

# **iNewsReader: Personal Netnews recommendation using Naïve Bayes & Support Vector Machines**

*Managing the problem of information overflow  
by combining adaptive user models in a multi-  
agent perspective*

H.T. de Groot  
February 2011

**Master Thesis**  
Artificial Intelligence  
University of Groningen, The Netherlands

Internal supervisor:

Dr. M.A. Wiering (Artificial Intelligence, University of Groningen)

Second Internal supervisor:

Dr. H.B. Verheij (Artificial Intelligence, University of Groningen)



university of  
 groningen

faculty of mathematics and  
 natural sciences

artificial intelligence



## *Abstract*

The Internet: Web sites; twitter; personal messages; blogs; RSS feeds... Continuously, an overwhelming growing amount of information reaches our human brain. Mostly only a small part of this information is of true personal interest and therefore tracking the interesting part becomes harder every day. This is referred to as the problem of *information overflow*. Approaches to tackle this problem are the usage of news summaries, community interests and collaborative intelligence algorithms. In this research a solution is sought by developing a personalized adaptive netnews recommender system: *iNewsReader*. In general, recommender systems try to deliver personalized items or information of interest based on a profile of the user. In this research an advanced framework design is proposed based on multi-agent technologies and known working technologies from literature. As a proof of concept the main modules (web crawler, data storage, portal & recommenders) and the crucial parts (agents) of this framework are implemented for research. The focus of this recommender system lies on the content-based recommendation methods, which are mainly based on Support Vector Machine and Naive Bayes machine learning algorithms for text classification. The implemented recommender system is accessible through a web-portal and the performance is tested in a small experiment on continuous real time data from the internet. During this experiment multiple pre-configured agents try to collect articles of personal interest by creating user models from the users' feedback and browsing history. Using these personal user models, agents collect news article data from a growing multi-dimensional data storage. Next, the selected articles are presented to the above classification algorithms to form the final personal recommendation. The results of the conducted experiment show a positive effect on learning and recommendation performance, but additions and improvements on many parts of the system can probably elevate this result enormously.

## *Acknowledgements*

First I would thank dr. Marco Wiering for his support and assistance during the whole period of this project. The successful completion is the result of his trust. Also thanks to dr. Bart Verheij for his collaboration and Marieke van Vugt for her insights on the analytical and statistical part of the project.

Special thanks go to my brother for his graphical design, testing and the many project suggestions. Also thanks to Pim Lubberdink for the many substantive discussions on the subject-matter.

Finally I would like to thank my girlfriend Alies, family, colleagues and friends for their patience and support through all my years of study.

# Contents

<b>ABSTRACT</b> .....	<b>II</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>III</b>
<b>CONTENTS</b> .....	<b>IV</b>
1 INTRODUCTION .....	1
1.1 <i>Information overflow</i> .....	1
1.1.1 Problem .....	1
1.1.2 Filter vs. recommendation .....	1
1.1.3 Research .....	1
1.2 <i>Netnews</i> .....	1
1.2.1 News .....	1
1.2.2 Online news representations .....	2
1.2.3 Online news sources .....	2
1.2.4 News collections .....	2
1.3 <i>Recommender systems</i> .....	3
1.3.1 Personalization .....	3
1.3.2 Currently available recommender systems .....	3
1.3.3 Recommender systems for netnews .....	3
1.4 <i>Research questions</i> .....	3
1.5 <i>Overview</i> .....	4
2 THEORETICAL FRAMEWORK .....	5
2.1 <i>Recommender systems</i> .....	5
2.1.1 Item-profiles .....	5
2.1.2 User profiles .....	5
2.1.3 Filter techniques .....	5
2.1.4 Cold start issues .....	6
2.1.5 Hybrid techniques .....	6
2.1.6 Multi-dimensional filtering .....	7
2.1.7 Clustering .....	7
2.2 <i>Data mining</i> .....	7
2.2.1 Data collection .....	7
2.2.2 Pre-processing .....	8
2.2.3 Feature extraction .....	9
2.2.4 Normalization .....	10
2.3 <i>Text classification</i> .....	10
2.3.1 Machine learning .....	11
2.3.2 Cosine similarity .....	11
2.3.3 Naïve Bayes .....	12
2.3.4 k-Nearest Neighbors .....	13
2.3.5 Support Vector Machines .....	13
2.4 <i>Agent models</i> .....	16
2.4.1 Multi-agent systems .....	16
2.4.2 Agent based recommender systems .....	16
2.5 <i>Related work</i> .....	17
3 METHODS .....	18
3.1 <i>Information sources</i> .....	18
3.2 <i>General system setup</i> .....	19
3.3 <i>Framework implementation</i> .....	20
3.4 <i>Data storage</i> .....	21
3.5 <i>Harvesting</i> .....	22
3.6 <i>Portal</i> .....	25
3.6.1 Web server .....	25
3.6.2 User accounts .....	25
3.6.3 User interface .....	25
3.7 <i>Recommendation</i> .....	28
3.7.1 Process overview .....	29
3.7.2 Naïve Bayes recommender .....	30

3.7.3	Support Vector Machine recommender .....	30
3.7.4	Recommender agent configurations.....	30
3.8	<i>Additional agents</i> .....	31
3.8.1	Search agent .....	31
3.8.2	RSS feed agent .....	31
3.8.3	Random agent.....	32
3.8.4	Order agent.....	32
3.9	<i>Shortcomings</i> .....	32
3.9.1	Scaling.....	32
3.9.2	Stacking agents .....	33
3.10	<i>Resources</i> .....	33
4	EXPERIMENT & RESULTS .....	34
4.1	<i>Experiment</i> .....	34
4.1.1	Setup.....	34
4.1.2	Data .....	34
4.1.3	Measurements.....	34
4.2	<i>Results</i> .....	35
4.2.1	Framework.....	35
4.2.2	Data .....	35
4.2.3	Article selection .....	36
4.2.4	Voting strategies .....	37
4.2.5	Agent configurations.....	38
4.2.6	Naïve Bayes vs. Support Vector Machine.....	48
4.2.7	Implicit vs. explicit .....	48
4.2.8	Short-time vs. long-time .....	48
4.2.9	Specific interests.....	50
4.2.10	Summary.....	50
5	CONCLUSION .....	51
5.1	<i>Conclusion</i> .....	51
5.2	<i>Discussion</i> .....	51
5.2.1	Crawler implementation .....	51
5.2.2	Performance .....	52
5.2.3	Multiple interests.....	53
5.2.4	Privacy.....	53
5.2.5	Future of RSS.....	53
5.3	<i>Future Work</i> .....	54
5.3.1	Advanced Agent Framework.....	54
5.3.2	Other agents .....	54
5.3.2.1	Collaborative agent.....	54
5.3.2.2	Additional recommender agents .....	55
5.3.2.3	Known agent.....	55
5.3.2.4	Tagging & categories agents .....	55
5.3.2.5	User-defined interests agent .....	56
5.3.2.6	3 <sup>rd</sup> party agents.....	56
5.3.3	Clustering & dimensionality reduction .....	56
5.3.4	Feed priorities.....	56
5.3.5	Implicit feedback.....	57
6	REFERENCES.....	58
7	APPENDIX .....	61
7.1	<i>Original system proposals</i> .....	61
7.2	<i>Usage instructions</i> .....	63
7.3	<i>Naïve Bayes Classifier implementation</i> .....	64

# 1 Introduction

## 1.1 Information overflow

### 1.1.1 Problem

Nowadays we are living in the era called the information age. An enormous, rapidly growing amount of information is accessible at any time and from anywhere. Of course, the largest source of this continuous and rapid flow of information is the internet. Digital information in the form of personal messages (e-mail, Facebook, etc.), news articles (netnews), discussions (community forums) and many other sources, are accessed by millions of people at a daily basis all around the world. But mostly only a small part of this massive amount of information is of real interest to the final user. Therefore it becomes almost impossible for us humans to keep track of the interesting part, without investing many hours of filtering time. This is referred as the problem of *information overflow*.

### 1.1.2 Filter vs. recommendation

One possible solution to this problem of information overflow is to automatically filter out the uninteresting part by using advanced intelligent filters. A well-known and also widely used example of such a filter is the spam-filter. A spam-filter is used to filter out unwanted e-mail from the personal e-mail inbox; thereby reducing time spent reading unwanted messages. In addition to only filtering the unwanted information, also personalized recommendation of probably interesting items can further tackle the problem. An example of such system is the product recommendation service used by many large web shops (*i.e. amazon.com* [7]). These web shop recommendation services try to recommend products based on previous bought or viewed items and use information from other buyers to present similar products probably of interest to the user.

### 1.1.3 Research

In this research a personalized adaptive netnews recommender system is implemented for managing the above described problem of information overflow. To handle the rapid flow of worldwide news articles, a framework based on multi-agent technologies is proposed, tested and analyzed using multiple agent configurations. All research is based on real time data and experiments are conducted in a large real world continuous and unpredictable environment; the World Wide Web.

## 1.2 Netnews

### 1.2.1 News

The term *news* is generally described as the communication of information of recent events to some audience. In this research a less 'strict' representation of the term 'news' is used. Because the internet contains articles with information of any kind and about any moment in time, the information targeted doesn't necessarily represent a 'recent event'. Information in all its forms and from any moment is accepted as 'news' source for the recommendation process as described in this thesis. Because the final recommendation is mainly based on the most recently available information, this information can be interpreted as

being news and therefore, in this research, the terms news and information are used ambiguously.

### **1.2.2 Online news representations**

There are many ways the news is represented on the internet. News or information in general is mainly represented as written articles in textual form, sometimes illustrated with additional media (i.e. images or video). But also raw images, audio, video or other media forms aren't uncommon item representations around the web. In this research, the mainly targeted resources are articles in the textual form, although articles consisting only of images, audio, video or other media are not omitted. Recommendation of these types of information will be based solely on the available textual data (*title, description, etc.*) for these sources.

### **1.2.3 Online news sources**

Also the online channels to access the news are widespread. First most of the printed newspapers also have an online version of the paper nowadays. These websites of classic news sources are often nicely ordered, articles are categorized and the most important news, selected by an editor, is displayed on top. The content at these news websites is mainly created and selected by professionals (reporters) and therefore an important source of information. Of course there also are lots of valuable online news sites which don't have a printed version of its contents anymore. Current examples of these 'newspaper websites' are the news portals of the BBC [11] and CNN [15]. On the other side there is the widely available user generated content. The blogging platform is probably the mainly used channel to express oneself on the internet. Blogs are small websites containing articles mainly written by nonprofessional individuals on any subject available. The information on these blogs is semi-structured (mainly by using tags) and can be very interesting but sometimes hard to find. Another highly popular medium to keep tracking the news nowadays is twitter. Twitter is an online social network of connected people posting small messages to each other or towards a larger audience. The popularity of this medium is probably because it's fast and it can be accessed from anywhere on a widespread of devices (mobile phones, pda, laptop, etc.). The news value of the 'tweets' (twitter messages) is highly unpredictable and the larger part of the messages are unstructured and dubious. Therefore twitter isn't targeted as a primary news source in this research, although it could be used.

### **1.2.4 News collections**

Subsequently the above sources are used by websites to create collections and overviews of the available news and information on the internet. Google's news service [29], for example, uses a large amount of news sources to summarize the ongoing worldwide events. Other sites (i.e. reddit [51] or digg [21]) are based on user delivered content. On these sites anybody can submit links to articles from anywhere on the web to the community. By using a voting system, automatically the articles of interest to the users of the community will bubble to the top of the pages. Many more similar services are available, summarizing in some sort of way the available content on the internet.

## 1.3 Recommender systems

### 1.3.1 Personalization

All the sources of information as described before are based on generated content for a larger (not personal) audience. Furthermore the performed filtering and ranking at the described 'collection' websites (paragraph 1.2.4) is based on community interests. The content of interest to the largest part of the users will be on top of those pages. To personalize the information flow, recommender systems come into view. Recommender systems try to deliver personalized items or information of interest based on a profile of the user.

### 1.3.2 Currently available recommender systems

Personalized recommender systems are already in use in some areas. For example, there are websites recommending movies (i.e. imdb [33]), music (i.e. last.fm [41]), television shows (i.e. showfilter [57]) or products (i.e. amazon [7]). These recommender systems are mainly based on *collaborative intelligence algorithms*. Collaborative systems try to find users with similar interest by matching user profiles. If a match is found, their items of interest (i.e. bought products or watched television shows) are recommended to the current user of the system. In this way not the articles of interest for the community in general, but those from specific users within the community with similar interests are displayed.

### 1.3.3 Recommender systems for netnews

Also personalized news recommendation systems are already in use. One of the larger systems is implemented by Google's newsreader [19], [30]. This important work describes a massive user, massive item collaborative filtering algorithm, serving millions of articles to millions of users. In more detail, this news recommendation system is based on an advanced modified *k-Nearest Neighbor* algorithm (k-NN; see also paragraph 2.3.4). The Google newsreader solely uses binary user click data (article opened or not?) as a user profile for this collaborative news recommendation system.

Another example of a netnews recommendation system is Genieo [27]. This system is based on a desktop application that creates a magazine style personalized homepage. All kind of information (browsing history, top news, personal interests, etc.) is used to match articles of interest to the users from all around the web.

## 1.4 Research questions

The main goal of this research is to find a solution for the problem of *information overflow*. The target is to reduce the time needed for filtering out unwanted information and increase the time actually consuming articles of personal interest. A solution is sought in the field of *recommender systems* by developing and implementing an intelligent personal netnews filter based on multi-agent technologies. Therefore the main research question is:

*“In what way can a recommender system alleviate the problem of information overflow?”*

To answer this main question, the following sub-questions need to be answered:

1. *Which techniques can be used to reduce human filtering time given a selection of articles?*
2. *How can a recommender system increase the time actually spent on reading the articles of personal interest?*

The first question is related to the number of interesting articles selected by the system and questions the effectiveness of the information selection method(s) related to the subjects' interests in the content of the selected articles. If more articles of personal interest are selected, the user will most likely open a higher percentage for reading.

If an article is opened for reading, the time spent will be an implicit indicator for the subjects' interest for the content. Increasing reading times (relative to earlier usage of the system) together with positive user votes can indicate the success of the profiling and feedback handling techniques. This is covered by the second research question.

The development and implementation of the netnews recommender is based on a mash up of modern (AI) technologies. The question of successfulness of each individual technique within the system is of interest for its overall performance. By using an agent based approach, it is possible to add/remove specific agents and measure the influence and performance of the single agents within the main system. So for each of the individual techniques the following questions are of interest:

1. *“What is the influence of this technique or agent towards the outcome of the system?”*
2. *“What is the performance of this technique?”*

These questions will help answering the main research question and can possibly indicate a synergy of the system.

## 1.5 Overview

In the next chapter, first a detailed description is given about the current available techniques and ongoing research within the field of recommender systems. The next section continues with the theory on data mining and text classification methods. Finally an overview of multi-agent systems is given. Chapter 3 describes the methods used to implement the personal recommender system. The results of a small experiment on the recommendation performance of this system are analyzed in chapter 4. Finally a conclusion is drawn in the last chapter, which also includes a discussion about the shortcomings and points to future work for further improvements.

# 2 Theoretical framework

## 2.1 Recommender systems

Originally *recommender systems* are defined as systems in which people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients [53]. Nowadays a broader view is appropriate where recommender systems can be described as personalized information agents that can provide recommendations: suggestions for items likely to be useful for a user [14]. Overviews of currently used techniques, shortcomings and possible extensions of recommender systems are given in [14], [5] and [13].

### 2.1.1 Item-profiles

To be able to recommend (news) items to a user, a recommender application needs to gather and exploit some information about both the individual and the available items. Generally *item-profiles* are kept relatively simple. Usually a small description, information about where to find the item and some semantic representation (i.e. *a feature vector* representing word counts for a news article) are being stored for each item. For detailed information on item-representation and feature vectors, see paragraph 2.2.3. The process of automatically gathering information to be stored within an item-profile is called *data mining* and is described in section 2.2.

### 2.1.2 User profiles

Most current research on recommender systems is directed to *user-profiling*. All kinds of information about the individual and its actions are being collected and stored to create accurate *User Models*. The two main types of information stored in a user-model are *users' preferences* (i.e. *interests*) and a *history of system interaction* (i.e. *viewed articles*). Thereby, both *explicit* (i.e. name, birth date, item ratings or a list of interests) and *implicit* (i.e. opened articles, reading times or visited links) information is used for mapping these user interests [17] and [25]. In addition different user *memory models* are presented to represent, *long-* and *short-term interests* of users. In [9] for example a framework is presented for adaptive news access built on a hybrid user-model consisting of a *short-term memory* (k-NN; see also paragraph 2.3.4) and a *long-term memory* (*Naïve Bayes*; described in 2.3.3), using both *implicit* and *explicit* user feedback. With this advanced hybrid User Model the system tries to recommend a set of interesting news items where the information which a user already 'knows' is filtered out.

### 2.1.3 Filter techniques

After data mining and profiling, the next step in the recommendation process is the filtering of relevant items. The main four techniques used, described and compared in the papers are: *content-based*, *collaborative*, *knowledge-based* and *demographic* filtering techniques. In [13] also a fifth technique: *utility-based* filtering is described and compared to the others. Short descriptions of these techniques are given below:

*Content-based* (CN): A Content-based recommendation system [48] recommends an item to a user based upon a description of the item and a profile of the user's interests. Content-based recommenders treat recommendation as a

user-specific classification problem and learn a classifier from the user's likes and dislikes based on product features.

*Collaborative (CF)*: Collaborative recommendation systems aggregate ratings or recommendations of objects, recognize commonalities between users on the basis of their rating, and generate new recommendations based on inter-user comparison. In [42] two types of CF techniques are described and combined in a hybrid system: *CF based on user* (CF-U) and *CF based on item* (CF-I) comparison.

*Knowledge-based (KB)*: A knowledge-based recommender suggests products based on inferences about a user's needs and preferences. This knowledge will sometimes contain explicit functional knowledge about how certain product features meet user needs.

*Demographic (DM)*: A demographic recommender provides recommendations based on a demographic profile of the user. Recommended products can be produced for different demographic niches, by combining the ratings of users in those niches.

*Utility-based (UT)*: Utility-based recommenders make suggestions based on a computation of the utility of each object for the user. The main problem here is how to create such a utility function for each user, which is a domain specific task.

In this research most attention is dedicated to Content-based recommendation.

#### **2.1.4 Cold start issues**

The *learning-based* systems (CF, CN & DM) suffer from the so called "cold start" problems in one way or another. These problems indicate a decrease in performance due to an initial lack of information. The main two scenarios are:

- 1) **New User**: When a new user is added to the system, no profile is present. So it is unknown what items to recommend to this person based on his or her interests alone. Possible solutions are explicitly asking for a users' interest information or the use of global popular items. Also demographic information could be used.
- 2) **New Item**: When a new item is added to the system, it has not been rated yet, so it will not be recommended. This is a big problem for single CF systems, particularly when there exists a high item turn over (i.e. a news recommender). A solution is to gather initial ratings from other sources or to recommend new items randomly to some users initially. Another solution is to use non-CF techniques in this case.

#### **2.1.5 Hybrid techniques**

To provide a general solution to overcome the cold start problems from the previous paragraph, researchers have combined these filtering techniques to create better performing systems. Multiple strategies for combining the techniques are compared in [13]. Seven different combination types are described: *weighted, switching, mixed, feature combination, feature augmentation, cascade and meta-level* combinations. The results show the largest synergy for cascade and feature augmented systems.

### 2.1.6 Multi-dimensional filtering

In [4] and [3] Adomavicius et al. try to lift the recommendation process to an even higher level. Opposed to the ‘traditional’ two dimensional user/item systems previously described, this system uses a multi-dimensional approach by incorporating contextual information (i.e. multiple ratings, demographic information, time, etc.). The system described in this paper is based on multi-dimensional feature vectors and supports data warehouse capabilities.

### 2.1.7 Clustering

Recommendation can become a time consuming process if the item space, the user space or both becomes large. One way to reduce computation time is to use clustering methods. By putting similar users and/or items into a single cluster, calculations of similarities between these clusters, instead of single users and items, can reduce computation time enormously. This clustering process itself can be executed offline. Recent research on clustering for recommendation systems [56] describes such an effective clustering method for collaborative tagging systems. In this research a framework based on hierarchical agglomerative clustering of tags is evaluated to bridge the gap between users and resources.

## 2.2 Data mining

Before being able to recommend articles to a user, a list of articles and more detailed information about these articles (item-profile) needs to be available first. The process of collecting and extracting patterns of information from large sources of data is called *data mining*. A typical data mining cycle (Figure 1) exists of four phases [23]. These phases include *collecting* data, *pre-processing* the data, *feature extraction* and *evaluation* or *classification*. The final phase is a key part of the recommendation process and will therefore be described separately in the subsequent section (2.3). The current section will focus on the harvesting part (collection, pre-processing and feature extraction processes) of the data mining cycle.

### 2.2.1 Data collection

The process of collecting data and harvesting information from the internet is called *web crawling* or *web spidering*. A computer program (called a bot, spider or agent) automatically searches the web for usable data. Early research on citation analyses by Garfield in the 1950s is the foundation of two well-known popular hyper textual crawling processes HITS [39] and PageRank [47] (Foundation of the Google Company; Figure 2). These two classic hyper textual algorithms are both designed to crawl and index the internet for search engine web-search activities. Hyper textual links are extracted from page-sources and article references are counted for ranking purposes.

