it to the archive.
Sharing:	A user decides to share the article.
Whole Article:	A user chooses to load the original article in the browser.
Reddit Comment:	A user opens the Reddit comments in the browser.
Similar:	A user selects an article from the list of similar articles to continue reading.

Left-to-Right Swipe:	A user swipes left-to-right to reach an already seen article.
Right-to-Left Swipe:	A user swipes right-to-left on an already seen article. If the user shows interest in an already seen article, we store a *viewed signal* interaction instead of the left-to-right swipe interaction.
Start Scroll:	A user starts to scroll down to reveal the complete text.
Scroll Whole Text:	A user has scrolled far enough to reveal the complete article text in the app.

All the data is stored in a dedicated table in our MySQL database.

5.2 Random Articles

To evaluate the recommender system's quality, we measure and compare the *viewed percentage* of the personalised news articles to randomly chosen news articles. For three weeks of the one and a half months of beta testing, we additionally mix in random articles to the recommendations, whenever a user requests news articles from the recommender system. During that period, 8 users have registered for the app but only 4 of them were actively using it. A user is considered active when he swipes through at least 100 news articles during the three weeks. We only consider the 4 active users for the evaluation. This way, we eliminate those users who have only opened up the app once or twice out of curiosity, but did not intend to read the news.

If the random articles are chosen completely at random from articles in the system, we could end up presenting the user outdated news articles. The results would be biased in favour of the personalised articles. Therefore, a random article is chosen uniformly at random from all articles available in the recommender system, which are not older than 24 hours.

The random articles might disturb the user experience and reduce the overall user activity. Hence, we limit the number of random articles to 1/4 of the amount of personalised articles we present each user.

5.3 Results

In the 3 weeks, a total of 1782 news articles were presented to the 4 active users, of which 301 returned a *viewed signal*. In total, 14.6% of all random articles and 15.6% of all trend articles were considered interesting by the users. The most *viewed signals* resulted from the personalised articles, with a *viewed percentage* of 18.5%. From our sample, we conclude that the personalisation of the recommender system tends to increase the *viewed percentage* of news articles.

The evaluation was conduced with a more basic solution, which was missing the confidence levels for user profiles and which did not perform any duplicate removal in the recommendation generation. The 3 weeks served not only for evaluation purposed, but also for producing feedback to improve the recommendations. In order to receive better feedback on the recommendations, we stopped producing random articles after the 3 weeks of evaluation. Now that all the components explained in this thesis are implemented and deployed, and the app has been released to the public, the effect of personalisation can be reevaluated with the final version of Newspaper 2.0.

Nevertheless, the user interaction data produced during the evaluation period indicates, that the personalisation of news articles improves the news selection for a user compared to a random selection of articles.

Chapter 6

Conclusion

Based on a survey with 181 participants, we have presented a design and implementation of a news Android app, called Newspaper 2.0. We have shown how different levels of user interests can be inferred from the user's reading behaviour. For the Newspaper 2.0 Android app, we have written 6'872 lines of code. Further, we have implemented a recommender system, which uses the user interest levels produced by the app to create personalised rankings of the available news articles. To extract topics from news articles, we introduce a mechanism that leverages ABC News Topics and Freebase. Content feature extraction from news articles has further been improved with an information filtering approach using natural language processing. We have implemented the recommender system in Python and Gremlin. The complete recommender code consists of 4231 lines of Python code, and an additional 401 lines are of Gremlin code. We have shown how a mix of user personalised news and trend news in general can satisfy both the reader's long-term interests and short-term interests and overcomes the cold start problem. To identify trend news, we have introduced a method that combines ABC News Topics and the Reddit rankings produced by its community members. Results of comparing the personalised news recommendations with randomly picked news articles have shown that the personalisation improves the news article selection for a user.

The beta testing period of one and a half months already produced insightful data on feature usage of the Newspaper 2.0 app and the quality of the recommendations. Now that the app has been released to the public, we hope to reach more people. A larger user base can produce more useful data to both (i) evaluate the recommender system and further improve the quality of the personalised news articles and (ii) enhance the features and UI of the Android app.

Many parameters chosen for the inner workings of the recommender system are merely based on assumptions and not hard factual data. Further analysis and interpretation of user interaction data of more users over a longer period of time can be leveraged to adjust those parameters and support their values with more robust data. Additional computational steps might be necessary to improve the recommendations. On the other hand, implemented mechanisms might prove unnecessary.

Using a first prototype of our recommender system, we found that CF is ineffective with the currently low number of users in comparison to the vast number of articles available in the system. In the future, when the user base of Newspaper 2.0

is large enough, we can extend our recommender system with a CF approach. A set of similar articles is precomputed whenever a new article is added to the recommender system. This relationship is already stored in the graph model for later access. The CF approach can directly make use of this information. When a new article is added to the recommender system, it suffers from the cold start problem, because it is missing any user interactions. To overcome the cold start problem we could exploit its similarity to other articles and make use of their user interactions. Another realisation of a CF extension could be implemented as follows: whenever a user shows interest in an article, the recommender system increases the ranking scores – for that user – for all news articles that have been *viewed* by users who have also *viewed* this particular news article. The exact amount a single score would have to be increased by depends on the parameter choices made for the ranking algorithm and is subject to further research.

The Newspaper 2.0 app identifies 3 levels of user interest, so called *viewed signals*, and one level of disinterest, the so called *trash signal*. The user interaction data suggests that user interest is not a binary function, but a continuous one. We have simplified the model to three discrete levels. The detection of user interest could be made more fine grained and make use of more behavioural data than only timing. The scrolling speed is one parameter that comes to mind to further extend the *viewed signal* identification. Additionally, different users have different reading behaviours: Some users are faster in reading a news article than others, or instead of actually reading an article, most of the time, they simply skim over the text. These differences ask for different time intervals for different users. The app would have to learn from the user's behaviour and automatically adjust its interest detection for each user separately. The reading behaviours could be detected with new eye-tracking capabilities of mobile phones [52]. Analysis of user data has shown that neglecting the *trash signals* as downvotes improves the article rankings for the users. Nevertheless, *trashes* are stored in the graph model. This data might still prove useful for recommendation enhancements or can be used for further reading behaviour analyses. Currently, we only detect a single level of *trash signal*. This model can be extended to find different levels of *trash signals*, e.g., by also looking more closely at time intervals or other user interaction patterns.

Some users read news several times a day, others might only open up the app once a week. One group of people is only interested in the most recent news articles, another group prefers reading older stories of greater interest to them. The reading habits of different groups can vary. The recommender system would have to learn the users' reading patterns from their interest signals and adjust its recommendation mechanism accordingly to satisfy the different groups of readers.

Bibliography

[1] Andrew Beaujon, Poynter: Pew: Half of Americans get news digitally, topping newspapers, radio. http://www.poynter.org/news/mediawire/189819/pew-tv-viewing-habit-grays-as-digital-news-consumption-tops-print-radio/ (2012) [Accessed: 2015-05-23].

[2] American Press Institute: The Personal News Cycle: How Americans choose to get their news. http://www.americanpressinstitute.org/publications/reports/survey-research/personal-news-cycle/ (2014) [Accessed: 2015-05-23].

[3] Salvador Atasoy, Radio SRF 4 News: Medientalk: Erfolgskonzept Newsroom? http://www.srf.ch/sendungen/medientalk/medientalk-erfolgskonzept-newsroom (2014) [Accessed: 2015-05-23].

[4] Liu, J., Dolan, P., Pedersen, E.R.: Personalized news recommendation based on click behavior. IUI, ACM (2010) 31–40

[5] Trevisiol, M., Aiello, L.M., Schifanella, R., Jaimes, A.: Cold-start news recommendation with domain-dependent browse graph. RecSys, ACM (2014)

[6] Leo Kelion, BBC: BBC News app revamp offers personalised coverage. http://www.bbc.com/news/technology-30894674 (2015) [Accessed: 2015-05-23].

[7] Ramin Yasdi: Acquisition of User's Interests. http://www.patrickbaudisch.com/interactingwithrecommendersystems/WorkingNotes/RaminYasdiAcquisitionOfUsersInterests.pdf (1999) [Accessed: 2015-05-23].

[8] Portal, S.T.S.: Number of apps available in leading app stores as of May 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (2015) [Accessed: 2015-05-24].

[9] Portal, S.T.S.: Distribution of free and paid Android apps in the Google Play Store from 2009 to 2015. http://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/ (2015) [Accessed: 2015-05-24].

[10] Bosomworth, D.: Statistics on mobile usage and adoption to inform your mobile marketing strategy. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ (2015) [Accessed: 2015-05-24].

[11] 20 Minuten: 20 Minuten (CH). https://play.google.com/store/apps/details?id=ch.iAgentur.i20Min&hl=en (2015) [Accessed: 2015-05-24].

[12] Fixxpunkt AG: Watson News. https://play.google.com/store/apps/details?id=ch.fixxpunkt.watson&hl=en (2015) [Accessed: 2015-05-24].

[13] Yahoo: Yahoo News Digest. https://play.google.com/store/apps/details?id=com.yahoo.mobile.client.android.atom&hl=en (2015) [Accessed: 2015-05-24].

[14] Media Applications Technologies for the BBC: BBC News. https://play.google.com/store/apps/details?id=bbc.mobile.news.uk&hl=en (2015) [Accessed: 2015-05-24].

[15] News Republic: News Republic - Breaking News. https://play.google.com/store/apps/details?id=com.mobilesrepublic.appy&hl=en (2015) [Accessed: 2015-05-24].

[16] Google Inc.: Google News & Weather. https://play.google.com/store/apps/details?id=com.google.android.apps.genie.geniewidget&hl=en (2015) [Accessed: 2015-05-24].

[17] Google Inc.: Personalizing Google News: Personalization basics. https://support.google.com/news/answer/1146405?hl=en (2015) [Accessed: 2015-05-24].

[18] Shae Bennett: 28http://www.adweek.com/socialtimes/time-spent-online/613474 (2015) [Accessed: 2015-05-24].