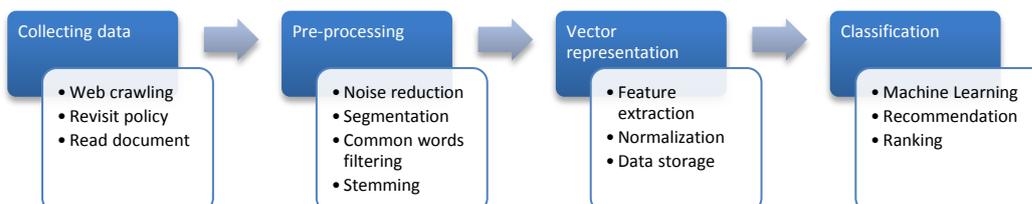


Figure 1: Data mining cycle

The core techniques from these HITS and PageRank algorithms still form the basis for many modern web spiders.

A web crawler starts searching by inspecting sources from a Queue of known web-addresses. This queue will continuously grow by adding new addresses extracted from the inspected page sources. Because of the dynamic structure of the web, the content of the visited sources changes over time. Therefore a *revisit policy* is used to rescan the sources in the queue. The most used policies are based on data accuracy (freshness) and age (latest update) [18]. Finally, the data extracted by crawlers is stored in large databases for further processing.

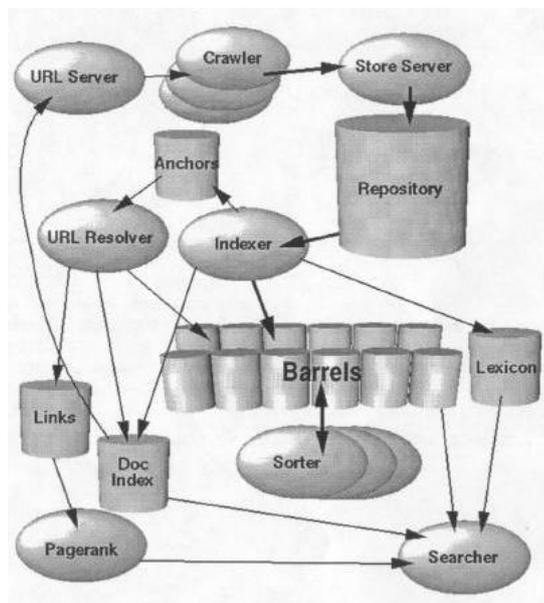


Figure 2: PageRank architecture, from the original paper [47]. Foundation of the Google Company.

### 2.2.2 Pre-processing

Once the raw data is collected during the crawling process, it needs to be preprocessed before the data can be used for further analysis. To structure the raw data and remove clutter, the following pre-processing steps are generally executed: *noise reduction*, *segmentation*, *common words filtering* and *stemming*.

First the data is cleared from unwanted information during the *noise reduction* process. In the case of web crawling for text classification, html tags and other obscured unnecessary information is therefore removed from the source. Thereafter, during the *segmentation* phase, the noise filtered data is split into usable chunks or segments of data. For text processing this generally means the data is split into sets of words, sentences or phrases. In the next step each segment is validated against a list of common words. It is known at forehand commonly used words (i.e. “the”, “and”, etc.) don’t contribute to the actual meaning and classification of the content; therefore these words are filtered out. Because the final goal is to create a unique profile for each item, these common words can be omitted without losing information. The next and final step during pre-processing for text classification is stemming.

Stemming is the process of reducing the words to their stem, root or base form. A stemming algorithm reduces the words “walking”, “walked”, “walk” and “walker” all to the word “walk”. By using word stems, the system is able to classify five different articles, each containing at least one of the above words, into the same category and not into five separate categories. Another advantage of using the stem form is the reduction of the length of the overall word lexicon. This furthermore results in a reduction of the size of the feature vectors (paragraph 2.2.3) and therefore increases the speed of the recommendation by reducing the dimensions of the data space used by the classification algorithms. A disadvantage of using stemming algorithms is the loss of word meaning. There are multiple implementations for stemming algorithms, ranging from using lookup tables to n-gram algorithms. Stemming is language specific, related to syntax of the languages’ grammar. The de-facto standard and widely used stemmer for English is written by Martin Porter [49].

### 2.2.3 Feature extraction

To be able to perform calculations on the data, a textual representation is of no direct use. A text needs to be represented as a list of numerical values called a *feature vector* to be usable for the recommendation algorithms. These features can be anything related to the text, ranging from publish date, text length, category index or community rating to individual word counts or other advanced structural values from language processing techniques. The goal of a feature vector is to create a unique representation of the article displaying (a measurement of) its contents.

Since different terms have different importance in texts, an importance indicator, the *term weight*, can be associated with each term instead of a regular word count. Three main components that affect the importance of a term in a text are the *term frequency (tf)*, the *inverse document frequency (idf)* and the document length *normalization*.

The first indicator (*tf*) measures the occurrences of a term  $t_i$  within a document  $d_j$ , defined as follows.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Formula 1: Term frequency (*tf*)

Where  $n_{i,j}$  indicates the number of occurrences of the term  $t_i$  in the document  $d_j$ .

The second measurement (*idf*) is a measure of general importance of the term in relation to the availability in the corpus (all documents). The *inverse document frequency* is calculated by taking the logarithm of the total number of documents  $D$  in the corpus divided by the number of documents  $d$  containing the term  $t_i$ , given by the following formula.

$$idf_i = \log \frac{D}{|\{d: t_i \in d\}|}$$

Formula 2: Inverse document frequency (*idf*)

Where  $|\{d: t_i \in d\}|$  indicates the number of documents the term  $t_i$  appears, division-by-zero must be prevented.

These first two factors (*tf* & *idf*) can be used to calculate the so called *tf-idf* measurement for each term. This commonly used measurement indicates the importance of a word within a document, where the importance increases proportionally to the number of occurrences in the document, but is offset by the frequency of the word in the corpus. This is defined as

$$(tf-idf)_{i,j} = tf_{i,j} \times idf_{i,j}$$

This *tf-idf* measurement can be used as *term weight* in the final *feature vector*. Finally, before the feature vector is stored in the database and ready for usage in the classification and recommendation processes, the values should be *normalized* to account for variations in document length. Normalization is explained in the next paragraph (2.2.4).

#### 2.2.4 Normalization

Document length normalization of term weights is used to remove the advantage that long documents have in retrieval over short documents. The main reasons to normalize the term weights are:

- 1) **Higher term frequencies:** Long documents usually use the same terms repeatedly. As a result, the term frequency factors may be large for long documents, increasing the average contribution of its terms towards the term frequency counts within the corpus.
- 2) **More terms:** Longer documents logically also have more distinct terms. This increases the influence and chances of retrieval for longer documents over shorter texts during the recommendation process.

Document length normalization is therefore used to penalize the term weights for a document in accordance with its length. The most commonly used normalization technique is the *Cosine Normalization* (CN). The Cosine Normal factor is computed as

$$\sqrt{w_1^2 + w_2^2 + \dots + w_t^2}$$

*Formula 3: Cosine normalization*

Where  $w_i$  is the calculated *tf-idf* term weight (paragraph 2.2.3).

Each  $w_i$  is finally divided by this normalization factor. This way the *Cosine Normalization* attacks both normalization reasons above (higher *tf* and more terms) in one step. Higher individual term frequencies increase individual  $w_i$  values, thereby increasing the penalty on the term weights. Also, if a document has more terms, a higher normalization factor is returned. More advanced research on normalization is given by [58].

This final normalized feature vector will be the stored article content representation within the item-profile in the database and can be accessed by the classification algorithms to generate recommendations.

### 2.3 Text classification

The goal of text classification is to automatically categorize a set of articles into two or more categories. Targeting netnews recommendation, this can be translated to labeling articles into at least the categories 'like' or 'dislike' to indicate a user's interest for the articles content. More categories could be thought of, for example a category 'known' to filter out articles of subjects already known, but this research will concentrate on the binary classification task. Besides a label, a likelihood parameter on how likely the article belongs to its category should be calculated. This likelihood parameter or 'degree of interest' is used for ranking and therewith presents the most interesting article to the user first (i.e. display on top of the page). The next paragraphs review a number of machine learning techniques [44] commonly used for this text classification task.

### 2.3.1 Machine learning

Because they try to learn a function that models each user's interests, classification algorithms based on machine learning techniques are the key component of content-based recommendation systems. Given a new item and a user model, the function predicts whether the user would be interested in the item. This is a *supervised* learning task. In the case of netnews personalization, the learning task is to create a model to categorize new articles from the web based on the users' feedback on already displayed articles in the past. This feedback can be both explicit (i.e. votes) or implicit (i.e. system interactions) and is read from the user profile (paragraph 2.1.2). An overview of machine learning techniques used for automated text categorization is given by [55].

The main difficulties with machine learning for text classification are *high dimensionality* and *noise*. Because of the huge set of possible terms in the corpus, the chances of *over fitting* increase. Especially when a classifier is trained with a small data set the chances of distributing articles over unique dimensions are higher (no referencing terms). The use of normalization techniques (paragraph 2.2.4) partly tackles this problem. Furthermore, text is unstructured and noisy data. The input noise is reduced by pre-processing the data (paragraph 2.2.2) and by the use of feature vectors with tf-idf values (paragraph 2.2.3). But especially output noise can reduce classification performance. When a classifier is trained with data pointing to the wrong classes, such a classifier can never predict accurately.

For personal recommendation these can be difficult problems to tackle. Because user feedback is scarce and hard to gather, the training sets are often small. Also, when using implicit feedback (system interactions) the output noise can become a large negative factor.

### 2.3.2 Cosine similarity

Although generally not documented as 'machine learning algorithm' on its own, the *cosine similarity* is a widely used measurement to compare and classify documents for content based recommendation. This measurement searches for the cosine of the angle between two vectors. The cosine similarity is defined as follows:

$$\cos(\alpha) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

*Formula 4: Cosine similarity*

Where A and B are the document *feature vectors* containing *tf-idf* values (paragraph 2.2.3).

The results of this cosine similarity range from 0 to 1 (*tf-idf* values cannot be negative) and can be interpreted as an inverted distance measurement between two documents. Therefore the cosine similarity can be used as a likelihood estimate for an articles' category by combining similarities of all documents in a category. Categorization itself is based on threshold values. For example values < .5 = "dislike", > .9 = "known" and articles in-between are labeled "like".

### 2.3.3 Naïve Bayes

Because of simplicity and effectiveness, also *Naïve Bayes* classifiers are often used in text classification applications and experiments. However its performance is often degraded because it does not model text well. These classifiers are mostly implemented for the long-term memory models.

*Naïve Bayes* models are a probabilistic approach to text classification. These models are based on the Bayesian theorem, given by

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

*Formula 5: Bayes theorem: Given a hypothesis (h) and data (D), calculate the chance of h given D.*

Generally the most probable hypothesis  $h \in H$ , called the *maximum a posteriori hypothesis* ( $h_{MAP}$ ), is used for classification and is given by

$$h_{MAP} = \arg \max_{h \in H} P(h|D) = \arg \max_{h \in H} P(D|h)P(h)$$

*Formula 6: Maximum a posteriori hypothesis*

Where  $P(D)$  is dropped because it is a constant independent of  $h$ .

By interpreting an article as a *bag of unrelated words* instead of a structured article, the ‘naïve’ assumption applies. This assumption can be used to define the Naïve Bayes classifier for inductive learning. This is defined as modeling a concept function  $f: X \rightarrow C$ , where  $c \in C$  represents a class label and  $x \in X$  models a feature described by a *feature vector*  $\langle x_i, \dots, x_n \rangle$ :

$$C_{MAP} = \arg \max_{c \in C} P(c|x_i, \dots, x_n) = \arg \max_{c \in C} P(x_i, \dots, x_n|c)P(c)$$

*Formula 7: Bayesian learning*

Finally, by using the Naïve Bayes assumption of independent features, the Naïve Bayes classifier is derived:

$$C_{NB} = \arg \max_{c \in C} P(c) \prod_n P(x_n|c)$$

*Formula 8: Naive Bayes classifier*

Therefore to classify a new article  $a$  with a feature vector  $\langle x_i, \dots, x_n \rangle$  by using a Naïve Bayes classifier trained on a dataset  $D$  with known feedback  $c$ :

- **Training:** Calculate the individual term probabilities  $P(x_n|c)$  from each  $d \in D$  (articles) with known classification  $c$  (feedback).
- **Testing:** Score each  $c$  using  $C_{NB}$  (Formula 8) on  $\langle x_i, \dots, x_n \rangle$  in article  $a$ . Return  $c$  with the highest probability.

### 2.3.4 k-Nearest Neighbors

Another widely used machine learning algorithm is *k-Nearest Neighbors* (k-NN). The nearest neighbor algorithm simply stores all its training data in memory. Therefore it's also called a *lazy algorithm* belonging to the category of *instance based* learning algorithms. For text classification, each article feature vector  $\langle x_i, \dots, x_n \rangle$  with a known class (feedback) is therefore stored as a data point in a multi-dimensional data space  $\mathbb{R}^n$ , where  $n$  is the variety of terms in the corpus. Because of the high demand on memory, this algorithm is mostly used for smaller datasets and therefore in the case of personal recommendation it is mainly implemented for short-term user models.

In order to classify a new unlabeled item, the algorithm compares it to all stored items using a similarity function and determines the “nearest neighbor” or  $k$  nearest neighbors. The default similarity measurement is the *Euclidian distance*. This distance  $d$  measurement between the current, unseen, article  $a_c$  and all other articles  $a_i$  in  $\mathbb{R}^n$ , is calculated by

$$d(a_c, a_i) = \sqrt{\sum_{r=1}^N (x_{r,i} - x_{r,c})^2}$$

Formula 9: Euclidian distance

Where  $x_{r,i}$  represents a feature (term value)  $r$  from article  $i$ .

Another distance measurement often used within k-NN, especially for text classification, is the *cosine distance* as explained in a previous paragraph (2.3.2). Finally, the class label and/or numeric score for a previously unseen item can be derived from the class labels of the calculated  $k$  nearest neighbors. Thereby, the most widely used selection method is majority voting.

### 2.3.5 Support Vector Machines

*Support Vector Machines* (SVM), introduced by V. Vapnik et al [61], are becoming increasingly popular as machine learning algorithm for text classification [36]. The main reasons of the success of SVMs in this field are: (1) SVMs are *universal learners*, (2) *independence of dimensionality* of feature space and (3) heuristics exist for *automatic parameter selection* [35].

First SVMs are universal learners. In their basic form, SVMs learn a linear threshold function. Nevertheless, by a simple “plug-in” of an appropriate kernel function, they can be used to learn polynomial classifiers, radial basis function (RBF) networks and three-layer sigmoid neural nets.

The second property of SVMs is the ability to learn independent of the dimensionality of the feature space. SVMs measure the complexity of hypotheses based on the margin with which they separate the data, not the number of features. This means that, unlike  $k$ -NN, SVMs can generalize even in the presence of very many features, if the data is separable with a wide margin using functions from the hypothesis space.

The same margin argument also suggests a heuristic for selecting good parameter settings for the learner. Usually a grid search combined with cross-validation is used to optimize the classification parameters.

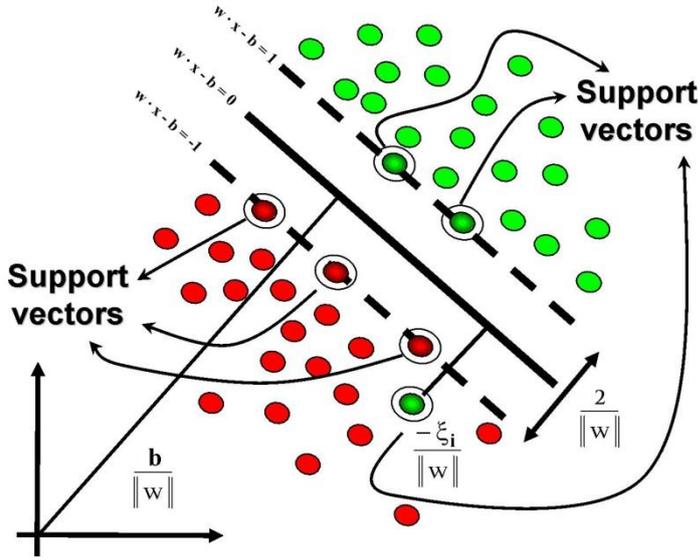


Figure 3: Support Vector Machine; 2-dimensional example

A *Support Vector Machine* performs classification by constructing an  $N$ -dimensional hyperplane that optimally separates the data into categories. To illustrate this, an idealized example in a 2-dimensional data space  $\mathbb{R}^2$  is given in Figure 3. Each dot in the picture represents a feature vector  $\mathbf{x}_i$ . Class labels  $y_i \in \{-1, 1\}$  are indicated by color. The basic idea is to find a hyperplane optimally separating the data by maximizing the margin between *support vectors*. A hyperplane can be described as a set of points  $\mathbf{x}$  satisfying  $\mathbf{w} \cdot \mathbf{x} - b = 0$ , where  $\mathbf{w}$  denotes the normal vector perpendicular to the hyperplane. The distance of the hyperplane to the origin is given by the value  $\frac{b}{\|\mathbf{w}\|}$ . The goal is to maximize the distance  $\frac{2}{\|\mathbf{w}\|}$  between the *canonical hyperplanes* connecting the *support vectors* (dotted lines in Figure 3), these hyperplanes are described by the equations  $\mathbf{w} \cdot \mathbf{x} - b = 1$  and  $\mathbf{w} \cdot \mathbf{x} - b = -1$ . Therefore the final goal becomes to minimize  $\|\mathbf{w}\|$ , described as

$$\Phi(x) = \min_{\mathbf{w}, b} \|\mathbf{w}\| = \min_{\mathbf{w}, b} \frac{1}{2} (\mathbf{w} \cdot \mathbf{w}) \quad \text{subject to } y_i (\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \text{ for } i = 1, \dots, n$$

Formula 10: SVM optimization problem

This equation is subject to constraints to prevent data points within the margin.

In Figure 3, one (green) data point is deflected from the group and is situated on the opposite of the separating hyperplane and will therefore be misclassified. To allow such mislabeled data a *soft margin* is introduced to split the examples as clearly as possible. The optimization problem from Formula 10 is therefore extended with slack variables  $\xi_i$  indicating the error of the margin as follows:

$$\Phi(x) = \min_{\mathbf{w}, b} \left\{ \frac{1}{2} (\mathbf{w} \cdot \mathbf{w}) + C \sum_{i=1}^n \xi_i \right\} \quad \text{subject to } y_i (\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i, \text{ for } i = 1, \dots, n$$

Formula 11: SVM optimization problem with soft margin

To solve this optimization problem, it can be rewritten to a Lagrangian formulation:

$$L_P \equiv \min_{\mathbf{w}, b} \max_{\alpha} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i - b) - 1 + \xi_i] \right\}$$

Formula 12: Lagrangian formulation

Where  $\|\mathbf{w}\|$  is replaced by  $\frac{1}{2} \|\mathbf{w}\|^2$  and positive Lagrange multipliers  $\alpha_i, i = 1, \dots, l$ , are introduced.

The solution can now be calculated using *quadratic programming* techniques (QP). QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints. The subject of *quadratic programming* lies outside the scope of this research and therefore the exact formulation of the solution [12] is omitted. The result should be a vector  $\mathbf{w}$  with a linear combination of relatively small percentage of points  $x_i$  (the support vectors) expressed by:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \quad \text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \text{ for } i = 1, \dots, n$$

Formula 13: Solution to optimization problem

The problem of classifying a new data point  $\mathbf{x}$  is now simply solved by looking at the sign of:  $\text{sign}(\mathbf{w} \cdot \mathbf{x} - b)$ .

Furthermore, the above simplified linear example in 2d space can be extended to higher (possibly infinite) feature spaces  $\mathbb{R}^n$ :  $\Phi(x) = \{\Phi_1(x), \Phi_2(x), \dots, \Phi_n(x)\}$  to solve non-linear classification problems by preprocessing the data  $x \rightarrow \Phi(x)$  using a *kernel function*  $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$ . Examples of such kernels are  $K(x, y) = (x \cdot y + 1)^P$  (Polynomial) and  $K(x, y) = e^{-\beta|x-y|^2}$  (Gaussian RBF). This process is illustrated in Figure 4.

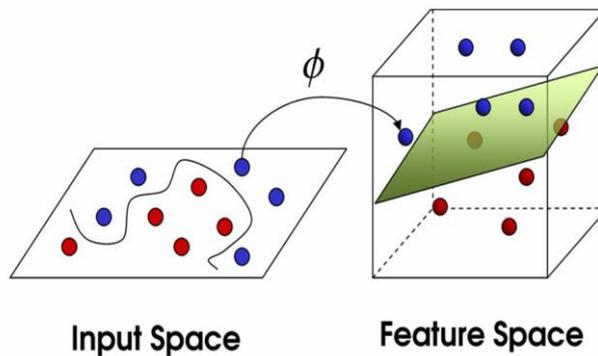


Figure 4: SVM; Mapping to higher dimension(s) by using a kernel function

## 2.4 Agent models

Much less attention has been gained by recommender systems based on multi-agent architectures [45]. An overview of this field of research is given by [62]. In the next two paragraphs a short introduction to agent based models and some examples of research on agent based recommender systems are given.

### 2.4.1 Multi-agent systems

Multi-agent systems (MAS) are systems composed of multiple interacting software agents. These systems can be used to solve problems of higher complexity which are difficult or impossible to solve for an individual agent. A general definition of an agent [62] is as follows:

An *agent* is a computer system that is *situated* in some *environment*, and is capable of *autonomous action* in this environment in order to meet its design objectives.

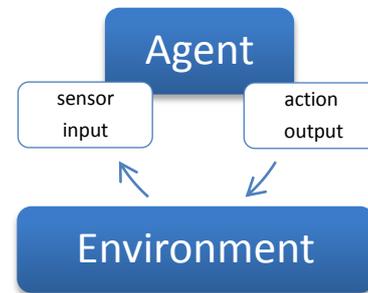


Figure 5: Abstract agent model

An abstract view of this agent model from the above definition is illustrated in Figure 5. An important aspect of this model is the decoupling of the environment and the agent. Generally the environment is assumed to be *non-deterministic* and *non-predictable*. Agents can interact with the environment and influence it, but they do not have full control over it.