[19] Monican Anderson, Andrea Caumont: How social media is reshaping news. http://www.pewresearch.org/fact-tank/2014/09/24/how-social-media-is-reshaping-news/ (2014) [Accessed: 2015-05-24].

[20] Justin Lafferty: Facebook Actually Does Help Deliver News. http://www.adweek.com/socialtimes/facebook-pew-report-2013-news/416380 (2013) [Accessed: 2015-05-24].

[21] Jesse Holcomb, Jeffrey Gottfried, Amy Mitchell: News Use Across Social Media Platforms. http://www.journalism.org/2013/11/14/news-use-across-social-media-platforms/ (2013) [Accessed: 2015-05-24].

[22] Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Commun. ACM (1992)

[23] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: An open architecture for collaborative filtering of netnews. CSCW, ACM (1994)

[24] Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. WWW, ACM (2001)

[25] Karypis, G.: Evaluation of item-based top-n recommendation algorithms. CIKM, ACM (2001)

[26] Ferman, A.M., Errico, J.H., Beek, P.v., Sezan, M.I.: Content-based filtering and personalization using structured metadata. JCDL, ACM (2002)

[27] Sparck Jones, K.: Document retrieval systems. Taylor Graham Publishing (1988)

[28] Loper, E., Bird, S.: Nltk: The natural language toolkit. ETMTNLP, Association for Computational Linguistics (2002)

[29] Luhn, H.P.: A statistical approach to mechanized encoding and searching of literary information. IBM J. Res. Dev. (1957)

[30] Larsen, B., Aone, C.: Fast and effective text mining using linear-time document clustering. KDD, ACM (1999)

[31] Reed, J.W., Jiao, Y., Potok, T.E., Klump, B.A., Elmore, M.T., Hurson, A.R.: Tf-icf: A new term weighting scheme for clustering dynamic data streams. ICMLA, IEEE Computer Society (2006)

[32] Cosley, D., Lawrence, S., Pennock, D.M.: Referee: An open framework for practical testing of recommender systems using researchindex. VLDB, VLDB Endowment (2002)

[33] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M.: Combining content-based and collaborative filters in an online newspaper (1999)

[34] Sarwar, B.M., Konstan, J.A., Borchers, A., Herlocker, J., Miller, B., Riedl, J.: Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. CSCW, ACM (1998)

[35] Ahmad Wasfi, A.M.: Collecting user access patterns for building user profiles and collaborative filtering. IUI, ACM (1999)

[36] Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: Scalable online collaborative filtering. WWW, ACM (2007)

[37] Popescul, A., Ungar, L.H., Pennock, D.M., Lawrence, S.: Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. UAI, Morgan Kaufmann Publishers Inc. (2001)

[38] Abel, F., Gao, Q., Houben, G.J., Tao, K.: Analyzing user modeling on twitter for personalized news recommendations. UMAP, Springer-Verlag (2011)

[39] Cantador, I., Castells, P.: Semantic contextualisation in a news recommender system (2009)

[40] Fossati, M., Giuliano, C., Tummarello, G.: Semantic network-driven news recommender systems: a celebrity gossip use case. CEUR Workshop Proceedings, CEUR-WS.org (2012) 25–36

[41] Tinkerpop: Documentation of Gremlin Traversal Language. http://gremlindocs.comhttp://gremlindocs.com (2015) [Accessed: 2015-06-22].

[42] Apache Tinkerpop: Documentation of Tinkerpop Framework. http://www.tinkerpop.com/docs/3.0.0-SNAPSHOT/#intro (2015) [Accessed: 2015-06-22].

[43] Thomas Pinckney: Graph Based Recommendation Systems at eBay. http://www.slideshare.net/planetcassandra/e-bay-nyc (2013) [Accessed: 2015-05-23].

[44] DB Engines: DB-Engines Ranking of Graph DBMS. http://db-engines.com/en/ranking/graph+dbms (2015) [Accessed: 2015-05-28].

[45] Neo Technology, Inc: Neo4J User Cases: Recommendations. http://neo4j.com/use-cases/recommendations/ (2014) [Accessed: 2015-05-23].

[46] Marko Rodriguez: Graph Database Movie Recommender Engine. http://markorodriguez.com/2011/09/22/a-graph-based-movie-recommender-engine/ (2015) [Accessed: 2015-06-22].

[47] Reddit: About Reddit. http://www.reddit.com/about/ (2015) [Accessed: 2015-06-01].

[48] Tinder: Tinder. https://play.google.com/store/apps/details?id=com.tinder&hl=en (2015) [Accessed: 2015-05-25].

[49] Shlomi Babluki: Efficient Way to Extract the Main Topics of a Sentence. http://thetokenizer.com/2013/05/09/efficient-way-to-extract-the-main-topics-of-a-sentence/ (2013) [Accessed: 2015-06-30].

[50] Amir Salihefendic: How Reddit Ranking Algorithms Work. http://amix.dk/blog/post/19588 (2010) [Accessed: 2015-06-30].