In a multi-agent architecture agents often share knowledge by communication, using a language available from a pre-specified communication protocol. Two important characteristics of many multi-agent systems are *decentralization* and *self-organization*. Decentralization is obtained by the absence of a central designated control agent. All agents act on their own based on their internal properties and the environmental influences. Mostly, the result is *self-organization*, whereby a (non-planned) pattern of agent behavior emerges from the system interactions.

### 2.4.2 Agent based recommender systems

An early and experimental agent based recommender system METIOREW is given by [20]. In this work a framework is proposed which combines a set of agents with specific goals and information sharing capabilities. It's a promising setup, but no experiments are conducted, so no results can be presented from this work. One of the first (positive) results using a collection of information filtering agents is found in [28]. These filterbots are capable of reducing noise in collaborative filtering results using content information. The conclusion in this research is that better results are obtained by using multiple simple agents instead of using a single complex agent. Another research [10] also shows an increase in performance of an agent based recommendation system. In this research a system supporting communities of people searching the web is proposed. Agents share their knowledge about users' behavior gained from data mining techniques. More recent work of [2] describes an agent based approach where personalized content models and missing data models are combined to produce item-based predictions of user ratings. These scores are combined in a stacked agent model, where the agents are trained using an SVM (paragraph 2.3.5) to produce recommendations. This research shows positive results compared to single content-based models.

## 2.5 Related work

Some classic influential papers for the field of recommender systems are: [52] One of the first collaborative filters for netnews articles; [8] Content-based and collaborative filtering using agents and [37] A tour guide software agent for the web capable of suggesting hyperlinks. Main research (groups) are GROUPLens research group [31] and work by Pazzani et al. The newest techniques are presented yearly at the international conference for the subject: RecSys [1]. Public commercial systems using recommender techniques are i.e. [22] a system filtering, scanning and collating stories from the web, based on user interests and [50] a party providing a search engine for news articles (Postrank; technique based on Google's pagerank for web pages). Furthermore this last service also includes a collaborative filtering technique.

## 3 Methods

The theory above is combined and tested by implementing a personalized adaptive netnews recommender system: *iNewsReader*. A conceptual multi-agent driven framework will be described in the next paragraphs. This framework is largely implemented, some parts are left out or ‘short wired’ to reduce workload and enable more precise measurements of the individual components. Finally, an experiment is conducted, whereby the system is tested and analyzed on real-world live data. This experiment is described in the next chapter.

### 3.1 Information sources

The recommender system as an intelligent filter for netnews articles uses *Really Simple Syndication* (RSS) feeds as primary data source. RSS feeds are dynamic lists of web content in a pre-specified XML format (Figure 6). These feeds are generally accepted and implemented all around the web to publish frequently updated information. This way a huge amount of structured articles is available for automatic processing.

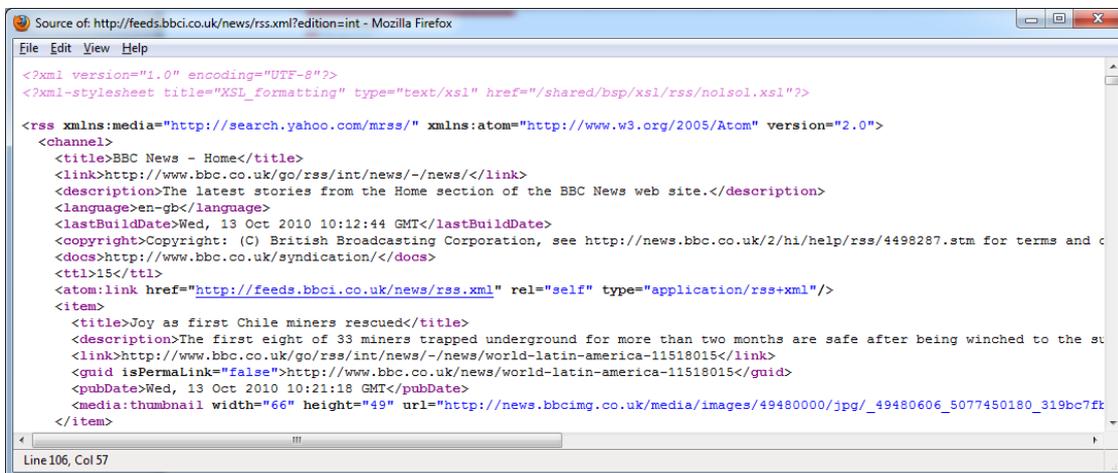


Figure 6: RSS source

By using RSS as a main data source, instead of plain HTML-documents, the clutter of html markup (not contributing to the meaning of an article) is partly omitted. Furthermore semantic information about the articles (i.e. author, title, tags, category, etc.) is directly available from within the RSS feed.

The initial implementation of *iNewsReader* was solely based on the contents of these RSS feeds, omitting the source of the full article referred to. During processing it turned out this information is often sparse and incomplete. Therefore the implemented crawler is extended by also inspecting the referred full article source. Information from the RSS feed is thereby used to extract the article body from the page and remove all other information (section 3.5).

## 3.2 General system setup

The concept of the *iNewsReader* recommender system is built around four main components (Figure 7): *data storage*, *web crawler*, *recommender(s)* and a *web portal / user interface* (For reference, see also appendix 7.1 for the original and initial system proposals). All system actions, communication and data transfers between these main components are executed by small software agents. So no direct communication takes place between the components. On the contrary, the agents themselves should be able to communicate and interact with each other. A central organ manages these agents (AMS) and provides the communication and transport layers for this agent interaction.

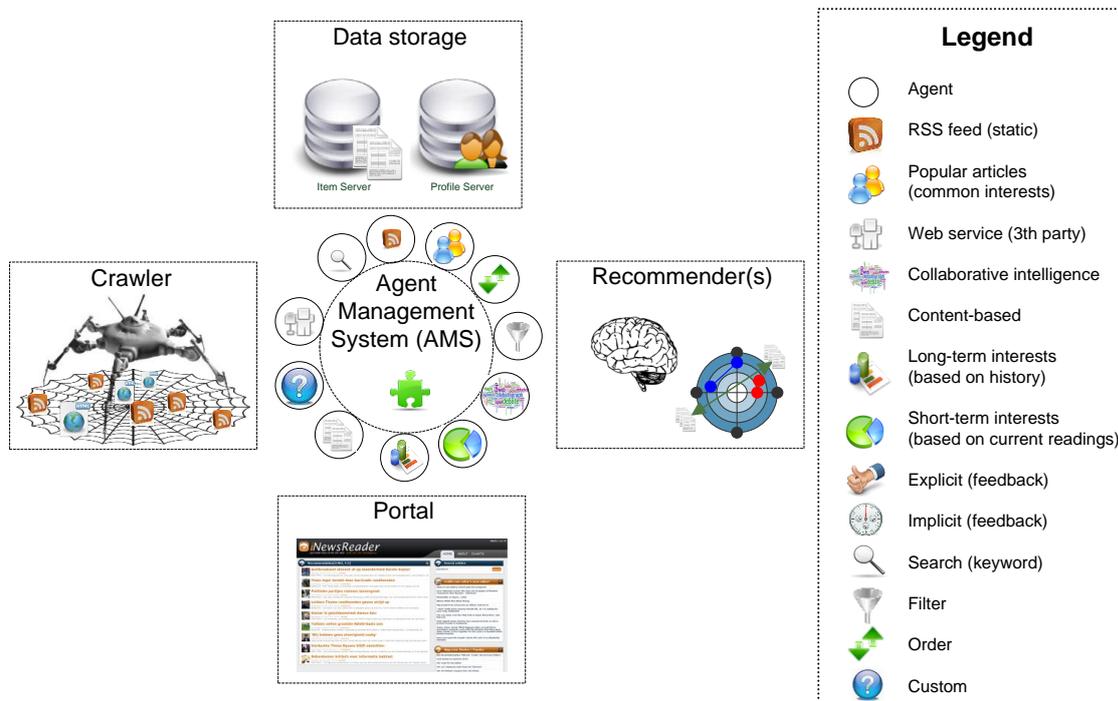


Figure 7: General system setup

The process starts with an agent asking for the next RSS feed to harvest; this feed is read from a queue stored at the item server in the data storage. The feed is presented to the crawler, which collects the data and returns item-profiles for each article in the feed (section 2.2). These item-profiles contain article information including pre-processed feature vectors to be stored into the item server.

When a user logs in to the system via the portal (web) interface, multiple user centered agents are activated to retrieve personal information for the user. Some agents directly return information by providing a list of articles from online RSS feeds (i.e. personal static feed agents or 3th party agents). Other agents contact the item server to retrieve the information (i.e. search or popularity agents). The more advanced agents use the recommender components to estimate the rate of user interest for a set of articles. These recommender components on their behalf use agents to retrieve user information and history from the profile server to train the classification algorithms (section 2.3). Finally intermediate agents can filter or sort the retrieved list of articles before it's presented to the user. Of course multiple alternative routes and non-described agents could contribute to the final personalized recommendation.

### 3.3 Framework implementation

As a proof of concept a large part of the above system is implemented. In the first place, all four main components (data storage, crawler, recommenders and portal) are built. No personalized netnews recommendation is possible if one of these components is missing. To reduce implementation costs (time), only a basic version of the agent management system (AMS) is constructed. By ‘short wiring’ the communication with the data storage, the functionality of the AMS can be reduced to only handle the recommendation agents and provide the primal communication between the portal and the recommender components. This way the core experiment of testing multiple recommender agent configurations isn’t influenced and all information displayed to the user is still gathered by calling the responsible agents. The final reduced implementation of the framework is illustrated in Figure 8. The main disadvantage of this approach is the reduced flexibility in distributed processing. This ‘short wiring’ requires multiple processes to be executed on the same machine; therefore this approach causes a decreased performance of the testing environment. For a detailed discussion about additional shortcomings, see also section 3.9 and paragraph 5.2.1 .

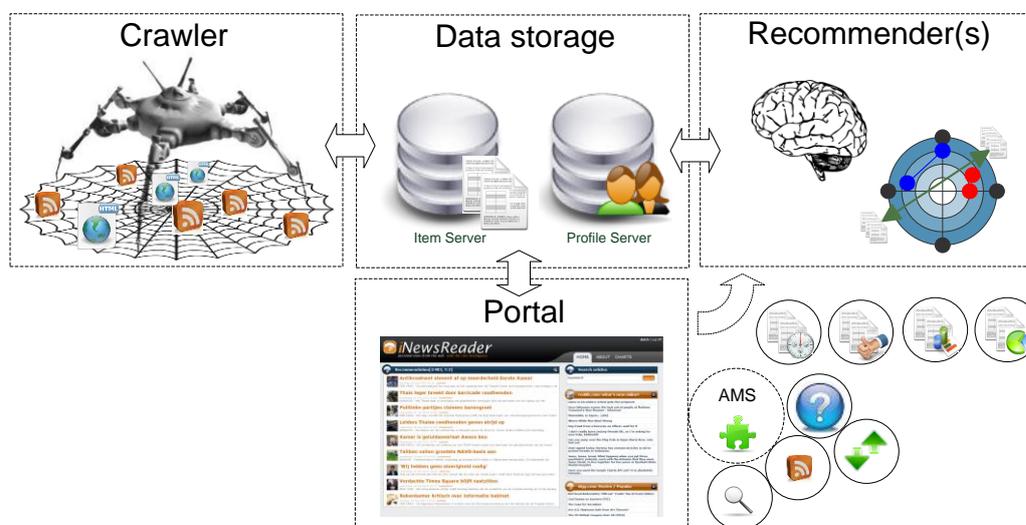


Figure 8: Implemented recommender framework (legend of Figure 7 applies)

Finally, a choice needed to be made what kind of recommenders, and related agents, to implement (theoretically, an infinite amount of agents can be attached to the system). Because it’s hard to generate a large reliable user base for a project this size, this research is more directed toward content based recommendation. Furthermore, lately a lot of research has been conducted towards collaborative algorithms and these techniques are already widely implemented in production systems. Therefore two content based recommender algorithms are implemented providing *Naïve Bayes* and *Support Vector Machine* classification. These algorithms are interfaced with eight unique configurations of recommender agents, based on recommender type, feedback (implicit/explicit) and memory model (short- / long-term).

### 3.4 Data storage

The data storage is split into two database servers, an item storage and a (user) profile storage. Both databases are relational storages hosted by MS SQL Server 2008 instances. The item server (Figure 9) is responsible for storage of all item related information. The *n-dimensional data warehouse capability* for storage of article information and related term feature vectors is implemented by using a cross table ('ArticleTerm' table in Figure 9), linking article information ('Article' table) with term feature values ('Term' table).

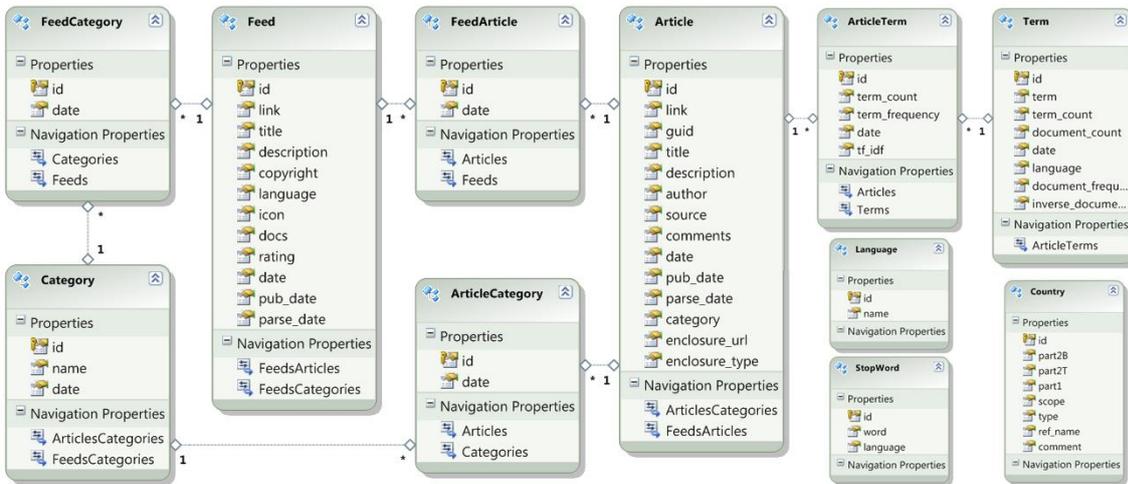


Figure 9: Item-server

The profile-server (Figure 10) contains all user information and their preferences. Information about user system interaction, user feedback and other user related information are also stored here. Furthermore, offline pre-calculated personal recommendations are stored at the profile server.

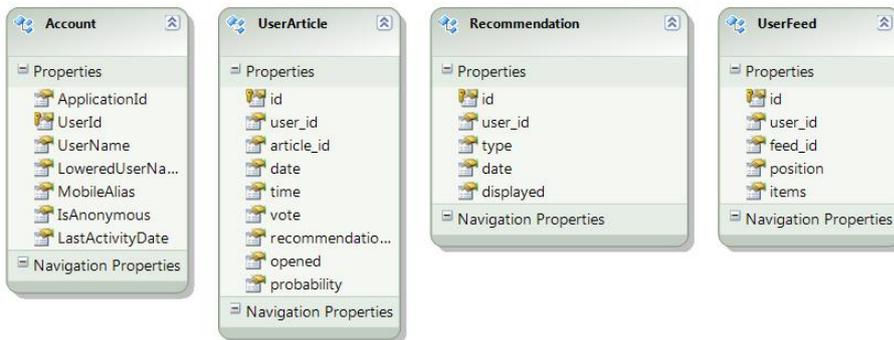


Figure 10: Profile-server

The data access layers providing the interfaces to both database servers are implemented using the ADO.NET Entity framework. This entity framework creates a data-oriented object to SQL Query translation. The Language-integrated query (LINQ) technology is thereby used to query the databases and return pre-defined data objects. This way the application is free from data (model) dependencies and data can be accessed using an object-oriented approach supporting compile-time syntax validation for queries against a conceptual model.

## 3.5 Harvesting

The crawler module for harvesting information from the internet is implemented using the techniques described in section 2.2. The crawler is responsible for the first three phases, *collecting* data, *pre-processing* the data and *feature extraction*, of the *data mining cycle* (Figure 1). This entire harvesting procedure is executed offline in a separate process, so it has no influence on the speed of the recommendation process itself and is therefore unnoticeable for the user of the system.

For reasons described in section 3.1, the crawler inspects both the provided RSS feeds as the html source of the articles referred to. But before the data is collected from the internet, first a few basic checks are executed. The feed and article urls are checked on format and parse dates. Already parsed articles and also black listed urls are skipped from processing. After these checks, the available article information from the RSS feed is stored into the item-server.

The next stage is the processing of the html page source of each article referred to from the RSS feed. Webpages contain lots of uninteresting information (i.e. headers, menu, footer, advertisements, etc.). The tree structured html source (Document Object Model) of each page is traversed to filter out the information referred to by the RSS feed. The general idea is illustrated in Figure 11 below.

```
<html>
  <head>
    <!-- document info -->
  </head>
  <body>
    <div class='header'>...</div>
    <div class='menu'>...</div>
    ...
    <div class='article'>
      <h2>Article title</h2>
      <span class='summary'>
        <span>publish date: 01-01-2010</span>
        <p>Article Description</p>
      </span>
      <span class='content'>
        <p>Article body</p>
        <img src='illustration.jpg' />
      </span>
    </div>
    ...
    <div class='advertisements'>...</div>
    <div class='footer'>...</div>
  </body>
</html>
```

Figure 11: HTML Document structure (DOM tree)

In the figure, the code block between the red lines (`<div class='article'> ... </div>`) contains the actual information referred to from the RSS feed, the additional code above and below clutters this information and needs to be filtered out. To find this block of code, first the crawler tries to match the title and description provided by the RSS feed with the information at the html page and registers their DOM nodes (`<h2>Article title</h2>` and `<p>Article`

Description

in the example source). Next the first parent node, traveling backward on the DOM tree using *XPath*, enclosing both title and description nodes (`<div class='article'>`) is registered as final 'article' node and returned for further processing.

Of course the example above is over simplified and real world data is much less structured, but the process still applies. A random example of a parsed TechCrunch [59] news article is given in Figure 12. The matched DOM nodes are bordered (CSS) using the same colors as in Figure 11.

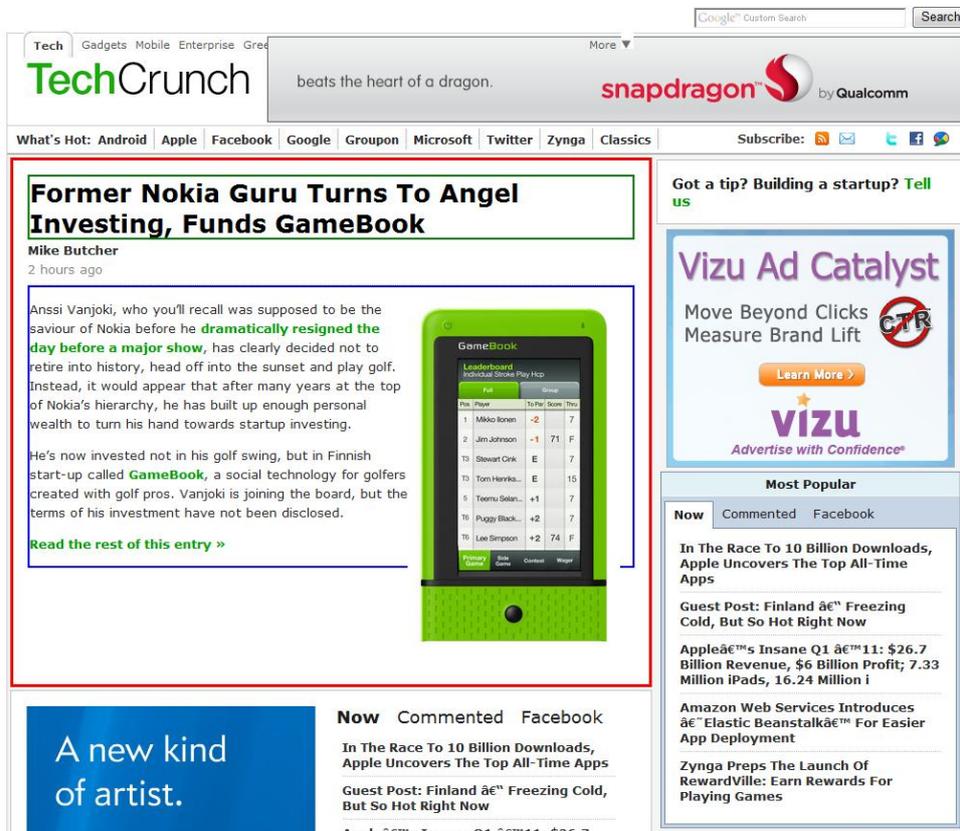


Figure 12: Crawler output

Two algorithms are implemented for matching strings of text. These algorithms are used for matching the title and description from the RSS feed with the text at the page source. First, a faster but less reliable algorithm calculates the percentage of words present in both strings by counting occurrences. If no reliable match is found (using a threshold constant  $T = .9$ , or  $< 90\%$ ), a second algorithm takes over, calculating the Levenshtein distance. The Levenshtein distance is obtained by calculating the cheapest way to transform one string into another by using insertions, deletions and/or substitutions, the process is illustrated in the example below (Figure 13).

		m	e	i	l	e	n	s	t	e	i	n
l	0	1	2	3	4	5	6	7	8	9	10	11
e	1	1	2	3	3	4	5	6	7	8	9	10
v	2	2	1	2	3	3	4	5	6	7	8	9
e	3	3	2	2	3	4	4	5	6	7	8	9
n	4	4	3	3	3	3	4	5	6	6	7	8
s	5	5	4	4	4	4	3	4	5	6	7	7
h	6	6	5	5	5	5	4	3	4	5	6	7
t	7	7	6	6	6	6	5	4	4	5	6	7
e	8	8	7	7	7	7	6	5	4	5	6	7
i	9	9	8	8	8	7	7	6	5	4	5	6
n	10	10	9	8	9	8	8	7	6	5	4	5
	11	11	10	9	9	9	8	8	7	6	5	4

Two shortest distances ('=' math; 'o' substitution; '+' insertion and '-' deletion):

l	e	v	e	n	s	h	t	e	i	n	or	l	e	v	e	n	s	h	t	e	i	n
o	=	+	o	=	=	=	=	=	=	=		o	=	o	+	=	=	=	=	=	=	=
m	e	i	l	e	n	s	t	e	i	n		m	e	i	l	e	n	s	t	e	i	n

Figure 13: Levenshtein distance

An implementation of the algorithm is given by

```

int LevenshteinDistance(char s[1..m], char t[1..n])
{
    // d is a table with m+1 rows and n+1 columns
    declare int d[0..m, 0..n]

    for i from 0 to m
        d[i, 0] := i // deletion
    for j from 0 to n
        d[0, j] := j // insertion

    for j from 1 to n
    {
        for i from 1 to m
        {
            if s[i] = t[j] then
                d[i, j] := d[i-1, j-1]
            else
                d[i, j] := minimum
                    (
                        d[i-1, j] + 1, // deletion
                        d[i, j-1] + 1, // insertion
                        d[i-1, j-1] + 1 // substitution
                    )
        }
    }

    return d[m,n]
}

```

Figure 14: Levenshtein algorithm

Alternatively, if neither of these algorithms finds a reliable match for the title or description within the article source, the 'article DOM node' can't be located and the full page body is returned.