[51] Matt Springer: The Mathematics of Reddit Ranking or how Upvotes are Time Travel. http://scienceblogs.com/builtonfacts/2013/01/16/the-mathematics-of-reddit-rankings-or-how-upvotes-are-time-travel/ (2013) [Accessed: 2015-06-30].

[52] Sun, Y.W., Chiang, C.K., Lai, S.H.: Integrating eye tracking and motion sensor on mobile phone for interactive 3d display, International Society for Optics and Photonics (2013) 88560F–88560F

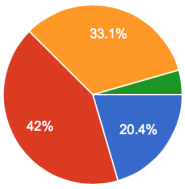
Appendix A

Appendix

A.1 Questionnaire

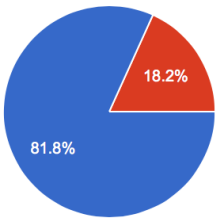
A.1.1 Participants

Age



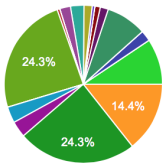
under 25 years	**37**	20.4%
25 - 44 years	**76**	42%
45 - 64 years	**60**	33.1%
64+ years	**8**	4.4%

Sex



Male	**148**	81.8%
Female	**33**	18.2%

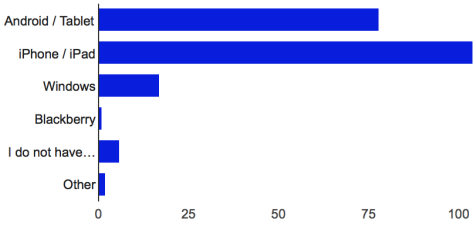
Occupation



Personal Care	0	0%
Community	0	0%
Education	26	14.4%
Business and Financial	44	24.3%
Office and Administrative support	6	3.3%
Life, Physical, and Social Science	6	3.3%
Installation, Maintenance, and Repair	0	0%
Computer and Mathematical	44	24.3%
Sales and Related	1	0.6%
Farming, Fishing, and Forestry	0	0%
Legal	4	2.2%
Architecture and Engineering	5	2.8%
Construction and Extraction	3	1.7%
Arts, Design, Entertainment, Sports, and Media	1	0.6%
Protective Service	0	0%
Healthcare Support	2	1.1%
Food Preparation and Serving Related	3	1.7%
Management	15	8.3%
Building and Grounds Cleaning and Maintenance	0	0%
Healthcare Practitioners and Technical	4	2.2%
Transportation and Materials	0	0%
Other	17	9.4%

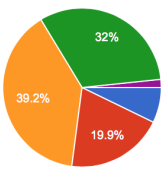
A.1.2 App Usage

What kind of smartphone / tablet do you use?



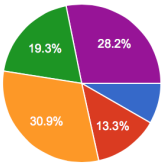
Android / Tablet	78	43.1%
iPhone / iPad	104	57.5%
Windows	17	9.4%
Blackberry	1	0.6%
I do not have a smartphone or tablet	6	3.3%
Other	2	1.1%

How many news apps are installed on your smartphone?



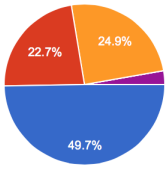
0	13	7.2%
1	36	19.9%
2-4	71	39.2%
5+	58	32%
I do not have a smartphone	3	1.7%

How many news apps are installed on your tablet?



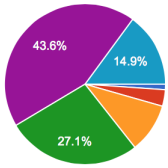
0	**15**	8.3%
1	**24**	13.3%
2-4	**56**	30.9%
5+	**35**	19.3%
I do not have a tablet	**51**	28.2%

How do you prefer to read news?



Smartphone / Tablet	**90**	49.7%
Laptop / PC	**41**	22.7%
Print media	**45**	24.9%
I do not read news	**0**	0%
Other	**5**	2.8%

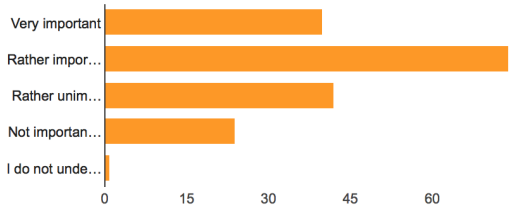
With respect to your preference from the previous question, how often do you read news via this medium?



Less than once a week	**2**	1.1%
Once a week	**6**	3.3%
Multiple times a week	**18**	9.9%
Once a day	**49**	27.1%
2-4 times a day	**79**	43.6%
5+ times a day	**27**	14.9%

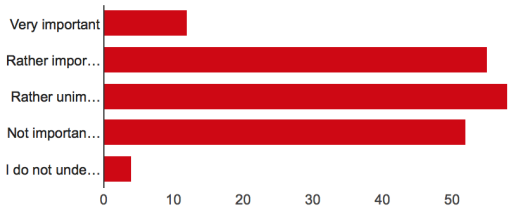
A.1.3 App Features

Save to Read Later [Feature Importance]