Next, the returned page source is stripped from its html elements, so only the inner text remains. The result is combined with the RSS data and further pre-processed as explained in section 2.2, continuing from 2.2.2. At the end, the page source is also inspected for references to new RSS feeds. These feeds are added to the feeds queue at the item-server to be harvested in future processing.

## 3.6 Portal

The main user interface (UI) and access point for the recommendation services is a public web portal. Users are able to subscribe and set preferences for their system usage. Prior to this registration the portal displays a random set of articles. After registration the interface shows articles of personal interest to the user. During reading of these articles, users are able to provide feedback for each article (voting). This explicit user feedback and additional implicit interest indicators are stored in the profile-server at the data storage. This feedback is thereafter accessed by the system to refine the list of interesting articles over time for future recommendations.

### 3.6.1 Web server

A Microsoft Internet Information Services v7.5 (IIS 7.5) web server is configured to run the public web portal. The portal itself is written in the ASP.NET (C#) web language and structured using the Model View Controller (MVC 2.0) pattern, provided by Microsoft. To prevent page refreshes Asynchronous JavaScript and XML (AJAX) Services are used to call special purpose web services. Examples of these services are the voting and search services. The server side framework for these web services is based on the Windows Communication Foundation (WCF), a service oriented architecture supporting distributed computing.

### 3.6.2 User accounts

To be able to identify users and provide personal recommendation, users need to register themselves first. The registration process is kept as simple as possible and only asks for a username, password and e-mail address. During registration a user account with a unique identifier (User ID) is created and stored at the profile server. This unique identifier is used for all further references and database couplings to the user profile. For authentication, the built-in authentication services from the MVC 2.0 framework are used. On every next visit the user can directly login using the created credentials.

### 3.6.3 User interface

It is known the user interface (UI) is of great influence for the successfulness of the overall system. A bad interface won't attract users, which are on their turn necessary for collaborative techniques. Also, a less attractive interface influences the reading 'pleasure', this probably results in a negative effect on system usage. There is also a huge diversity in web browsers and screen resolutions to account for accessibility. Again, to reduce the overall size of the project, these collaborate techniques were already omitted in the first place and also the UI gains less attention as it deserves. The focus in this project lies on the techniques 'under the hood'. So the usability, attractiveness and accessibility of the system are beforehand left open for future research.

The first impression of the interface, the home-page, is shown in Figure 15. At the top of the page a header includes the logo, menu and authentication components. The logo identifies the portal and can be clicked to return to the home page at all times. For navigation the menu provides the buttons to access all functionality on the site. This menu is extended at user login with additional buttons to restricted pages. Users can login and register using the login form on the top right of the page.

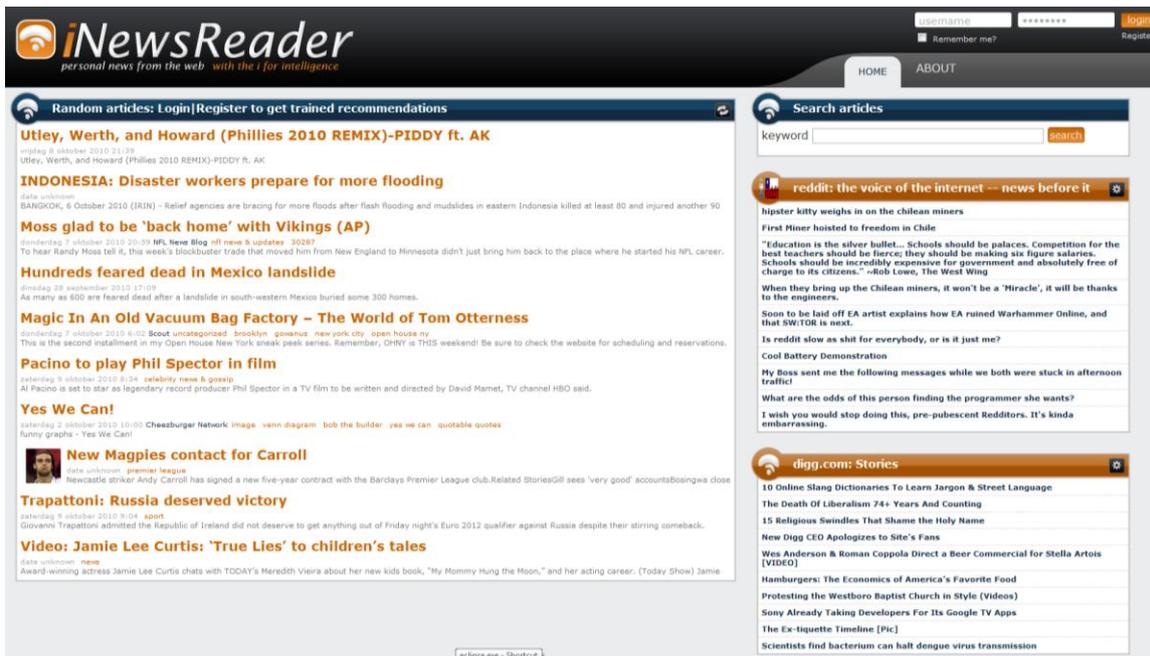


Figure 15: User Interface (Portal - Home)

Underneath the header the actual content is presented. On the left side, the main recommendation view is displayed. In the screenshot above a set of random articles is shown, after authentication, this list will be replaced by a set of articles from the recommendation agents. A refresh button can be used to call for a new set of articles. The right side displays a search box and two random static personal RSS feeds. The search box can be used to call a 'search agent' to query the item-server for articles on a specific subject. The static feeds can be configured (if logged in) using the configuration icon. Each user can add a (theoretically) unlimited amount of their personal favorite RSS feeds to occupy the home page (Figure 16).

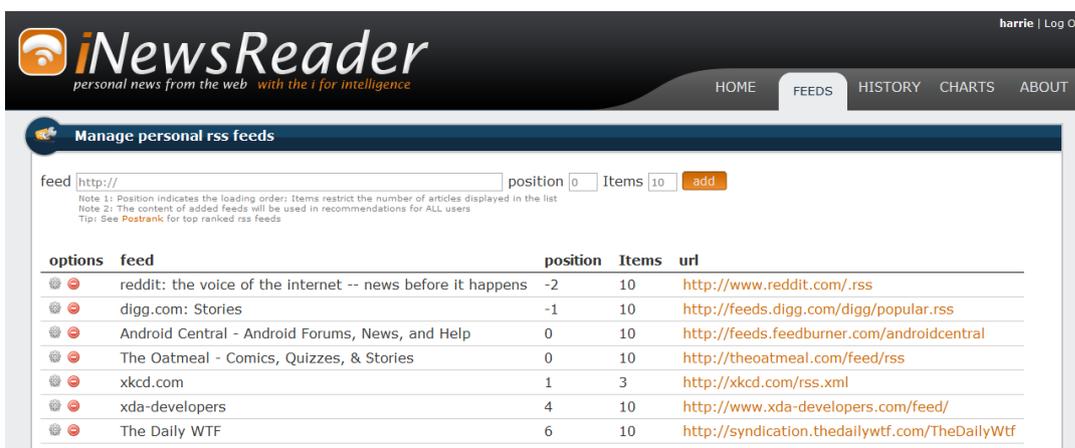


Figure 16: Manage personal RSS feeds

The portal is built using a scalable 100% screen width layout. This way higher screen resolutions also display more information. The layout of the page is constructed using Cascading Style Sheets (CSS) to decouple the page html source from the graphical interface and thereby increase the flexibility and control of the interface. The portal is furthermore optimized and tested for the Firefox and Google Chrome web browsers.

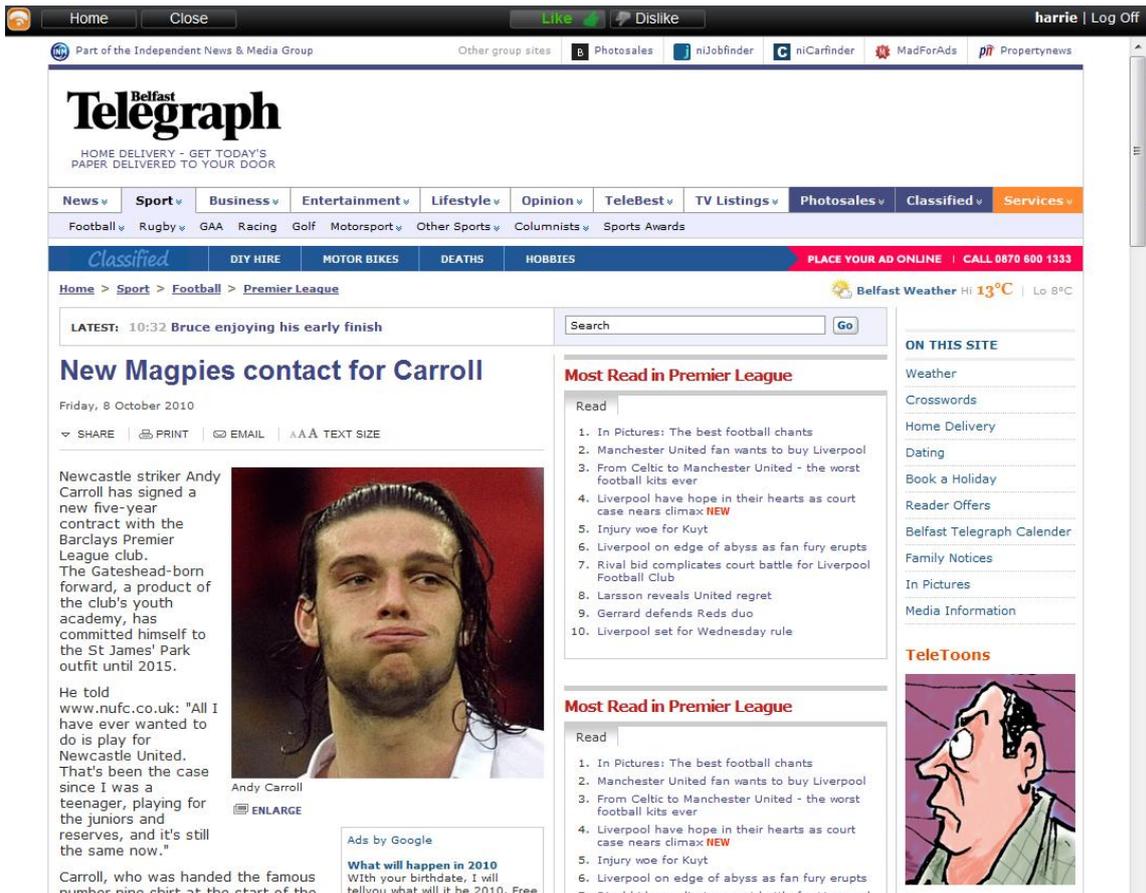


Figure 17: User Interface (article + voting)

If the user has selected an article for reading, the reading interface (Figure 17) is shown. Again this interface is split in two layers. On top the *iNewsReader* toolbar, providing functionality for voting, authentication and main navigation buttons (home and close buttons). Below the requested article is displayed.

Technically the screen is split using three html frames, a main frame handling all communication and two inner frames, one loading the toolbar and another displaying the article. This way an external web page can be displayed without leaving the *iNewsReader* domain and thereby preserve all inner database communication possibilities. The external source is automatically loaded into its own security sandbox, so no direct communication is possible between the split domains. Disadvantages of this setup are the outdated and discouraged technology of using frames and some sites don't allow to be displayed inside a frame. These websites use a so called frame bust script (Figure 18), causing the toolbar to be removed and thereby disable the possibility to vote for the article.

```
<script type="text/javascript">
  if (top.location != location)
    top.location.href = document.location.href;
</script>
```

Figure 18: Frame bust script

To overcome the problem of frame busting scripts, the main frame includes an anti frame bust script (Figure 19). This script detects the unloading of the inner

web page and prevents the busting out. The trick is to load a 204 non-displayable error page before the page unloads to stop the browser from acting.

```
<script type="text/javascript">
  var prevent_bust = 0;
  window.onbeforeunload = function () { prevent_bust++; }
  var interval = setInterval(function () {
    if (prevent_bust > 0) {
      prevent_bust -= 2;
      top.location = 'http://clients1.google.com/generate_204';

      articleFrame.location = "";
      articleFrame.document.write("
        This page does not allow to be framed.
        Click <a href='url'>this link</a> to open the article
        in an external window.
        For voting, use the above buttons on top of this page.
      ");
    }
  }, 1);
</script>
```

Figure 19: Anti frame bust script

To prevent endless loops and crashing of the browser, an empty page is displayed with a message and a link to open the article in a new window. This way the article can still be read and also voting functionality is preserved.

At the moment the user activates (i.e. open browser tab) the reading interface containing the article, a timer starts running to measure the time spent reading. Both the start and ending (close window) timestamps are sent to the profile server to account for implicit feedback.

Additional pages available to the user within the portal are an ‘about’ page, containing information about the project and the usage of the system. See also appendix 7.2 for the provided usage instructions to the user. Furthermore a ‘history’ page is available with links to personal recommendations from the past and a ‘charts’ page with statistics about the system usage and recommendation performance.

### 3.7 Recommendation

The recommendation component of the framework contains the methods to estimate the user interest for a set of articles. These methods are based on machine learning algorithms for text classification as described in section 2.3. The final implementation of the system is equipped with two of these algorithms, *Naïve Bayes* (paragraph 2.3.3) and *SVM* (paragraph 2.3.5). Also additional recommendation functionality (i.e. *k-NN*, collaborative algorithms or artificial neural networks) could be added here. Recommender agents called from the portal use these algorithms to present a personalized list of articles.



table is queried (online) every time a user calls (refresh) for a new set of recommended articles at the portal. If a new recommendation is available, it will be displayed; otherwise a random set of articles is returned (see also ‘random agent’; paragraph 3.8.3).

### 3.7.2 Naïve Bayes recommender

The Naïve Bayes recommender (RecommenderBayes in Figure 20) is based on the theory from paragraph 2.3.3. The exact implementation is a modified version of the open source C# Naïve Bayes classifier provided by Joel Martinez [43]. For reference, the source code of the Naïve Bayes classifier is attached (appendix 7.3).

Furthermore, the theory from paragraph 2.3.3 is extended with a *log-likelihood* calculation. As a consequence of the huge dimensionality of the data, chances can become extremely small and therefore buffer underflows are likely to occur. These buffer underflows can be prevented by using the logarithmic values of the individual term probabilities. The classification targets two mutually exclusive alternatives (‘like’ and ‘dislike’); therefore the usage of log-likelihood ratios does not influence the outcome of the final recommendation.

### 3.7.3 Support Vector Machine recommender

The Support Vector Machine recommender (RecommenderSVM in Figure 20) is based on the theory from paragraph 2.3.5. The implementation is based on the open source LIBSVM [16], ported to SVM.NET by Matthew Johnson [38].

The SVM recommender is equipped with functionality for three stages of processing: *Loading*, *training* and *testing*. First a recommender agent loads the training data to the SVM recommender. This training data contains the article feature vectors with *tf-idf* term values (paragraph 2.2.3) linked to a class label obtained from the user feedback (implicit or explicit). From this data a new classification problem is created which includes information about the dimensionality of the data-space (number of known terms). Next, during training, a model is trained using the problem and a parameter selection based on *5-fold grid search* optimization. Because the number of features is large [32], the model is trained using a LINEAR kernel:  $K(x, y) = x \cdot y$ , to improve performance. For this reason the parameter selection is reduced to only search for the optimal C value (cost constant; penalty parameter of the error term) to prevent for the overfitting problem. Once the model is trained, new articles are classified and class label probabilities are calculated at the last stage during testing. The outcome of this classification is then used by the recommender agents to create the final SVM based recommendation.

### 3.7.4 Recommender agent configurations

The recommender component is interfaced by configurations of recommender agents. These agents are responsible for the communication with the portal. The configurations are a combination of settings on recommender type, feedback model and memory model. In this research eight agents are configured:

Recommender type	Support Vector Machine		Naïve Bayes	
	long-term	short-term	long-term	short-term
Explicit feedback	1	2	3	4
Implicit feedback	5	6	7	8

Table 1: Recommender configurations

Depending on the recommender type, first one of the implemented agents is initialized (SVMRecommenderAgent or BayesRecommenderAgent in Figure 20). These agents collect and format the training data to be sent to the corresponding recommender in the recommenders module. The short-term agents thereby collect articles displayed within 24 hours. Long-term agents use all available (limited to 365 days) articles displayed to the user during system usage. Furthermore, explicit agents only collect articles the user explicitly voted for using the toolbar vote buttons at the portal reading interface. Implicit feedback is calculated using reading times. Therefore the mean reading time (in seconds) of voted liked ( $L$ ) and disliked ( $D$ ) articles is used to calculate the implicit feedback, given by the following formula:

$$\frac{1}{2}(\bar{L} + \bar{D}) = \frac{\frac{1}{n}\sum_{i=1}^n L_i + \frac{1}{m}\sum_{j=1}^m D_j}{2}$$

Formula 14: Implicit feedback

Where  $n$  indicates the available liked and  $m$  the number of disliked articles.

The final implicit feedback label is obtained by comparing the reading duration of displayed but non voted articles to the value obtained from Formula 14. A ‘like’ label (1) is attached when the reading time is longer, lower values get a ‘dislike’ (-1).

## 3.8 Additional agents

Lots of different agents can be thought of performing some kind of action at the RSS input of articles. A few example agents (see also Figure 7) are: *static feed*, *collaborative*, *content-based*, *ranking* and *search agents*. For this research a pre-configured set of agents is implemented. Descriptions of the main content-based classification agents are already given above. For a discussion on more potential non-implemented agents see also paragraph 5.3.2. This section concentrates on the implemented additional agents indirectly contributing to the recommendation process.

### 3.8.1 Search agent

A search agent is implemented to increase the functionality to faster train the recommendation system with articles of known interest. This agent is activated by providing a search query in the search field at the portal home page. By pressing the search button, a search agent is executed to query the items-server for the provided keywords. Therefore a default text search is used on the ‘title’ and ‘description’ fields in the database. These fields are populated by the raw data as provided by the RSS feed. A list of max  $n$  articles is returned and displayed back at the portal to be read and voted for next using the reading interface. During the experiment,  $n$  is set to 10 articles. Additionally, the articles are ordered by date (order agent), where newer articles are displayed on top.

### 3.8.2 RSS feed agent

As mentioned in paragraph 3.6.3, see Figure 16, users are able to provide a list of RSS feeds of personal interest. Again these feeds increase the learning speed of the system, by providing feedback on articles of known interest. On page load of the portal home page, each of these configured feeds starts its own RSS feed agent. In the first place this agent calls a crawler to harvest the contents of the

RSS feed. Because an article identifier (ArticleId) from the item-server is needed to register votes to an article, first the non-registered articles are added to the data storage. The article ids are returned immediately and together with the already available information from the RSS feed, a list of  $n$  articles is displayed at the user interface. The activated crawler continues with the full article processing routine as a background service, so each article can be used for future recommendations. This way, only a small delay, caused by the registration of the new articles at the item-server, is noticeable to the user. The full (slower) processing of the articles is executed offline and the users are still already able to provide feedback (vote) for the new articles not yet fully processed.

### **3.8.3 Random agent**

For research purposes also a random articles agent is implemented. This agent uses the last  $n$  articles from the item-server to collect a list of  $i$  articles. During research,  $i$  is set to 10 and  $n$  to 10.000. The data is limited by the newest  $n$  articles to prevent the identification of the random agent by examination of the articles release dates.

### **3.8.4 Order agent**

Finally a basic ordering agent is provided to sort the provided list of articles. If possible, the list is ordered by using the likelihood parameter or ‘degree of interest’ as provided by the text classification algorithms from the recommendation module. If this parameter is not available, the list is ordered by release date of the articles. Alternatively, if even the release date is unknown, the ordering is based on harvest date.

## **3.9 Shortcomings**

Due to choices made at the implementation stage of the framework, the system also has some shortcomings affecting the flexibility of the experimental setup. The main weakness as already mentioned before is speed. Also the reduced functionality of the Agent Management System (AMS) forces the behavior of the agents into a single direction. These and other shortcomings caused by the methods as described above are discussed in the next paragraphs. For an additional discussion about missing components and future work, see also section 5.2.

### **3.9.1 Scaling**

Initially the system is built for running on a single testing machine (section 3.10). All framework modules are running simultaneously on this single development server. The possibility for up-scaling to larger server clusters or cloud computing services is kept in mind during implementation. All modules and services are built independently and use Object Oriented Programming (OOP) for structuring and decoupling of the components. Another bottleneck is the usage of the relational databases. Because the system is both read and write heavy, large disk seek times are the consequence. The usage of fast *key-value* stores (NoSQL; i.e. Google BigTables, Membase or Microsoft FlashStore) combined with memory caching models (i.e. MemCache) can possibly increase data transits enormously. Rerouting data can be achieved by rewriting the data access layers handling all database manipulations.

The speed and capacity of the system certainly needs to be kept in mind. Because of these limited system resources, the experiment is setup to measure the system usage and performance of the components using only a small user base. This also immediately excludes experiments using collaborative algorithms.

### 3.9.2 Stacking agents

Although the core experiment of testing multiple recommender agent configurations isn't influenced, also the reduced functionality of the Agent Management System (AMS) has some drawbacks. The functionality of the AMS is reduced to activating of, and communicating with single agents. Therefore stacking of agents is excluded. So it is not possible to refine or combine the results of multiple agents into a single recommendation. The stacking of agents is known to result in more refined recommendations (see also paragraph 2.1.5); this functionality is left for future research. Also communication between agents is left out of the system. This way the interesting property of *self-organization* within multi-agent systems is excluded. The usage of multi-agent frameworks (see FIPA [24]) can possibly overcome these shortcomings. See also paragraph 5.3.1 for a discussion on possible future work on advanced agent frameworks.

The main influence of the above on the experiment is the queue for the agent recommendations. The recommendations of the unique agent configurations are calculated offline and stored at the user-server. Each time a user requests a new recommendation the oldest available recommendation from the queue is returned and displayed at the portal. Therefore it takes  $n$  'rounds' before an article vote is included into a new recommendation, where  $n$  is the amount of available recommendations in the queue. By combining the results, this problem could be resolved, but then performance measurements for the individual agent configurations become much harder and can be dubious.

## 3.10 Resources

The implementation of the recommender system is completely based on the C# programming language combined with ASP.NET for web development. Using the Microsoft Visual Studio Express 2010 edition, a prototyping environment is available for free. The main advantage is that all sub-parts of the system are written using the same programming language. Furthermore a set of open source code libraries is directly available for use. The main development and testing systems have the following specifications:

Development Server (Desktop PC):

Processor: Intel Core i7 930 at 2.8 GHz

Memory: 6 GB DDR3

OS: Microsoft Windows 7 (64 bit)

Testing System (Notebook):

Type: Acer Aspire 6935G

Processor: Intel Core 2 DUO T9400 at 2.53 Ghz

Memory: 4 GB DDR3

OS: Microsoft Windows 7 (64 bit)

Both systems interact using a 1 GB LAN network. This network is accessible from the internet through a 20000/1024 Mbps line.

# 4 Experiment & results

To answer the research questions from section 1.4, a small experiment is conducted using the implemented *iNewsReader* framework described in the previous chapter. In this experiment a limited set of subjects use the system to train their interests for some longer period of time. During this period the measurements described below are gathered for the different agent configurations and system usage in general. A controlled experiment is conducted by comparing the performance of the recommender agents against the data from the lists of random articles. The results of the experiment are described in section 4.2.

## 4.1 Experiment

The experimental setup of the *iNewsReader* framework is biased to test the final recommendation performance of the system. Although it is known the user interface and crawler are of great influence, as explained, no explicit experiments are conducted on these and the other components.

### 4.1.1 Setup

To test the final recommendation performance, users are given access to the system and asked to use it for personal news aggregation spread over multiple days for some longer period of time. At the introduction, these subjects are pointed to the instructions and notifications from appendix 7.2. During usage of the system, the recommendations from the eight recommender agent configurations are displayed one at a time. Besides, also a forced ten percent of the recommendations are given by a random selection of articles, gathered by the ‘random agent’. If a recommendation is gathered by an advanced agent or is just a random selection of articles, is unnoticeable to the user. This way the random selections represent the control samples to test against. Each recommendation contains ten articles, with the exception when an agent cannot classify this amount.

Finally, also some accounts are created to train the recommender system using only articles on a pre-defined subject. The difference in performance on both wide (multiple) and narrow interest areas can be compared this way.

### 4.1.2 Data

The news article data available for the experiment is gathered by approximately one month of (ongoing) harvesting. The continuously growing queue of RSS feeds contains above 50.000 news sources from all around the web. From these feeds, above a million articles are parsed and stored at the item-server.

### 4.1.3 Measurements

Multiple measurements can be taken to answer the research question. Many possible measurements regarding the performance of recommender systems are described in [26]. To answer the first sub research question, recommendations and the included articles are registered and counted. Also the items opened from a recommended list of articles are registered. For each article timestamps are automatically recorded at the opening and closing actions to register reading time. These measurements are already available for implicit interest indication and can also be used to answer the second research question. The performance

of the recommendation can finally be analyzed by combining the above with the results of the explicit user feedback (votes).

## 4.2 Results

### 4.2.1 Framework

As mentioned before, no explicit experiments are conducted to test the performance and behavior of the framework itself. For the framework in general can be stated the implemented system worked as expected. All components described, behaved as designed. The main remark is the execution time. During implementation ‘function’ was stated above ‘speed’, meaning a lot of individual processes can be optimized.

### 4.2.2 Data

During the experiment a total of six subjects used the system for personal news aggregation. From these users, three are ‘normal users’ (users 1 - 3) and three are forced to vote positive for all articles on a single pre-defined subject: ‘apple’, ‘dutch’ and ‘sports’. The collected data exists of data points representing a single recommendation with a set of articles. For each data point the values from Table 2 are available for analyzing.

Data	Range	Description	
id	1 ... $N$	Recommendation identifier in the data storage	
type	0; 1 – 8	Type recommendation / agent configuration (See Table 1); 0 = random	
date	dd-mm-yyyy	Storage date the agent completed the recommendation	
User	1 – 6	The user recommended to	
TotalArticles	1 – 10	Number of presented articles within this recommendation	
OpenedArticles	0 – 10	Number of articles opened by the user	
Likes	0 – 10	Number of like votes	
Dislikes	0 – 10	Number of dislike votes	
Unvoted	0 – 10	Number of articles opened (read) but not voted for	
ReadingSeconds	0 ... $N$	Total time spend reading opened articles (in seconds)	
ReadingLikeSeconds	0 ... $N$	Time spend reading articles voted like (in seconds)	
ReadingDislikeSeconds	0 ... $N$	Time spend reading articles voted dislike (in seconds)	
Calculated	Range	Formula	Description
LikesDislikes	-10 – 10	Likes – Dislikes	Likes votes minus dislike votes
% Likes	0 – 100%	Likes / TotalArticles	Percentage of like votes
% Dislikes	0 – 100%	Dislikes / TotalArticles	Percentage of dislike votes
% LikesDislikes	-100% - 100%	LikesDislikes / TotalArt.	Percentage of like minus dislike votes
TotalVotes	0 – 10	Likes + Dislikes	Likes plus dislike votes
CumSumVotes	0 ... $N$	SUM(TotalVotes; 0 ... id)	Cumulative sum over votes

Table 2: Data structure for the results of the experiment

Starting with a global indication of the data, in Table 3 the distribution of the number of recommendations over users and agent configurations (see Table 1) is displayed. This data is graphically displayed in Figure 21.

Agent:	random	1	2	3	4	5	6	7	8	Total
apple	6	2	1	3	1	2	2	2	1	20
dutch	13	5	4	14	12	4	3	11	11	77
sports	14	4	5	7	6	5	5	6	5	57
user 1	11	6	7	4	4	5	4	3	3	47
user 2	238	41	29	30	24	39	28	25	21	475
user 3	68	9	18	15	11	11	12	15	10	169
<b>Total</b>	<b>350</b>	<b>67</b>	<b>64</b>	<b>73</b>	<b>58</b>	<b>66</b>	<b>54</b>	<b>62</b>	<b>51</b>	<b>845</b>

Table 3: Number of recommendations

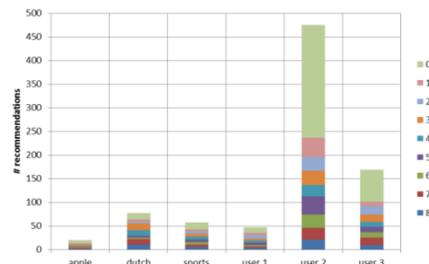


Figure 21: Distribution of recommendations

So during the experiment 845 recommendations are presented to the users, from which 350 are 'random' and the remaining 495 are based on one of the agent configurations. Within these recommendations, a total of 7722 articles are displayed, from which 2312 are opened. 3371 articles were random (701 displayed) the other 4351 where recommended by one of the agents, from which finally 1611 are opened. Also can be seen the second user (user 2) has processed a lot more recommendations as the other users, but also half of these recommendations are random. Finally it needs to be mentioned that the data contains a sampling bias, caused by the low number of users.

#### 4.2.3 Article selection

The first sub- research question from section 1.4 assumes: "If more articles of personal interest are selected, the user will most likely open a higher percentage for reading". To verify this assumption, the percentage of opened articles over time is displayed for each user in the figure below.

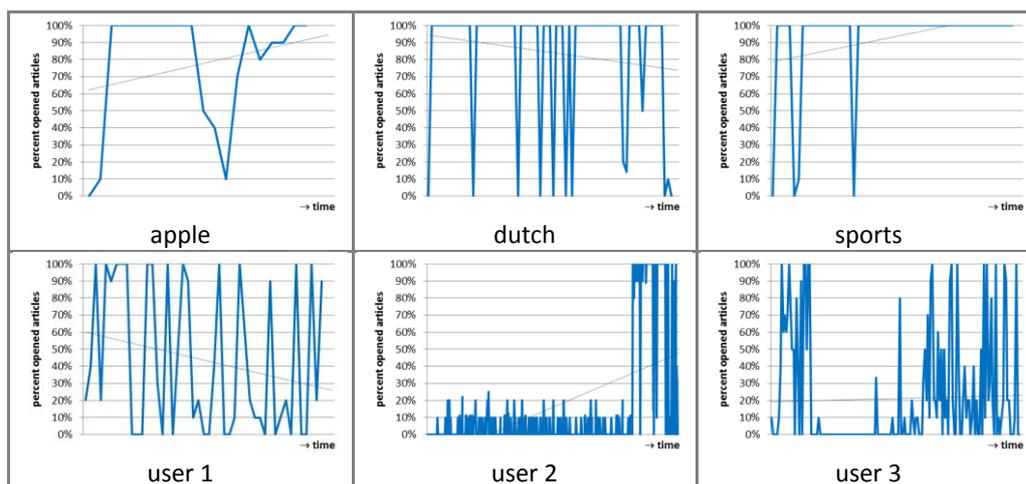


Figure 22: Percentage opened articles per user over time

From these graphs in Figure 22 the following can be observed:

1. For the specific subjects (top row): Mostly all and sometimes none of the recommended articles are opened for reading.
2. For the normal users (bottom row): Diverse opening patterns are displayed; showing all three possible trends (increase; decrease; steady).
3. There is no clear noticeable increase in opening of articles over time.
4. Both at the beginning and at the end of the trials many 100% openings are observed.

At the start of the training, the recommendation results cannot be optimal. But a close to 100% openings at start of the trials for multiple users can be clearly noticed. A possible explanation can be the selection of non-interesting articles for penalty votes (dislikes). So, presumably the number of articles selected for reading does not solely depend on the amount of presented articles of interest. More affects need to be considered (i.e. voting strategies or psychological effects on both positive and negative voting). Finally, the consequence of the above is that the stated assumption of the correlation between the article selection and openings needs to be reconsidered. Therefore the corresponding sub- research question cannot be answered adequately to support the main research question and will be omitted in the remaining analysis. Nevertheless the first part of this question: "Which techniques can be used ..." rephrased to netnews recommendation in general, is already mainly answered from literature and by the proposed agent framework in the previous chapters.

#### 4.2.4 Voting strategies

After approximately one week of usage, one of the subjects (user 2) was disappointed with the results so far. Therefore this user was given some extra instructions about the basic workings of the internal system and hinted on how to train the system more effectively. Basically the user was told to vote more, both positive and negative. The effect is clearly visible in the graph below, which displays this user's voting (likes & dislikes) over time.

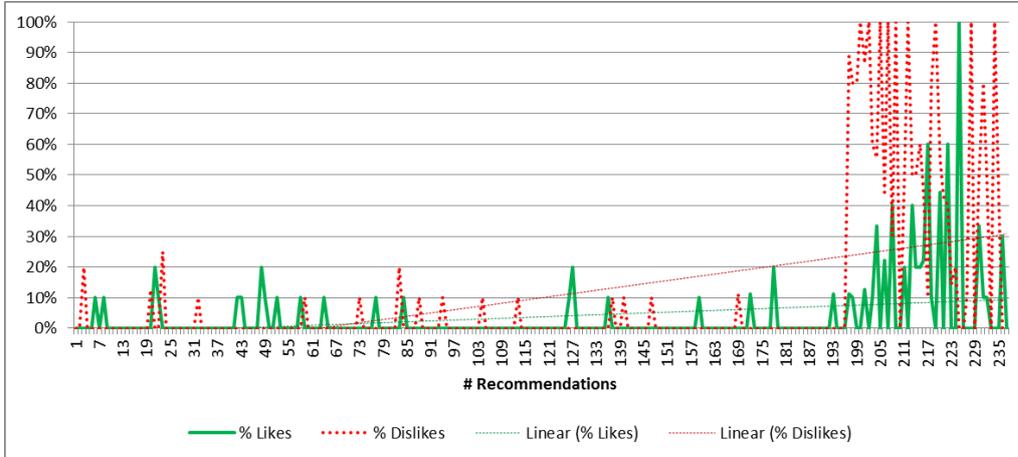


Figure 23: Explicit feedback user 2

As expected, the graph displays almost no positive result. To compare the effect of minimal voting vs extensive voting, the graph is split in two parts, the first part representing the first 190 recommendations and the last part including the final 45. In Figure 24 the effect is displayed by using the subtraction of dislikes from the likes. Notice there is no learning effect visible in the first part (top-left) and a small positive trend in the second part (top-right). Finally, also the reading time is plotted for the final 45 recommendations (bottom-right). There can be seen the total time spent reading articles voted 'like' does slightly increase over time and the articles voted 'dislike' decrease. From this can be concluded the voting strategy (number of votes over time; both positive and negative) influences the outcome of the system and also voting has at least some effect on the final recommendation performance.

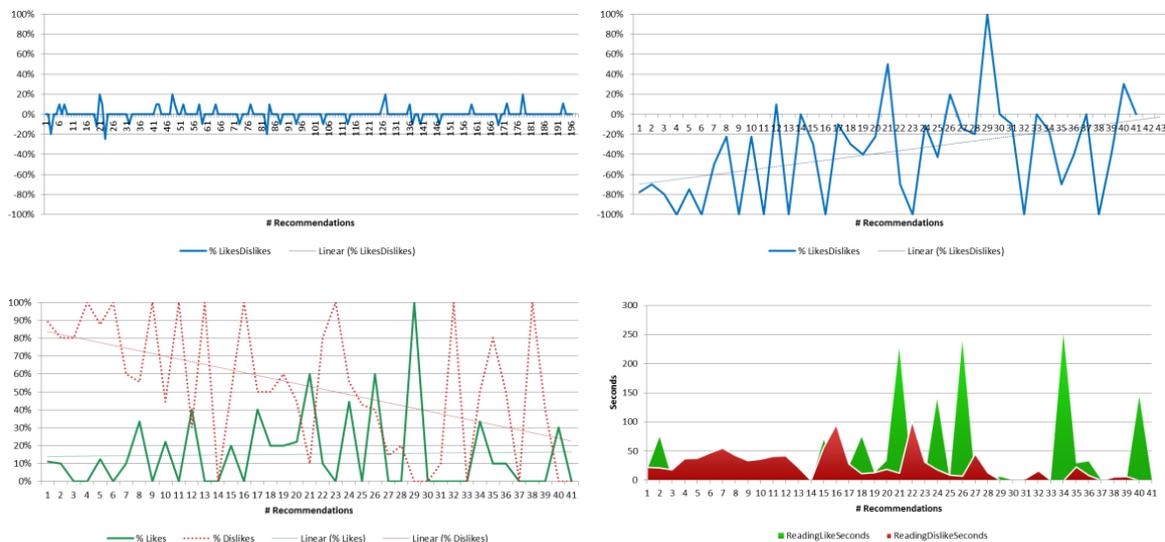


Figure 24: Splitted plots for user 2 (top-left: first part LikesDislikes; top-right: final part LikesDislikes; Bottom-left: Feedback final part; bottom-right: Reading time final part)

Another approach to overcome this problem of less vs. intensive voting is to plot against the number of votes instead of time (number of recommendations). The graph below displays the same data plotted against the cumulative number of votes (CumSumVotes from Table 2).

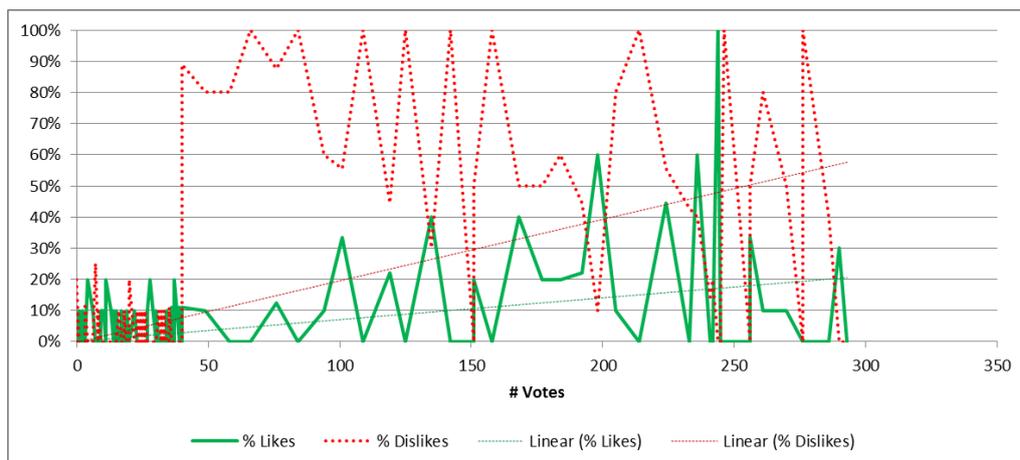


Figure 25: Explicit feedback user 2 plotted against number of votes

This way, the effect of voting is much clearer. But unfortunately nothing useful can be said about the performance of the recommender system or its components based on the presented data and plots for this single user, other than a slightly positive trend is noticeable. Furthermore, this trend is only visible after the given instructions to the user. Thus at daily usage under default conditions users probably would provide less votes as the users in this experiment and therefore their learning effect will probably be less visible.

#### 4.2.5 Agent configurations

Multiple measurements (Likes, LikesDislikes, ReadingSeconds, etc.) from Table 2 can account for the recommendations performance indicator. To be able to compare the performance of the different agent configurations a single measurement is chosen. At first thought, the difference between like and dislike votes (LikesDislikes) would be most accurate, because thereby all explicit user feedback is used to test against. But the importance and continuity of the dislike vote value can be argued against. In general, those negative votes indicate a 'penalty' for the system, as in "This is not what I want" or "Skip these articles the next time" or "I've already seen this subject enough". Many other reasons can be thought of. On the other hand, the like votes only indicate: "I like this, select similar articles in the future". Also assumedly users more likely vote for articles of interest as for non-interesting subjects. This concludes the value of the like and dislike votes is not equally distributed and also the like votes probably are a more accurate performance indicator. For that reason finally solely the explicit like votes are chosen to indicate for the recommendation performance in the following analysis.

To answer the final research questions about the influences of the individual system components, the performance of the agent configurations is next examined by combining the results of all users. A first global indication is given in Table 4; displaying the default descriptive statistics and corresponding box plots for the like votes per agent configuration. The agent numbers in this and all following statistics and graphs represent the configurations from Table 1; Zero (0) represents the random selection.

	0	1	2	3	4	5	6	7	8	box plots
Count	350	67	64	73	58	66	54	62	51	
Minimum	0	0	0	0	0	0	0	0	0	
Maximum	4	10	9	10	10	10	8	10	10	
1st Quartile	0	0	0	0	0	0	0	0	0	
Median	0	0	0	0	0	0	0	0	0	
3rd Quartile	0	1	1	2	2	1	2	2	3	
Sum	82	66	46	126	95	76	73	114	95	
Mean	0,234	0,985	0,719	1,726	1,638	1,152	1,352	1,839	1,863	
Variance	0,260	4,560	2,364	7,896	7,533	4,900	4,912	9,908	8,441	
Std. dev.	0,510	2,136	1,538	2,810	2,745	2,214	2,216	3,148	2,905	

Table 4: Descriptive like vote statistics per agent configuration over all users

From these statistics (Table 4) can be noticed:

1. These statistics do not include time.
2. The mean, variance & standard deviations of the agent configurations are generally much higher as the values for the random selections. This indicates better performance and more divergence in article votes.
3. Most agents received a score of 10 like votes at some moment in time; the random selections received a maximum of 4.
4. All medians are zero. Showing most of the time, no positive feedback is given.
5. Configuration 8 (Implicit short-term Naïve Bayes) has the highest mean and 3<sup>rd</sup> quartile among the agents. So relatively, this agent received the most positive votes. Differences with other configurations are minimal.
6. The SVM agents (1, 2 & 5) received less positive feedback.