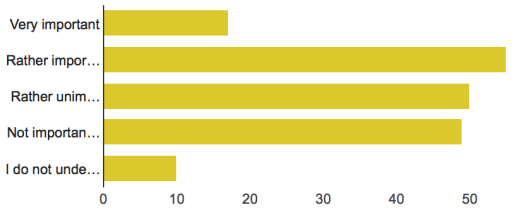
Very important	**40**	22.1%
Rather important	**74**	40.9%
Rather unimportant	**42**	23.2%
Not important at all	**24**	13.3%
I do not understand the feature	**1**	0.6%

Reading History [Feature Importance]



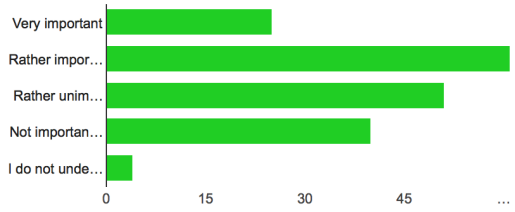
Very important	**12**	6.6%
Rather important	**55**	30.4%
Rather unimportant	**58**	32%
Not important at all	**52**	28.7%
I do not understand the feature	**4**	2.2%

Remove Single Unread Article [Feature Importance]



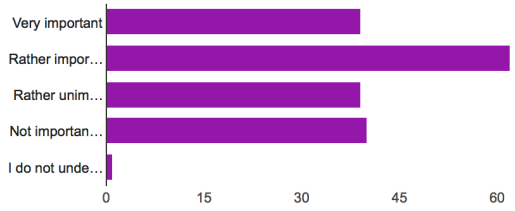
Very important	**17**	9.4%
Rather important	**55**	30.4%
Rather unimportant	**50**	27.6%
Not important at all	**49**	27.1%
I do not understand the feature	**10**	5.5%

Remove Read Articles [Feature Importance]



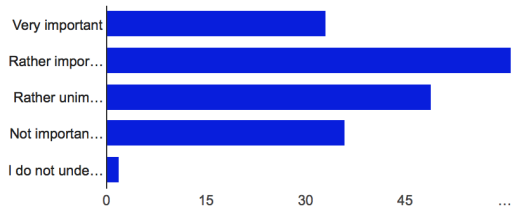
Very important	**25**	13.8%
Rather important	**61**	33.7%
Rather unimportant	**51**	28.2%
Not important at all	**40**	22.1%
I do not understand the feature	**4**	2.2%

Mark Favourites [Feature Importance]



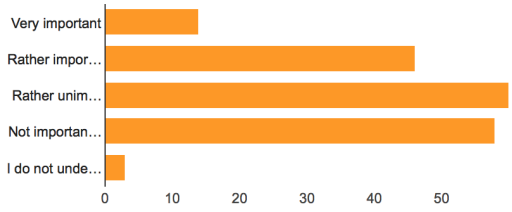
Very important	**39**	21.5%
Rather important	**62**	34.3%
Rather unimportant	**39**	21.5%
Not important at all	**40**	22.1%
I do not understand the feature	**1**	0.6%

Sharing an Article [Feature Importance]



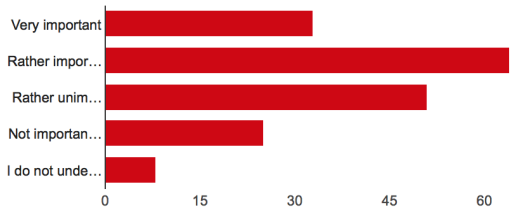
Very important	**33**	18.2%
Rather important	**61**	33.7%
Rather unimportant	**49**	27.1%
Not important at all	**36**	19.9%
I do not understand the feature	**2**	1.1%

Comments [Feature Importance]



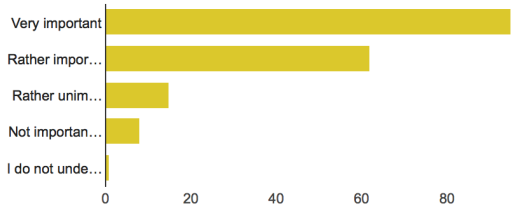
Very important	**14**	7.7%
Rather important	**46**	25.4%
Rather unimportant	**60**	33.1%
Not important at all	**58**	32%
I do not understand the feature	**3**	1.7%

Remove Topic Sections [Feature Importance]



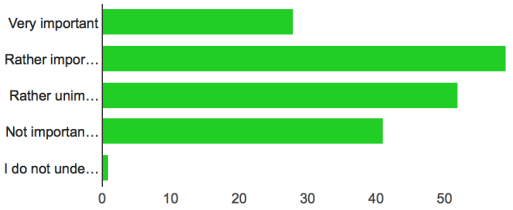
Very important	**33**	18.2%
Rather important	**64**	35.4%
Rather unimportant	**51**	28.2%
Not important at all	**25**	13.8%
I do not understand the feature	**8**	4.4%

Search for Articles [Feature Importance]



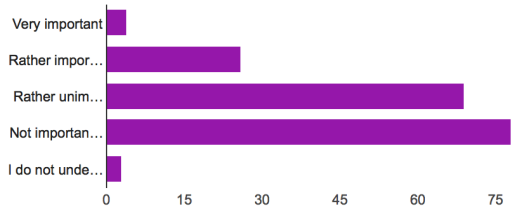
Very important	**95**	52.5%
Rather important	**62**	34.3%
Rather unimportant	**15**	8.3%
Not important at all	**8**	4.4%
I do not understand the feature	**1**	0.6%