This data can be further analyzed by using a statistical hypothesis test with:

$H_0$ : The samples come from the same population.

$H_a$ : The samples do not come from the same population.

Many of the default (parametric) statistical tests (T, F, Z,  $\chi^2$ , ANOVA, ANCOVA, etc.) have all or most of the following assumptions:

1. The scale on which the dependent variable is measured has the properties of an equal interval scale
2. The k samples are independently and randomly drawn from the source population
3. The source population can be reasonably supposed to have a normal distribution
4. The k samples have approximately equal variances.

Within this research none of the above assumptions apply and a more advanced non-parametric test needs to be selected to further compare the agent configurations. Therefore a Kruskal-Wallis analysis is chosen to test the equality of population medians among groups. This analysis leaves assumptions 1, 3 & 4, but still assumes independent and random samples and thereby still omits time. The results of the test on the data from Table 4 are given in the table below.

K (Observed value)	57,719
K (Critical value)	15,507
DF	8
p-value (Two-tailed)	< 0,0001
alpha	0,05

Table 5: Kruskal-Wallis test over all users

The computed p-value is much lower than the significance level ( $\alpha=0,05$ ) and therefore the null hypothesis  $H_0$  is rejected, and the alternative hypothesis  $H_a$  is accepted with a risk of 0,01% (type I error). Rephrased; within this measurement the samples do not come from the same population, which concludes there is a difference in agent performance. By applying a pairwise comparison on the data, more can be said about the individual differences between the agent configurations and the random samples. This comparison is displayed in Table 6 by using a Dunn's procedure / Two-tailed test.

Sample	Frequency	Sum of ranks	Mean of ranks	Groups
0	350	129394,500	369,699	A
1	67	27441,000	409,567	A B
2	64	27115,500	423,680	A B
5	66	28850,000	437,121	A B
6	54	24733,500	458,028	A B
7	62	29361,000	473,565	B
4	58	28271,000	487,431	B
3	73	36455,000	499,384	B
8	51	25813,500	506,147	B

Table 6: Multiple pairwise comparisons using the Dunn's procedure / Two-tailed test

From this comparison two groups are formed, group A and B. The first group (A) displays the unwanted result of the similarity between the random samples and the SVM configurations (1, 2, 5 & 6). The other group (B) displays the expected result of the boundary between the agent configurations and the random samples. To go one step further, these groups are drawn from the following reciprocal distribution of p-values (bold values are significant):

	0	1	2	3	4	5	6	7	8
0	1								
1	0,131	1							
2	0,045	0,683	1						
3	<b>&lt; 0,0001</b>	0,007	0,026	1					
4	<b>&lt; 0,0001</b>	0,028	0,076	0,732	1				
5	0,011	0,422	0,699	0,064	0,158	1			
6	0,002	0,181	0,348	0,245	0,432	0,565	1		
7	<b>0,000</b>	0,067	0,157	0,450	0,701	0,298	0,673	1	
8	<b>&lt; 0,0001</b>	0,009	0,027	0,852	0,622	0,062	0,213	0,384	1

Table 7: Distribution of p-values among agent configurations like votes for all users

Additional notable less significant values ( $c < 0,01$ ) from Table 7 are the relations between configurations 1 and 3 (0.007; two long-term explicit agents) and 1 and 8 (0.009; no similarities). This last observation is inexplicable. Another configuration performing close to random is the implicit short-term SVM configuration (6) with significance 0.002.

Next the influence of the users on the above findings is explored. Therefore first the full distribution of the positive votes among the users is displayed in Figure 26 below. From this graph can be seen the users dedicated to specific subjects (first three) have in total provided more positive votes compared to the other users.

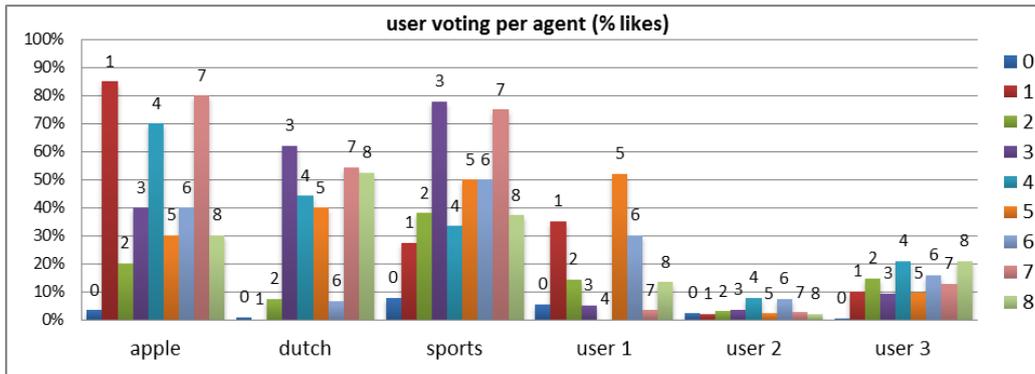


Figure 26: Like votes distribution for each user per agent; Numbers represent agent configurations

If the results from these user groups are combined, the following pie charts can be constructed (Figure 27):

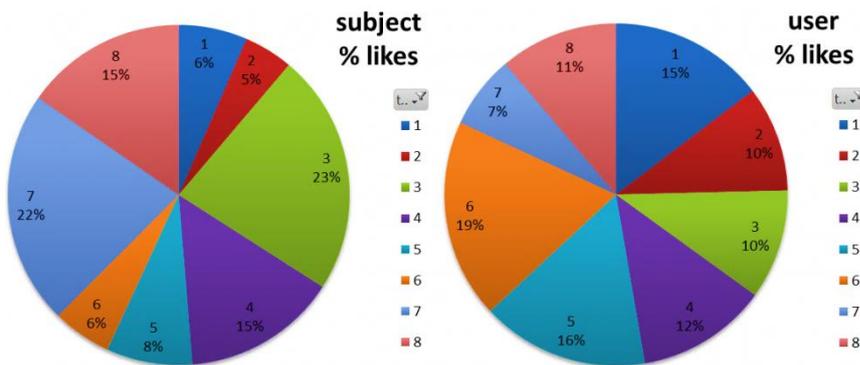


Figure 27: Voting distributions per user group (excl. random)

The remarkable observation in the above pie charts is the difference between groups in voting for Support Vector Machine configurations (1, 2, 5 & 6) and Naïve Bayes (3, 4, 7 & 8). When adding the numbers for the specific subjects, the result is: Bayes (75%) > SVM (25%); and for the normal users the other way around: SVM (60%) > Bayes (40%). This could indicate the Naïve Bayes recommenders perform better at narrow subject interests and SVM outperforms when wider and multiple interest areas are included. However, by means of the many relations, high abstraction and exclusion of time, a lot more causes needs to be considered. The final conclusion is that all configurations display some positive results and none of them outperforms them all in one or both groups given this data. But probably the most important observation is the difference in voting between the user groups, this need to be included in further analysis. Therefore first the Table 4 data is split for both user groups next:

	0	1	2	3	4	5	6	7	8
Count	27	11	10	24	17	11	10	17	16
Minimum	0	0	0	0	0	0	0	0	0
Maximum	3	10	9	10	10	10	8	10	10
1st Quartile	0	0	0	1	0	0	0	2	1
Median	0	0	1	4	2	2	1	7	3
3rd Quartile	1	6	3	7	7	6	4	9	8
Sum	14	28	20	99	63	35	24	96	66
Mean	0,519	2,545	2,000	4,125	3,706	3,182	2,400	5,647	4,125
Variance	0,567	14,273	8,889	13,679	15,846	11,764	9,378	14,368	15,050
Std. dev.	0,753	3,778	2,981	3,699	3,981	3,430	3,062	3,790	3,879

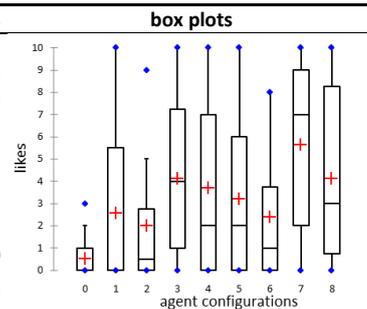


Table 8: Descriptive like vote statistics per agent configuration over specific subject users

	0	1	2	3	4	5	6	7	8	box plots
Count	129	48	46	41	35	47	42	35	31	
Minimum	0	0	0	0	0	0	0	0	0	
Maximum	4	6	5	4	5	7	7	4	5	
1st Quartile	0	0	0	0	0	0	0	0	0	
Median	0	0	0	0	0	0	0	0	0	
3rd Quartile	1	0	1	1	2	1	2	1	1	
Sum	68	38	26	27	32	41	49	18	29	
Mean	0,527	0,792	0,565	0,659	0,914	0,872	1,167	0,514	0,935	
Variance	0,407	2,722	1,051	1,130	2,022	3,201	3,898	0,963	2,462	
Std. Dev.	0,638	1,650	1,025	1,063	1,422	1,789	1,974	0,981	1,569	

Table 9: Descriptive like vote statistics per agent configuration over normal users

Again the large differences in voting between the groups are clearly visible. Additional noticeable differences are the larger median and mean values for the specific subject users (Table 8), indicating a more positive usage of the system. Also the lower maximum values of the normal users (Table 9) stand out. This indicates no close to perfect recommendations are received by them.

Also for this split user group data the same non-parametric hypothesis tests and pairwise comparisons are applied, the results are displayed below.

	specific subjects	normal users
K (Observed value)	26,626	5,195
K (Critical value)	15,507	15,507
DF	8	8
p-value (Two-tailed)	0,001	0,737
alpha	0,05	0,05

Table 10: Kruskal-Wallis tests per user group

Sample	Groups	0	1	2	3	4	5	6	7	8	
0	A	0	1								
1	A B	1	0,295	1							
2	A B	2	0,297	0,979	1						
6	A B	3	<b>0,000</b>	0,077	0,093	1					
5	A B	4	0,008	0,249	0,275	0,535	1				
4	A B	5	0,042	0,406	0,433	0,428	0,812	1			
8	A B	6	0,205	0,828	0,852	0,145	0,378	0,553	1		
3	B	7	<b>&lt; 0,0001</b>	0,014	0,018	0,332	0,142	0,123	0,032	1	
7	B	8	0,002	0,111	0,128	0,954	0,609	0,490	0,189	0,350	1

Table 11: Comparison specific interest users; left: Group comparison; right: p-values

Sample	Groups	0	1	2	3	4	5	6	7	8	
0	A	0	1								
1	A	1	0,118	1							
2	A	2	0,298	0,677	1						
3	A	3	0,602	0,421	0,692	1					
4	A	4	0,875	0,185	0,353	0,592	1				
5	A	5	0,228	0,774	0,896	0,600	0,291	1			
6	A	6	0,835	0,282	0,507	0,797	0,770	0,427	1		
7	A	7	0,185	0,957	0,741	0,489	0,237	0,833	0,346	1	
8	A	8	0,942	0,278	0,480	0,740	0,857	0,409	0,924	0,334	1

Table 12: Comparison normal users; left: Group comparison; right: p-values

From these tables can be noticed:

1. From Table 10: The data from the normal users is not significantly different. Meaning  $H_0$  can be accepted and thus for normal users the differences in performance are minimal (incl. random).
2. Table 11 displays that only the long term Bayes recommenders (3 & 7) distinguish themselves from the others for the specific interest data.
3. No statistical difference is measured for the like votes from the normal users between the agents, as displayed in Table 12. All samples are as close to random (group A) as to each other, with no significant p-values.

As mentioned multiple times before, the above statistics don't include time and assume the samples are independent. Time is an important factor because of the learning model and also ideally every next sample should outperform the previous so they are not independent. But it is much harder to evaluate the recommendation performance over time using the data from multiple subjects. Absolute time (in days) is not useful because the subjects used the system over multiple and different days. Furthermore, they did not spend an equal amount of time on news aggregation. Also the number of provided recommendations, as explained in the previous paragraph, does not always explain the results well and also cannot be combined for multiple users. Therefore the variable 'total number of votes' (incl. negative) is used as a time indication to be able to evaluate the agents' performance over time for the combined results from multiple users.

To evaluate the performance over time, for each agent configuration and for the different user groups a linear regression analysis is performed. The figures on the following pages display the linear trend lines, based on mean squares analysis. First all graphs will be displayed, an interpretation is given at the end. Each next page displays two figures; the top figure summarizes the trend lines over all agents, the bottom figure displays small graphs for all individual configurations. In these graphs, the data points (active) and three lines are displayed. The main line (model) indicates the linear regression, whereby agent performance is roughly represented by the slope of this line. The other two lines indicate the 95% confidence intervals. First the inner line represents the confidence interval on mean of the prediction for a given value. The outer line is the confidence interval on a single prediction for a given value. The title of each graph contains the configuration number and also displays the r-squared correlation coefficient ( $R^2$ ), which indicate a goodness of fit (the closer to 1, the better the fit). Finally, for all graphs, the horizontal axes represent time (by number of votes) and the vertical axes the positive feedback.

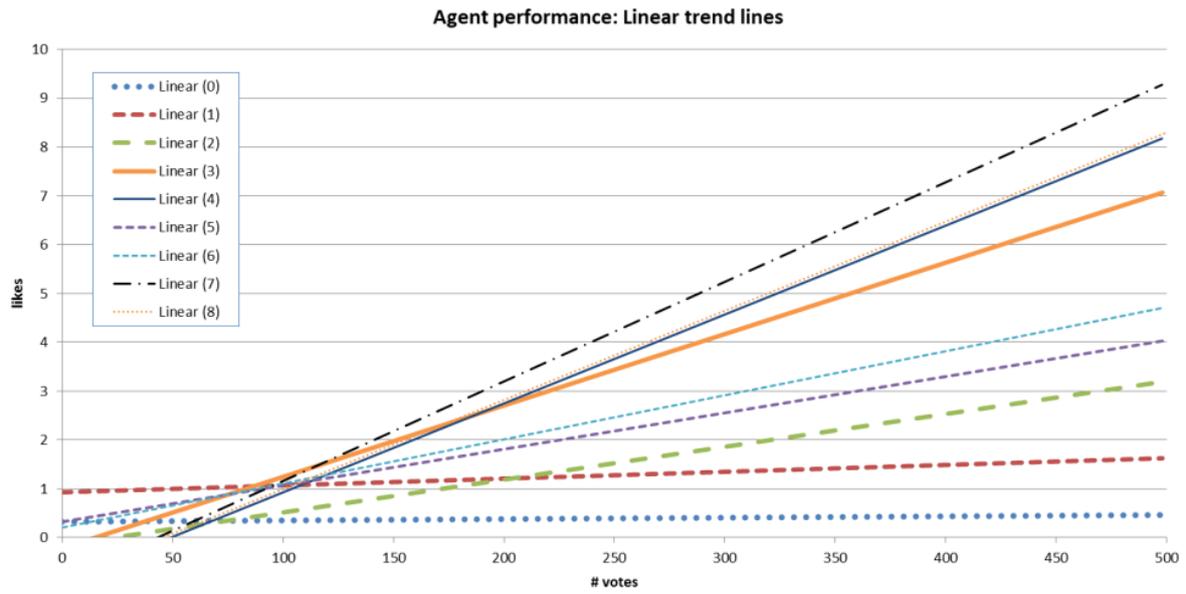


Figure 28: Overview of linear trend lines for each agent configuration over **all** user data

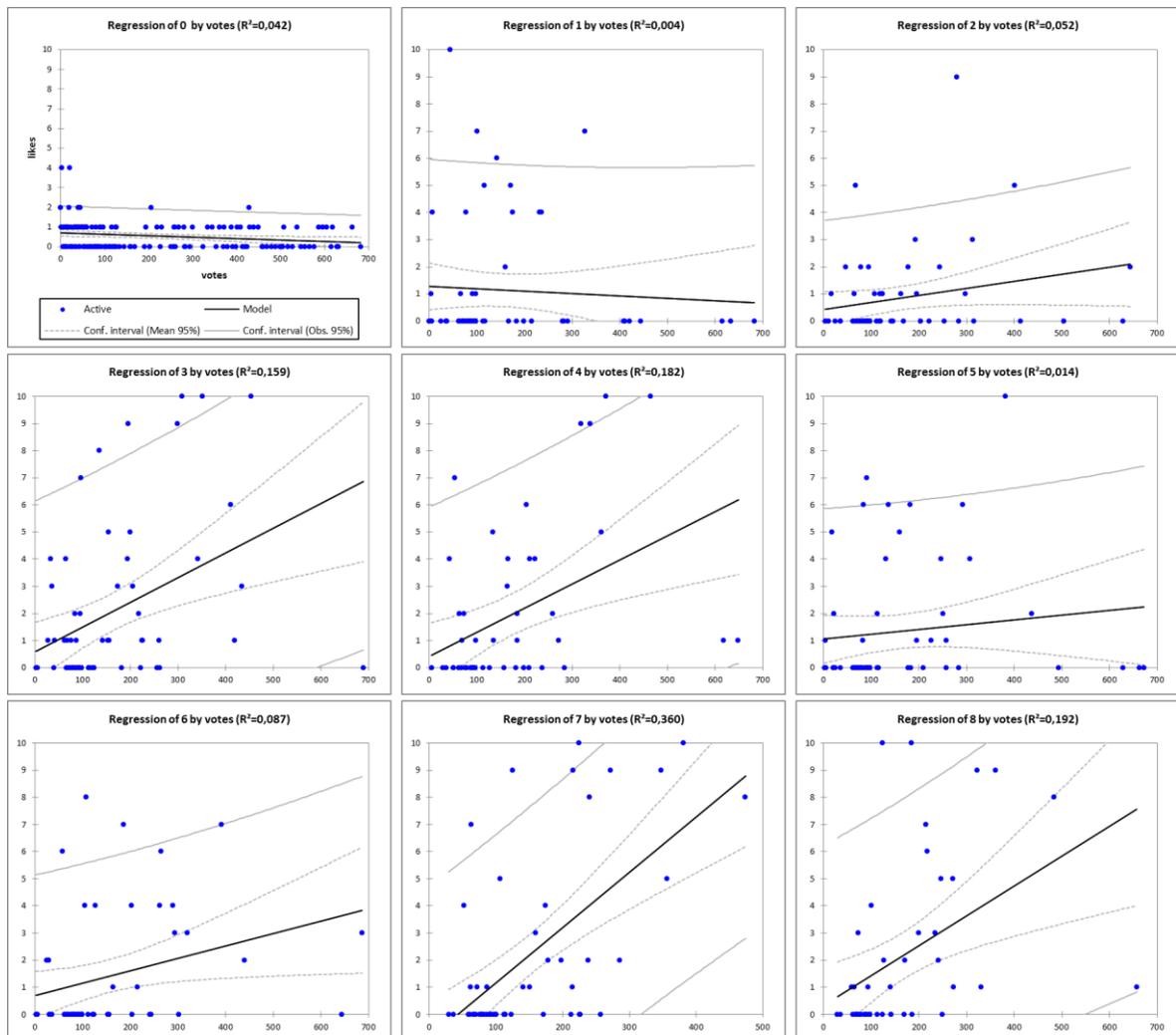


Figure 29: Individual linear regression graphs over **all** user data; incl. 95% confidence intervals

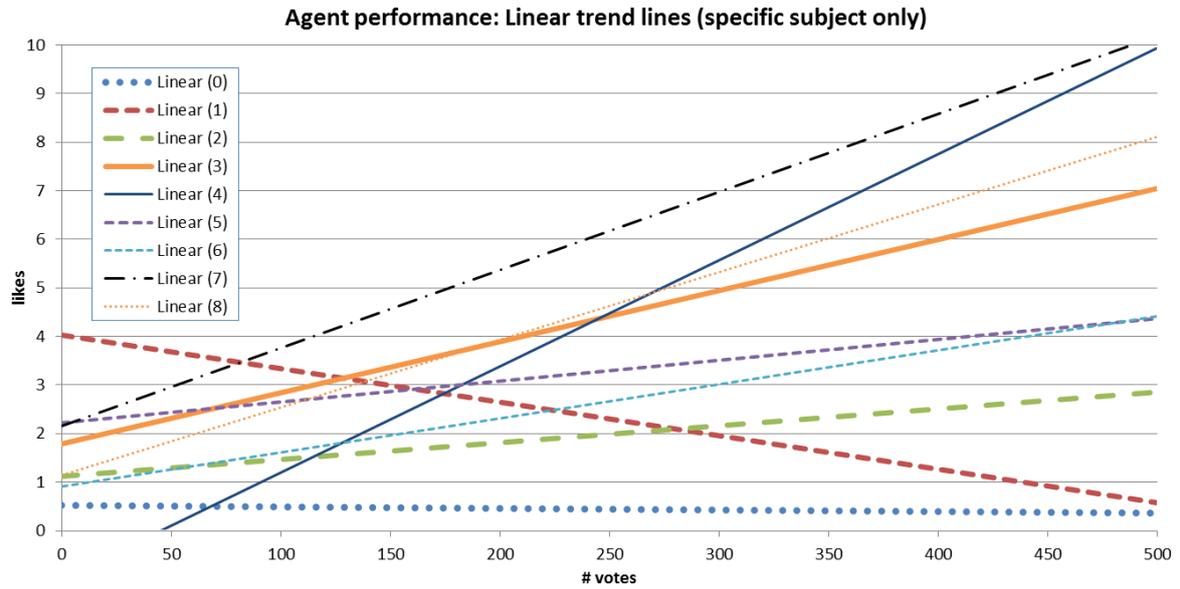


Figure 30: Overview of linear trend lines for each agent configuration over **specific subject** user data

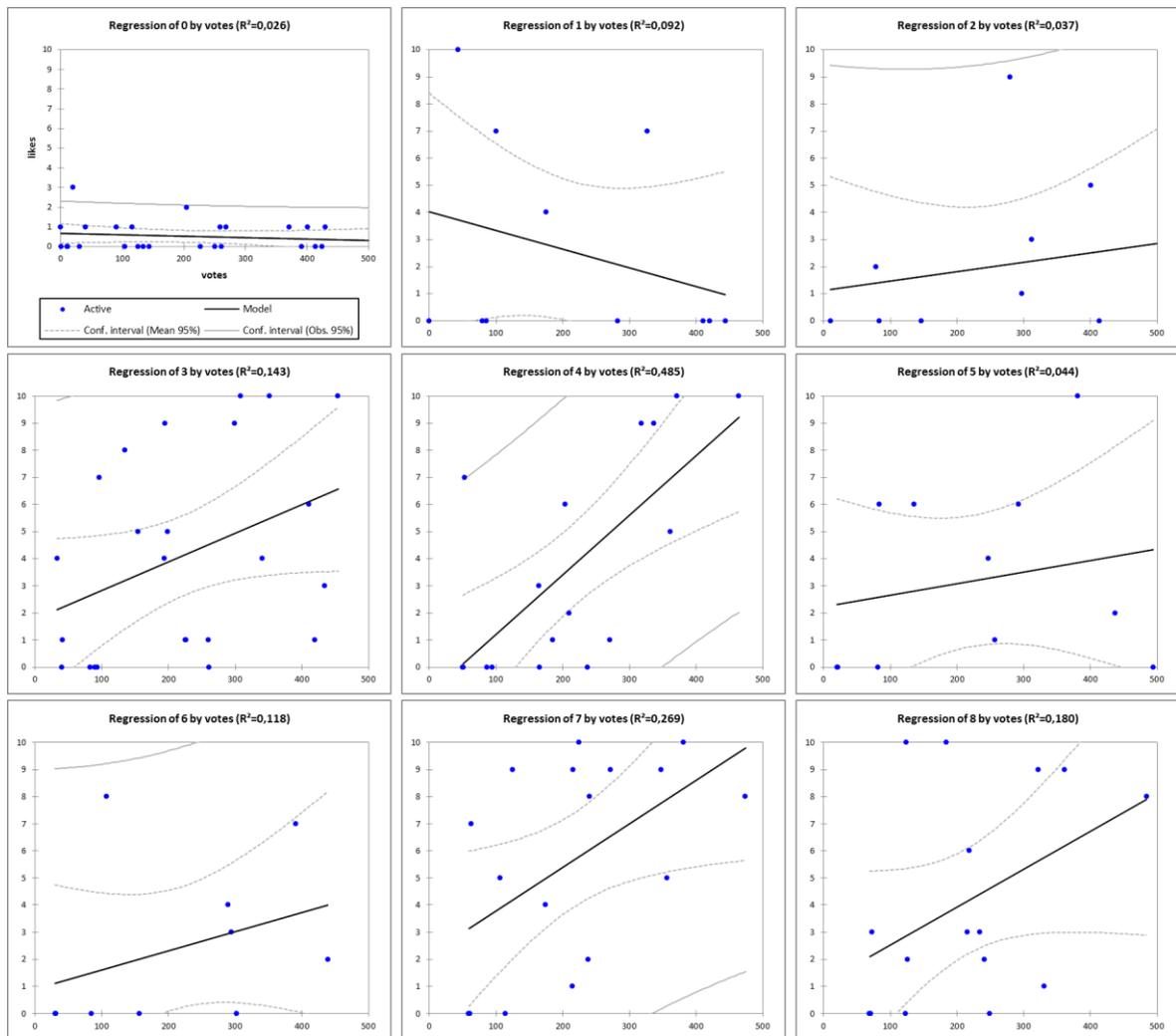


Figure 31: Individual linear regression graphs over **specific subject** user data; incl. 95% confidence intervals

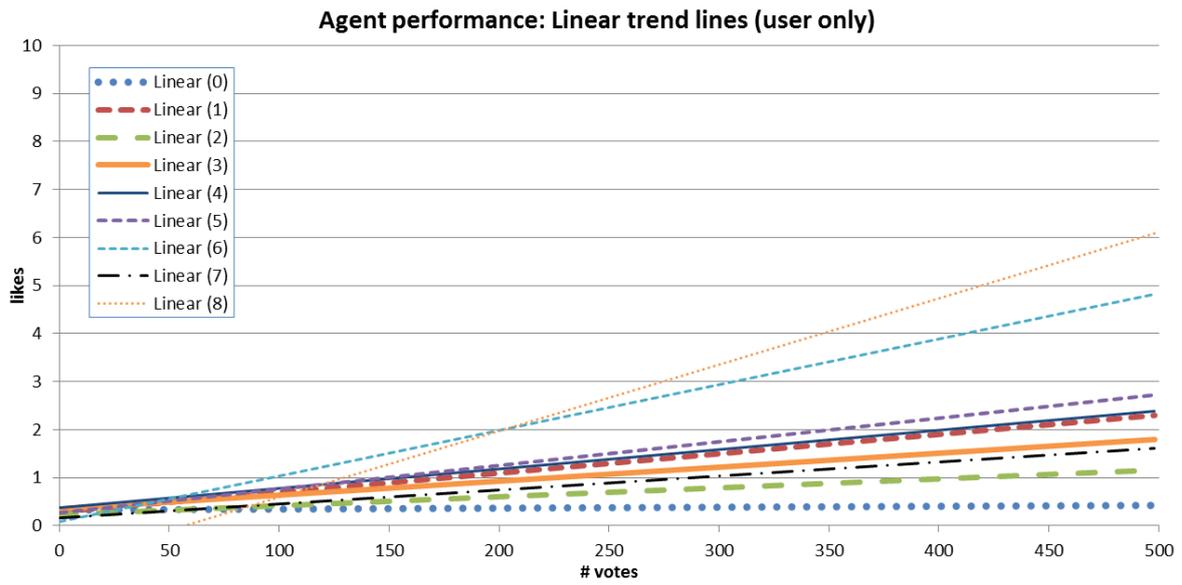


Figure 32: Overview of linear trend lines for each agent configuration over **normal** user data

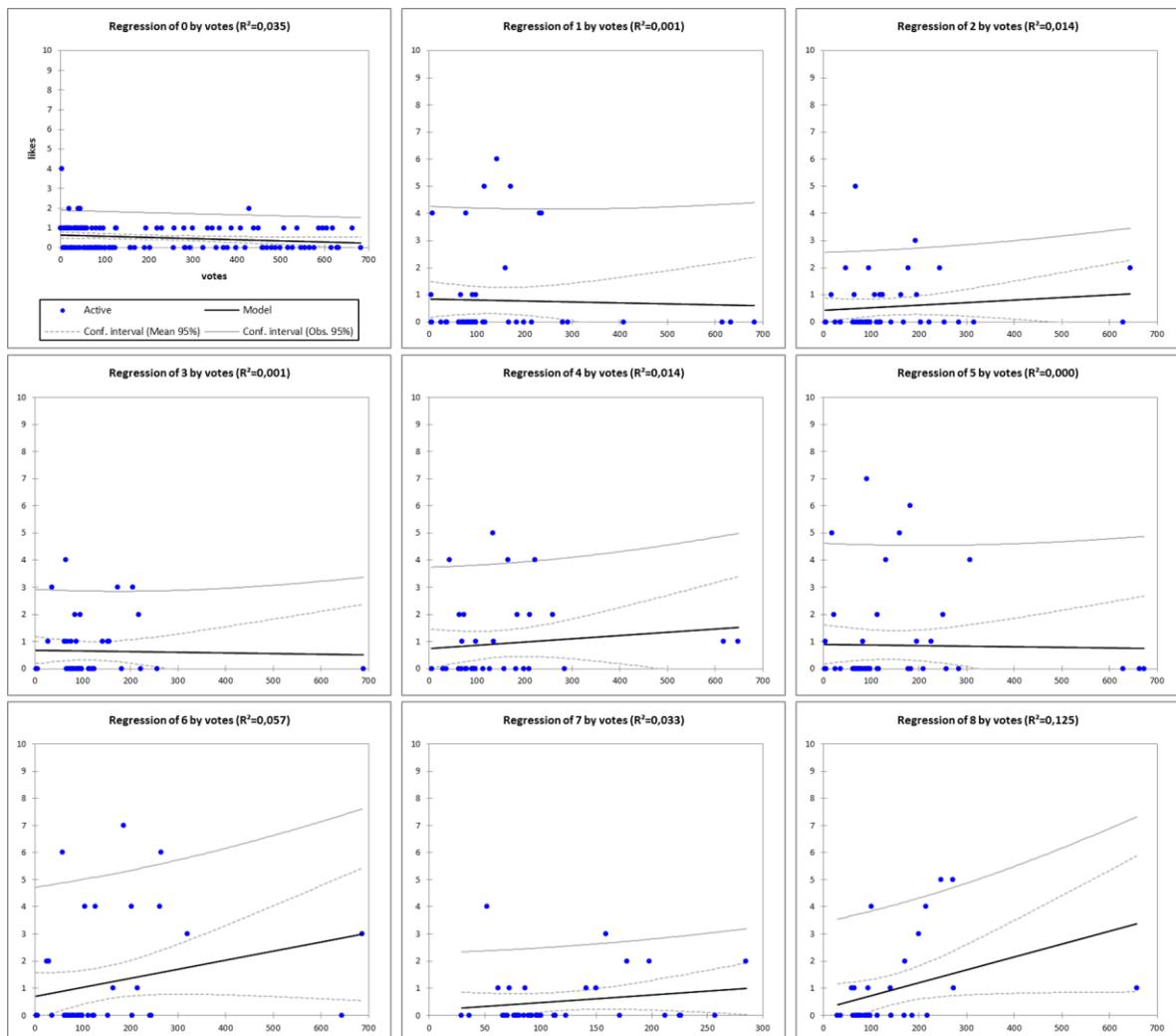


Figure 33: Individual linear regression graphs over **normal** user data; incl. 95% confidence intervals

From the above regression graphs on the previous pages can be observed:

1. Lots of linear models are based on extremely low  $R^2$  values (close to zero). Worst values are observed for the regression on the normal users data (Figure 33).
2. The 'best' configurations over all users (Figure 28) are:
  1. 7 - implicit long-term Bayes ( $R^2=.360$ )
  2. 4 - explicit short-term Bayes ( $R^2=.182$ )
  3. 3 - explicit long-term Bayes ( $R^2=.159$ )
3. The 'best' configurations over specific subject user data (Figure 30) are:
  1. 7 - implicit long-term Bayes ( $R^2=.269$ )
  2. 4 - explicit short-term Bayes ( $R^2=.485$ )
  3. 8 - implicit short-term Bayes ( $R^2=.180$ )
  4. 3 - explicit long-term Bayes ( $R^2=.143$ )
4. The 'best' configurations over the normal user data (Figure 32) are:
  1. 8 - implicit short-term Bayes ( $R^2=.180$ )
  2. 6 - implicit short-term SVM ( $R^2=.057$ )
5. The 95% confidence intervals of normal users (Figure 33) are relatively small compared to the large intervals for specific subject user data (Figure 31).
6. One extreme negative trend is noticed for configuration 1: Explicit long-term SVM in Figure 31. ( $R^2=.092$ ; 11 data points)
7. In Figure 33 a few final data points perhaps have a huge effect on the trend lines for each agent config ( $\pm 3$  points at 500-700 votes range).
8. The time effect of short-term agents (2, 4, 6 & 8) is not visible. Larger fluctuations are expected over time (different days), because of the small feedback timespan. This will be further analysed in paragraph 4.2.8.

From points 2-4 can be concluded the Bayes agents finally outperformed the SVM configurations, thereby nothing can be said about the influence of the memory and feedback models from this analysis.

For each agent configuration also a default ANOVA F-test is performed to validate the significance of the linear regression model on the data. The exact results are not displayed, but most (except some random) are not significant. This means statistically most of the data cannot be explained by the linear regression models. This, together with many of the above observations, raises the question if the linear least squares model is the best possible reflection of the data. Maybe more advanced models (i.e. higher order, logarithmic, Theil's regression, etc.) can improve analysis. These tests are left for future research.

Another point for discussion is the total absence of measurements for the effect of voting on personal RSS feeds, which is completely omitted so far. This can probably explain the negative trend from point 6 above. Especially if a user votes positively for a lot of personal feeds at the beginning of a trial. Then a trained classifier can start with interesting recommendations and perform worse over time because of less consistent votes. This effect can also have significant influence on all given trend lines.

So, this raises the question: Is the experimental setup valid? The answer to this question is twofold. When the whole system is analyzed and only a global indication of performance is needed, the answer is 'yes'. But a more advanced experiment (i.e. equal time spans, less variables, etc.) needs to be conducted when trying to explain the exact performance and influence of individual system components over multiple users.

#### 4.2.6 Naïve Bayes vs. Support Vector Machine

The above analysis on combined user data cannot explain the influences on performance for each variable in the agent configurations. Therefore the following paragraphs also include selective results from individual users, to analyze some of the relations and differences between the configurations.

The difference in performance between Bayes (3, 4, 7 & 8) and SVM (1, 2, 5 & 6) configurations is the only measured effect from the results in the previous paragraph. The general observation is Bayes outperforms SVM. Furthermore voting distributions between user groups (Figure 27) suggested the opposite for normal users. The voting graphs for one of the users (user 1) are displayed below. From this can be seen the Bayes recommenders received much less votes (both positive and negative); this could explain the suggested difference.

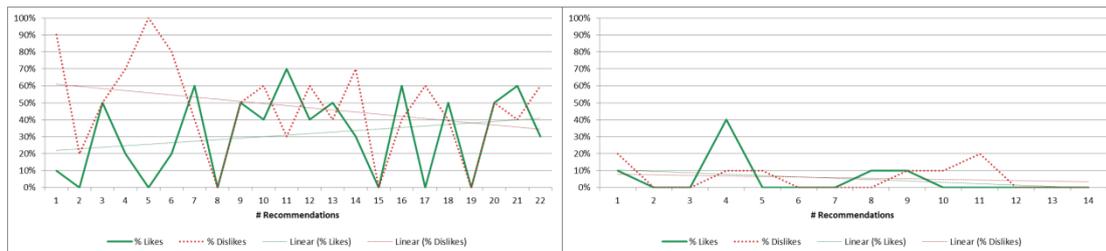


Figure 34: Voting graphs for user 1; left: SVM; right: Bayes

#### 4.2.7 Implicit vs. explicit

Much less attention is given to the difference between recommendation based on implicit (5-8; including reading time) and explicit (1-4; votes only) feedback. In the graphs below, this difference is displayed for user 3.

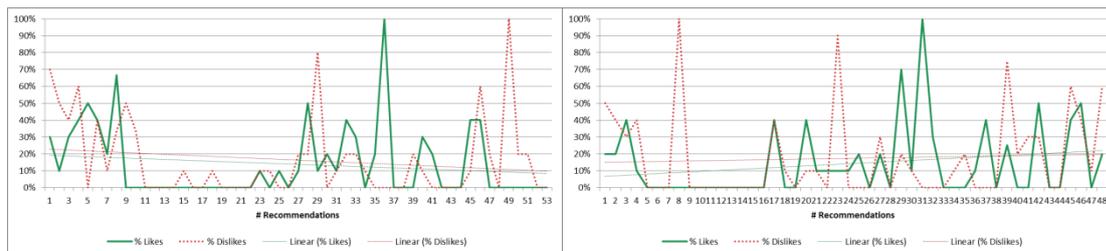


Figure 35: Voting graphs user 3; left: explicit; right: implicit

No clear difference can be noticed between these graphs and this is the same for the other users. The reason is the extensive voting. In the first place the users for the specific subjects vote on almost all articles, so there is no difference between implicit and explicit recommendations. Therefore these users are omitted for this case. Also the normal users were hinted to vote a lot. So from this experiment, nothing can be said about the difference between implicit and explicit votes. To overcome this shortcoming, some future work is proposed in paragraph 5.3.5.

#### 4.2.8 Short-time vs. long-time

Also differences between short-term (even; 24h feedback) and long-term (odd; 365 days feedback) are not clearly observed from the extensive analysis in paragraph 4.2.5. The graphs below show the voting difference on short- and long-term recommendations for user 3.

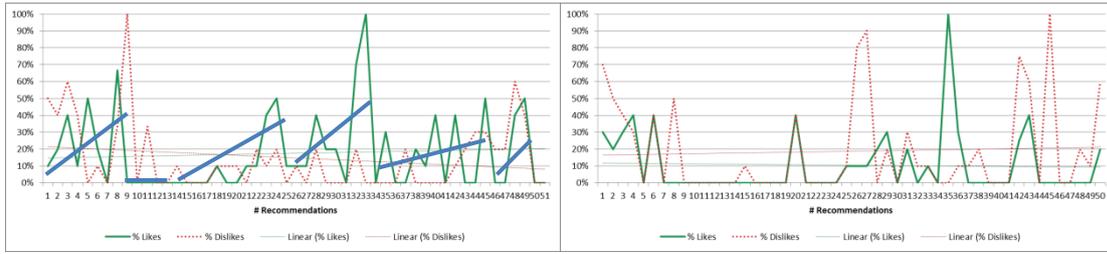


Figure 36: Voting graphs user 3; left: short-term; right: long-term

The expected effect of a drop in performance for the short-term memory agents, when reusing the system another day, is displayed in the left graph. Each bold (blue) line estimates the short-term performance for a single day. Most blue lines show a positive trend, which cannot be seen from the global linear model (dotted green). On the other hand, unfortunately for this user the long-term model (right) does not show an increasing trend as expected. This short-term effect can also be observed when analyzing the data of the other users.

To further analyze if the recommender type has an effect on the above, the same data can be split into Bayes and SVM recommender data. The corresponding graphs are displayed below:

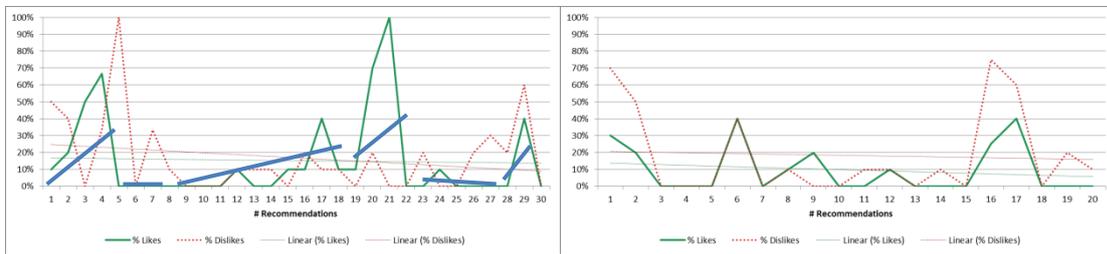


Figure 37: Voting graphs user 3; left: SVM short-term; right: SVM long-term

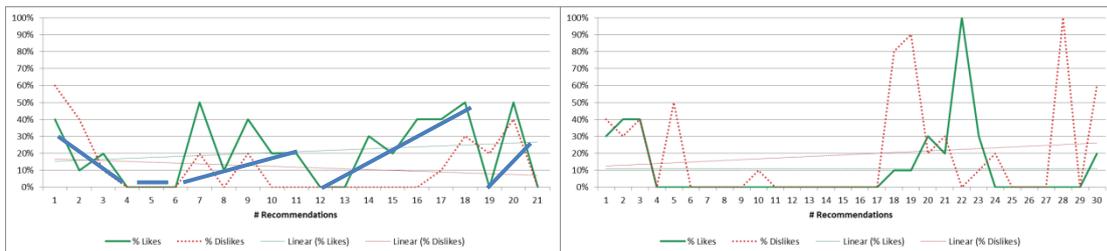


Figure 38: Voting graphs user 3; left: Bayes short-term; right: Bayes long-term

Nothing extra can be observed from the long-term models (right). But also no noticeable difference in performance exists between the SVM and Bayes short-term models within the data of this individual (left).

If the same depth of analysis is performed for the other users, the sampling bias becomes a problem. The above graphs all contain 20+ recommendations, but data for the other users at this level of detail mostly contain much less measurements, or each day contains just one or none of the data points. Therefore also no conclusions can be drawn from the other users at this level.

So finally, a positive effect of the short-term memory models is clearly made visible, which is not observed from the measurements in paragraph 4.2.5. But the effect of the classification algorithms on these memory models is unknown.

### 4.2.9 Specific interests

To test the system performance on single interest training, some users only voted for one pre-defined specific subject. The best results are found for the 'sports' user, who voted 'like' for all sport subjects and 'dislike' for everything else. The final performance graphs of this user are displayed in Figure 39.