Push Notifications [Feature Importance]



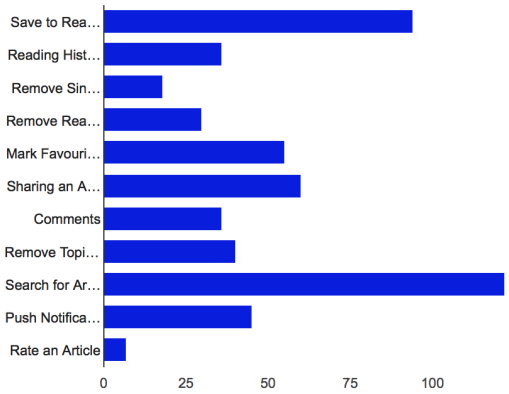
Very important	**28**	15.5%
Rather important	**59**	32.6%
Rather unimportant	**52**	28.7%
Not important at all	**41**	22.7%
I do not understand the feature	**1**	0.6%

Rate an Article [Feature Importance]

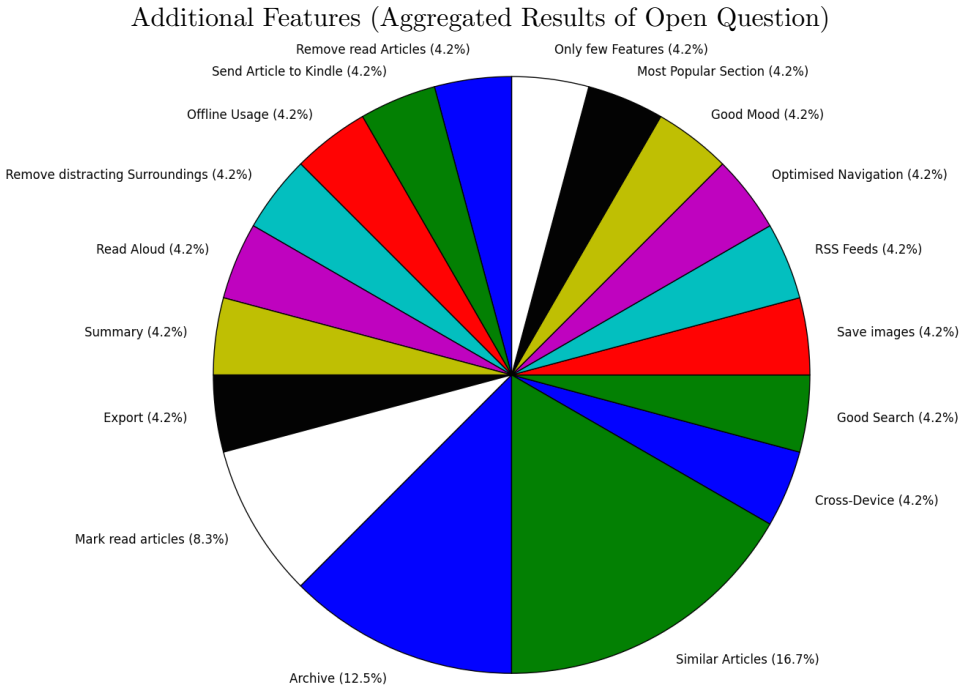


Very important	**4**	2.2%
Rather important	**26**	14.4%
Rather unimportant	**69**	38.3%
Not important at all	**78**	43.3%
I do not understand the feature	**3**	1.7%

Features Ranking

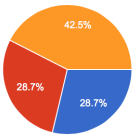


Save to Read Later	**94**	51.9%
Reading History	**36**	19.9%
Remove Single Unread Article	**18**	9.9%
Remove Read Articles	**30**	16.6%
Mark Favourites	**55**	30.4%
Sharing an Article	**60**	33.1%
Comments	**36**	19.9%
Remove Topic Sections	**40**	22.1%
Search for Articles	**122**	67.4%
Push Notifications	**45**	24.9%
Rate an Article	**7**	3.9%



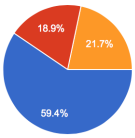
A.1.4 App Usability and UI

Number of articles



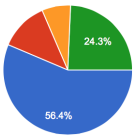
Infinitely many to potentially go on forever.	**52**	28.7%
Up to 100 articles, which can only rarely all be seen / read.	**52**	28.7%
Only a small number (< 20) so I can finish reading everything.	**77**	42.5%

Refreshing of articles

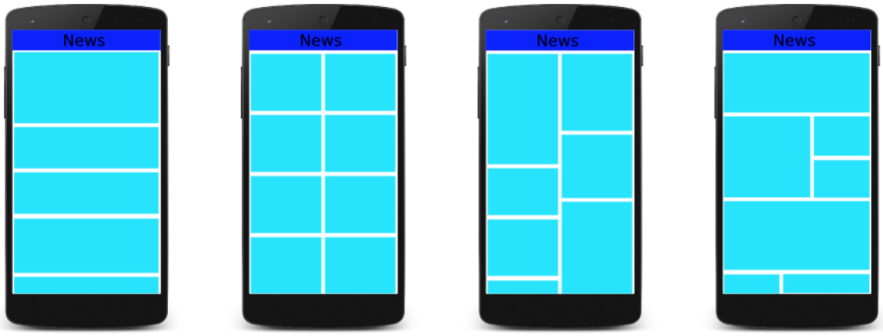


Whenever new articles are available.	**107**	59.4%
Only twice a day (once in the morning, once in the evening).	**34**	18.9%
Never automatically refresh, let the user decide and only allow manual refreshing.	**39**	21.7%

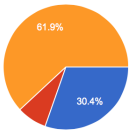
Design Alternatives



A	**102**	56.4%
B	**22**	12.2%
C	**13**	7.2%
D	**44**	24.3%

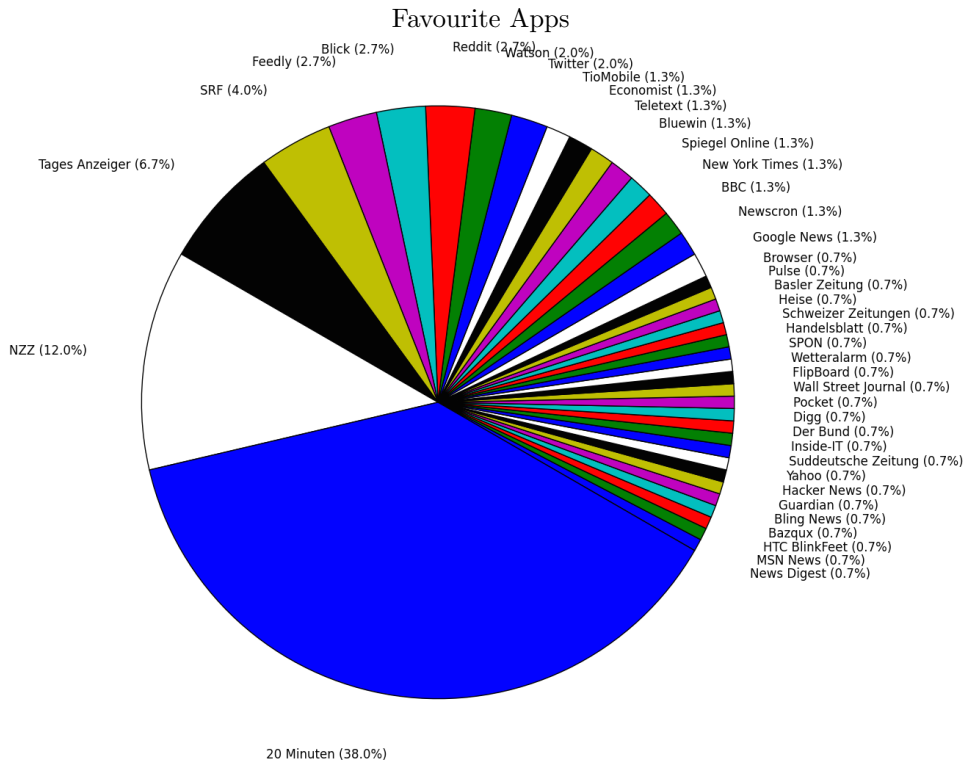


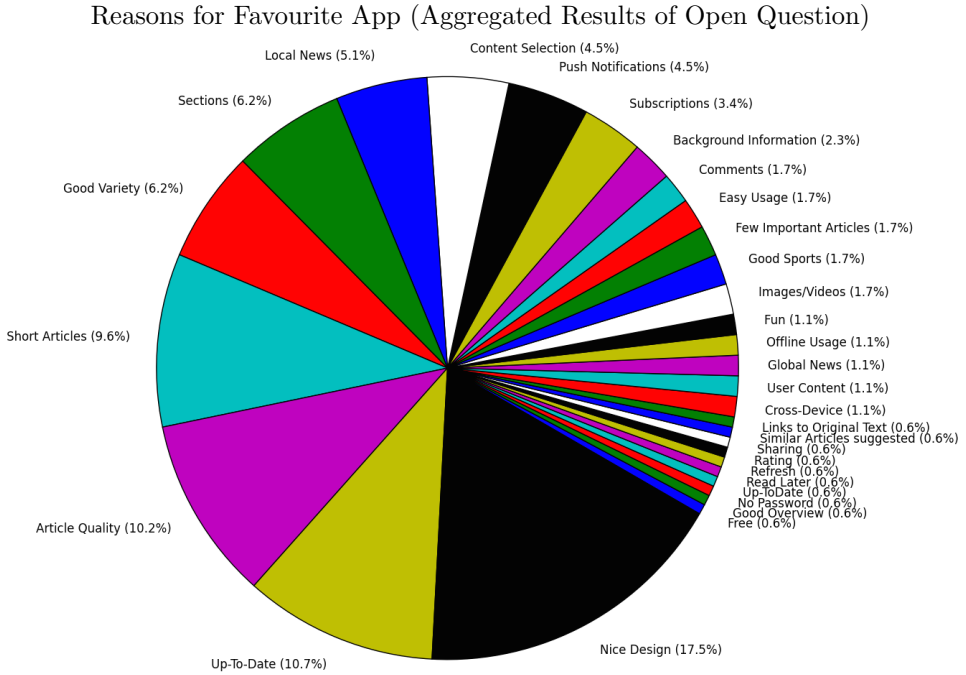
Colors



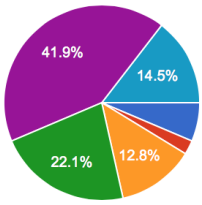
Only use a very limited and restrained set of colors (example A).	**55**	30.4%
Many Different colors make the app more attractive (example B).	**14**	7.7%
Different colors are ok, but the same colors should be used for the same topic / section.	**112**	61.9%

A.1.5 Favourite App



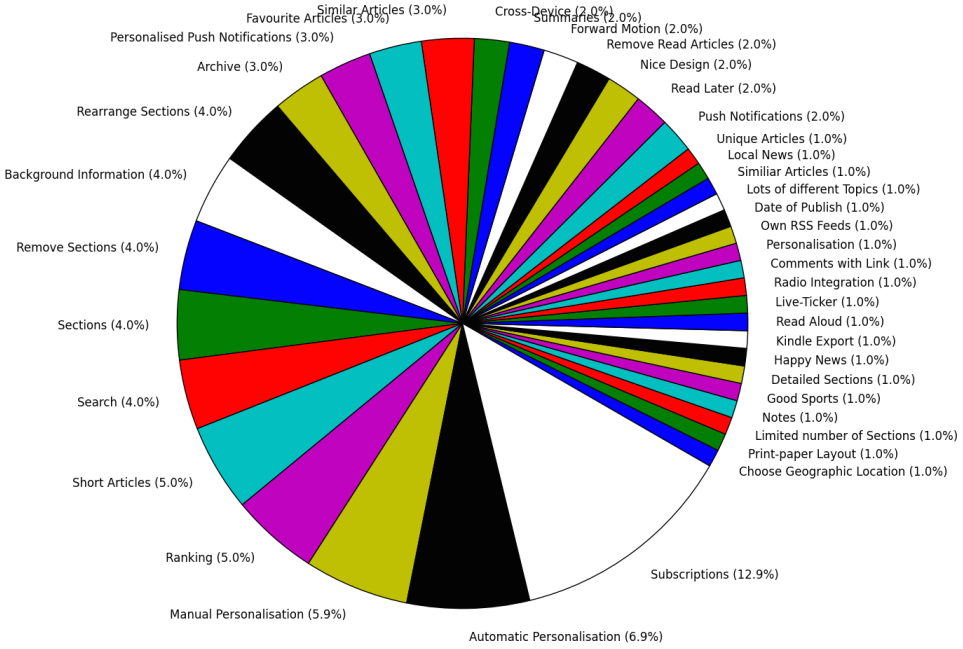


How often do you check your favourite news app?



Less than once a week	**11**	6.4%
Once a week	**4**	2.3%
Multiple times a week	**22**	12.8%
Once a day	**38**	22.1%
2-4 times a day	**72**	41.9%
5+ times a day	**25**	14.5%

Wish List (Aggregated Results of Open Question)



A.2 Capitalisation Rules

Capitalization Reference List (Source http://www.grammarbook.com/punctuation/capital.asp)

- Brand names
- Companies
- Days of the week and months of the year
- Governmental matters
  Congress (but *congressional*), *the U.S. Constitution* (but *constitutional*), *the Electoral College, Department of Agriculture.* **Note:** Many authorities do not capitalize *federal* or *state* unless it is part of the official title: *State Water Resources Control Board*, but *state water board*; *Federal Communications Commission*, but *federal regulations*.
- Historical episodes and eras
  the Inquisition, the American Revolutionary War, the Great Depression
- Holidays
- Institutions
  Oxford College, the Juilliard School of Music
- Manmade structures
  the Empire State Building, the Eiffel Tower, the Titanic
- Manmade territories
  Berlin, Montana, Cook County
- Natural and manmade landmarks
  Mount Everest, the Hoover Dam
- Nicknames and epithets
  Andrew "Old Hickory" Jackson; Babe Ruth, the Sultan of Swat
- Organizations
  American Center for Law and Justice, Norwegian Ministry of the Environment
- Planets
  Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, but policies vary on capitalizing *earth*, and it is usually not capitalized unless it is being discussed specifically as a planet: *We learned that Earth travels through space at 66,700 miles per hour.*
- Races, nationalities, and tribes
  Eskimo, Navajo, East Indian, Caucasian, African American (**Note:** *white* and *black* in reference to race are lowercase)
- Religions and names of deities
  Note: Capitalize *the Bible* (but *biblical*). Do not capitalize *heaven, hell, the devil, satanic.*
- Special occasions
  the Olympic Games, the Cannes Film Festival
- Streets and roads