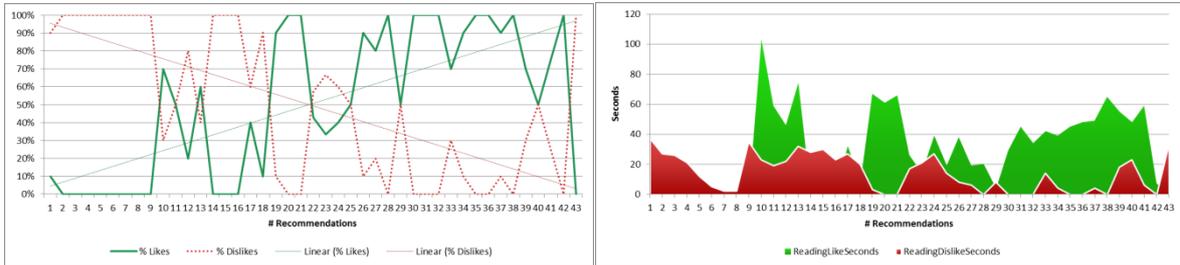


Figure 39: 'Sports' user graphs; left: voting over time (excl. random); right: corresponding article reading time

These above graphs present the close to perfect performance models. The positive votes steadily increase over time and the opposite goes for the negative votes. Also the increase in positive reading time for this user is clearly visible, even when taken into account most of the time this user probably barely actually read the articles because of the assignment. Finally some illustrations are given of the actual working of the system, displayed by the screenshots of some successful recommendations:



Figure 40: Apple user recommendation example

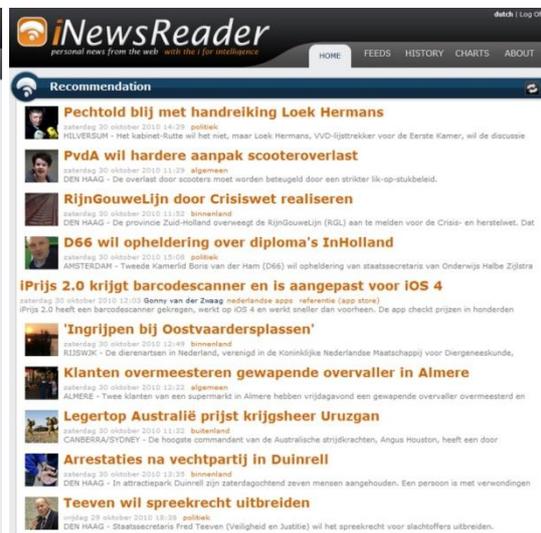


Figure 41: Dutch user recommendation example

### 4.2.10 Summary

The first observation was the existence of a sampling bias due the small amount of subjects. Next was shown that the amount of opened articles over time is not related to the success of the articles selection methods. Thereafter the positive effect of voting on the recommendation performance was shown. But in normal conditions this effect will be less visible because of the influence of the extensive voting instruction. The combined results over multiple users and user groups for each agent configuration were analyzed next. The disappointing outcome was that no significant results were found and the data model was questioned. Finally the results showed better performance of the Bayes compared to the SVM agents. Further analysis on individual users gave more positive results.

# 5 Conclusion

## 5.1 Conclusion

The goal of this research is to handle the problem of *information overflow* by implementing a personalized adaptive netnews recommender system. The main research question asks how to achieve this goal. The answer from literature is to combine multiple techniques to achieve a successful result. Therefore a recommender framework based on multi-agent technologies is developed. From an implementation of this system is concluded it is a working and promising setup. Not all individual framework components are fully optimized and tested on performance, but a small experiment on the final system output showed an overall positive effect on learning and recommendation performance.

By measuring both explicit user feedback and implicit reading time, a more detailed answer was sought for the influences and performance of the individual recommender components and agent configurations. The main conclusion is that the Naïve Bayes agents outperformed the Support Vector machine configurations. Also large differences were found between individual users and user groups. Unfortunately no significant conclusions could be formed from the extensive analysis of the combined results of these users for each configuration. By analyzing results of individual users, a positive influence of the short-term memory is shown. A sampling bias prevented a further in depth analysis of inter-related influences. Also nothing can be concluded for the difference between implicit and explicit feedback from the fact that users were instructed to vote extensively.

The final conclusion is the overall results are positive, but there are some remarks on the conducted experiment. More advanced research is needed to form significant conclusions about the influences of the individual system components. Furthermore additions and improvements on many parts of the implemented system can probably elevate the recommendation performance enormously. This will be discussed in the next sections.

## 5.2 Discussion

### 5.2.1 Crawler implementation

The initial version of the crawler was built to only harvest information from the RSS feeds and apply default and widely used pre-processing techniques to extract the features. As mentioned before, the RSS data was often incomplete and too limited for the recommendation process. Therefore the crawler has been extended during research to include the articles' web content.

At forehand no extended experiments were planned to test the crawling performance, because it would be a default component using known working procedures and pre structured RSS data. Therefore influence of the crawler is underestimated from the beginning. More extended research should be conducted on the crawling methods, the crawling performance itself and the influence on the recommendation algorithms.

The implemented crawling procedure is internally tested using a small graphical user interface (Figure 42). The results of the full parsing of an article are displayed in a web interface as given in Figure 12. CSS borders are used to display the selected DOM nodes. These results are subject to user interpretation.

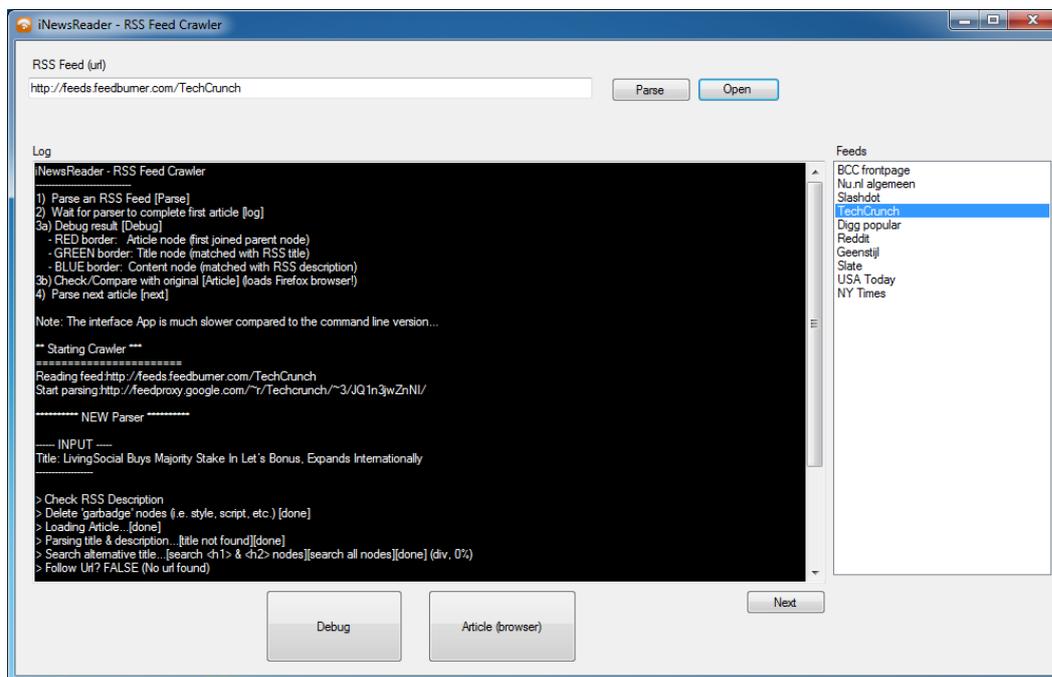


Figure 42: Crawling test interface

### 5.2.2 Performance

Because of the limited system resources, the experiment only measured the system usage and performance of the components using a small user base. To get more reliable results, be able to generalize and to exactly measure the influences of each component, more advanced research on a large user group is needed. Handling a massive user base is a challenge on its own, both in getting the users as providing the system resources.

The reason behind this resource problem is the simultaneous execution of many agents. For each system user at each home-page refresh two types of agents are executed: Crawler agents to harvest the contents of the personal RSS feeds and recommender agents to check for available recommendations. So far this only takes milliseconds. But processing power is requested when the crawler (one for each rss feed) finds new articles and needs to start harvesting and processing (feature extraction). This can be optimized in many ways, for instance by executing one agent working on a queue of rss feeds eventual on an external machine. But most resources are consumed when a recommender agent starts classifying. First the classifier (Naïve Bayes or SVM) is trained. When a large amount of feedback articles is available this process can take up to some minutes of full processing power. Thereafter approximately 100-1000 articles per agent per second are processed for classification. Some agents needed >100.000 articles for a single recommendation. A possible optimization is to use incremental training algorithms for the classifiers.

### 5.2.3 Multiple interests

The current version of the recommender framework only trains the recommenders by using user feedback data of all user displayed articles. Therefore the final recommendation contains articles on all areas of interest. Users mostly are interested in multiple topics, but during system usage, they often only want to read articles from one topic at a time. The results from the conducted experiment thereby also indicated a better system performance when trained on a specific pre-defined subject. Therefore the introduction of a topic selection or some kind of 'mood' filter agent probably improves the usage and final recommendation of the system vastly. More research on this subject is advised.

### 5.2.4 Privacy

During usage of the system, a lot of information of each subject is stored in the user profile. Within this project, this information is only used for research purposes and kept anonymous. But for commercial recommender systems, these profiles can be valuable and profitable for web vendors. Personalized content is highly valued by online users, but not against all prizes. Examples of misuse of this personal information are spamming and advertising purposes. Many computer users are concerned about their privacy on the internet and therefore this subject is a hot topic in recent debates. The collection of personal data is also subject to legal regulations in many countries and states. Approaches to reconcile both privacy and personalization are presented in [40].

### 5.2.5 Future of RSS

The structure of the internet is evolving continuously and thereby the question arises: Will RSS survive? By introduction of the term 'semantic web' or 'web 3.0' a few years ago, some argue the usage of RSS will become superfluous. If the web itself contains semantic structure, RSS won't necessarily add any valuable information. The introduction of the corresponding techniques (i.e. HTML5, RDF, Scheme, OML, etc.) prosecutes gradually; at moment of writing the first internet browsers supporting HTML5 are already introduced.

On the other side, the popularity and usage of RSS feeds for personal news aggregation is still increasing. The figure below (Figure 43) displays two graphs with usage information from an online popular RSS reader (Google News Reader [30]). The source and content of the graphs are questionable, but the usage trend is still clearly visible.

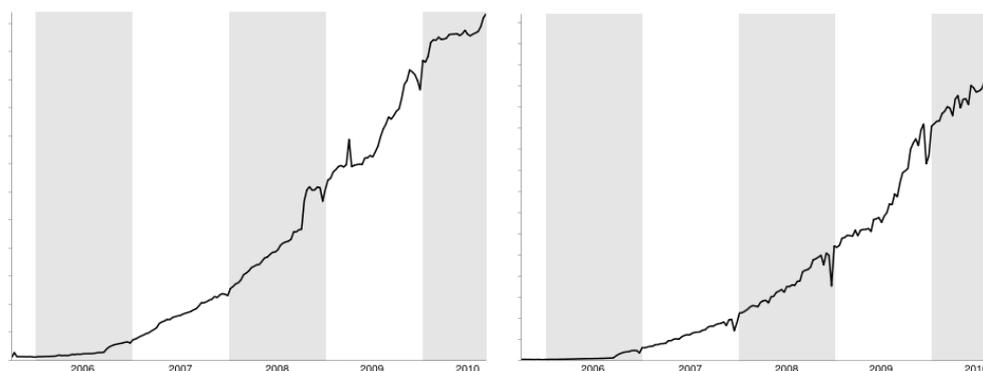


Figure 43: Left: RSS reader usage; Right: Number of items read daily;  
Source: <http://googlereader.blogspot.com/2010/09/welcome-and-look-back.html>

## 5.3 Future Work

The methods as proposed in this paper are not optimally developed and far from complete. Also the conducted experiment could be extended to gather more reliable data. To improve recommendation and further study this subject, the following paragraphs propose multiple directions for improvement of the system.

### 5.3.1 Advanced Agent Framework

As mentioned before (section 3.3), the most influencing design choice on the implementation of the proposed recommender framework (section 3.2) is the reduced agent management module (AMS). Thereby one of the most interesting properties of multi-agent systems is excluded; the property of *self-organization*. By implementing an advanced agent framework (i.e. JADE [34] or AgentService [6], see also FIPA [24]), the advantages of multi-agent technology can be exploited. In Figure 44 the framework model from AgentService [6] is given.

This model includes components equal to the described modules in the proposed recommender framework (i.e. AMS, Messaging, Logging, etc.). Other components (i.e. White/Yellow pages, Mobility, Load balancing, etc.) can possibly greatly improve system performance by taking advantage of distributed system control. The details of these frameworks are left for the reader, but this example model demonstrates the possibilities of extension towards such a multi-agent architecture and therefore further research on this subject is greatly advised.

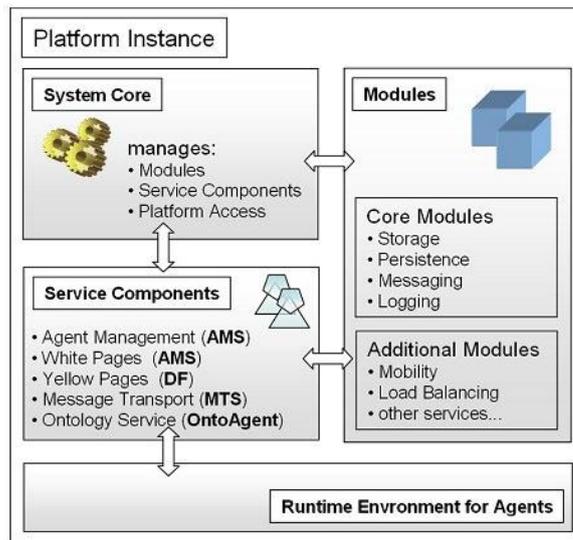


Figure 44: AgentService [6] framework model

### 5.3.2 Other agents

For this research a small amount of agents, targeting content-based recommendation, are implemented (sections 3.7 and 3.8). A lot more agents could be added to the system and contribute to the final recommended list of articles. The following paragraphs describe additional examples of possible agents. Of course even a lot more agents can be thought of.

#### 5.3.2.1 Collaborative agent

The current research in this paper is mainly targeted towards content-based recommendation for reasons described. As explained in paragraph 2.1.5, improved results can be obtained by including collaborative algorithms in the recommendation process. Therefore a collaborative agent could be added to the system. The details of these collaborative models are outside the scope of this project, but some brief information is given below.

A *collaborative agent* recommends articles based on items marked interesting from users ( $j$ ) with similar behavior. Generally a user-item ( $x_{jn}$ ) similarity matrix  $A$  based on a *Pearson correlation coefficient* (Formula 15) is used to predict the active ( $a$ ) users' interests  $I$  for an article  $n$  using Formula 16.

$$w(a, j) = \frac{\sum_{n \in I_a \cap I_j} (x_{an} - \bar{x}_a)(x_{jn} - \bar{x}_j)}{\sqrt{\sum_{n \in I_a \cap I_j} (x_{an} - \bar{x}_a)^2 \sum_{n \in I_a \cap I_j} (x_{jn} - \bar{x}_j)^2}}$$

Formula 15: Pearson correlation coefficient

$$p_{an} = \bar{x}_a + \frac{\sum_{j=1}^M w(a, j)(x_{jn} - \bar{x}_j)}{\sum_{j=1}^M w(a, j)}$$

Formula 16: Active user  $a$  interest indicator for article  $n$

In [63] Xia et al. present a model for collaborative filtering using *Support Vector Machines*. Missing elements within the similarity matrix will be iteratively updated using trained SVM classifiers for each unknown user-item pair  $x_{an} \in A$ . The input for each SVM at training will be the set of feature vectors from all neighbor users' rated items  $x_{jn}$ . The final recommendation is based on a list of top  $n$  rated items not in  $I_a$ .

### 5.3.2.2 Additional recommender agents

Besides Naïve Bayes and Support Vector Machines, more machine learning techniques are suitable for the classification task of text recommendation. Some are already mentioned (i.e. k-NN; paragraph 2.3.4) other examples are *Artificial Neural Networks* (ANN), *Rocchio classifiers* or *decision tree learning* models (i.e. ID3). Recommendation can probably be improved by adding more of these techniques to the recommendation module and combine them with the methods described in this paper. Also additional filter techniques (i.e. demographic; paragraph 2.1.3) possibly increase recommendation performance by including more data in the recommendation process.

### 5.3.2.3 Known agent

News generally spreads fast over multiple news sources in a very short period of time. If an article is voted interesting, this often results in the displaying of the exact same 'news' just from another source. Therefore a filter of already 'known' articles could increase the usability of the system by excluding these articles from further recommendations. This could for example be accomplished by excluding articles with a high cosine similarity towards liked articles (see also paragraph 2.3.2).

### 5.3.2.4 Tagging & categories agents

Many RSS feeds also include a list of categories the feed and articles belong to. The current recommenders only make use of the word counts and feature vectors with tf-idf values from the articles content within their recommendation algorithms. Including these pre assigned categories can probably also improve recommendation. Going one step further, users could be given the possibility to tag articles within the system (provide their own categories). This way the users' idea about the content is registered (what they think is interesting about an article) and thereby these articles can next be used to train for directed recommendation.

#### 5.3.2.5 User-defined interests agent

The current recommenders automatically create a user model based on their implicit and explicit system behavior. Another way of creating this model is by just asking the user. Users could for example be asked to select their interest from a predefined list of categories or to provide a list of tags indicating their interests. Next a *static interest agent* can use the top  $n$  features (i.e. tf-idf weights) from the sum of word vectors from the articles stored in each category of interest for recommendation. This furthermore can be extended by including a weight (i.e. a slider in the interface) for each category, which can be adjusted by the user to increase/decrease the level of interest for a specific subject.

#### 5.3.2.6 3<sup>rd</sup> party agents

Some online 3<sup>rd</sup> party services in the field of RSS feeds and personal recommendation are already available (i.e. PostRank [50]). Some of these services also provide an Application Programming Interface (API). This API can be used to interact with these services and use their 'knowledge' within the recommendation process. By interacting with 3<sup>rd</sup> party services, large tested and globally accepted sources can be accessed for news aggregation and recommendation purposes.

### 5.3.3 Clustering & dimensionality reduction

Most of the above proposed future work is targeted to the final tasks of the recommendation process; the article classification and filtering methods. But improving recommendation probably starts with improving and structuring resources. The general saying "*garbage in garbage out*" also applies to the field of recommender systems. In document classification tasks, the dimensionality of the data can explode to extremely high dimensions (the size of  $10^4$  and larger). Therefore structuring the input space (item-server) can improve both recommendation speed (computational costs) and accuracy enormously.

One method of structuring is *clustering* of the data (see also paragraph 2.1.7). Thereby the Open Directory Project (ODP) [46], a data structure for relational information, can possibly provide a central role. Another approach is *term reduction*. As the dimensions of the data (number of terms) grow, also the risk of *overfitting* increases. The consequences of overfitting are learners overweighting features which are of less significance for the class of the data. See [55] for an extensive comparison of multiple approaches to term space reduction.

### 5.3.4 Feed priorities

In the current system, all feeds and articles are handled with equal priorities. During harvesting the selection of the next feed to harvest is solely based on the latest harvesting date. At recommendation, articles most recently added to the item-server are selected first. When both articles and feeds are given priorities (i.e. by using PostRank [50] or popularity metrics within the community; [21] or [51]), these selection procedures can be improved by returning high quality and popular content first. Also many RSS feeds contain a skipDays or skipHours element to hint for the next moment to visit the feed for new content. Using these values and by using more advanced revisit policies the system resources used for harvesting can be distributed more accurately. This way the overall quality of the recommender system can increase by providing more recent and high quality content.

### **5.3.5 Implicit feedback**

The implemented implicit feedback model used by the recommender agents for article selection, as described in paragraph 3.7.4, is solely based on the time spent reading an article. From [17] and [25], many more implicit feedback indicators can be used to create an accurate model of the user's interests (see also paragraph 2.1.2). By using more advanced measurements, probably the effort users need to put into the training of the recommender can be minimized. One of these measurements to be considered is the negative effect of not opening a recommended article. In the current research this information is omitted, but the recommender system probably can much more automatically adapt to the user and much less rely on the users' voting strategies if these more advanced implicit feedback indicators are included.

## 6 References

- [1] ACM Conferences on Recommender Systems. [Online]. <http://recsys.acm.org/>
- [2] J.M. Adams, P.N. Bennet, and A. Tomasic, "Combining Personalized Agents to Improve Content-Based Recommendations," Language Technologies Institute, Carnegie Mellon University, Pittsburgh, CMU-LTI-07-015, 2007.
- [3] G. Adomavicius, R. Sankaranarayanan, R. Sen, and A. Tuzhilin, "Incorporating contextual information in recommender systems using a multidimensional approach," *ACM Transactions on Information Systems (TOIS)*, vol. 23, no. 1, pp. 103-145, January 2005.
- [4] G. Adomavicius and A. Tuzhilin, "Extending Recommender Systems: A Multidimensional Approach," in *International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.
- [25] G. Adomavicius and A. Tuzhilin, "Personalization Technologies: A Process-Oriented Perspective," *Communications of the ACM*, vol. 48, no. 10, October 2005.
- [5] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *Transactions on knowledge and data engineering*, vol. 17, no. 6, June 2005.
- [6] AgentService Framework. [Online]. <http://www.agentservice.it>
- [7] Amazon. [Online]. <http://www.amazon.com>
- [8] M. Balabanovic and Y. Shoham, "FAB: Content-Based, Collaborative Recommendation," *Communications of the ACM*, vol. 40, no. 3, March 1997.
- [9] D. Billsus and M.J. Pazzani, "User Modeling for Adaptive News Access," *User Modeling and User-Adapted Interaction*, vol. 10, no. 2-3, June 2000.
- [10] A. Birukov, E. Blanzieri, and P. Giorgini, "Implicit: An Agent-Based Recommendation System for Web Search," in *Autonomous agents and multi-agent systems (AAMAS '05)*, Utrecht, Netherlands, July, 2005, pp. 25-29.
- [11] British Broadcasting Corporation (BBC). [Online]. <http://www.bbc.com>
- [12] Christopher J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121-167, 1998.
- [13] R. Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction*, no. 12, pp. 331-370, 2002.
- [14] R. Burke, "Hybrid Web Recommender Systems," in *The Adaptive Web*. Berlin, Heidelberg: Springer-Verlag, 2007, ch. 12, pp. 377-408.
- [15] Cable News Network (CNN). [Online]. <http://www.cnn.com>
- [16] C.-C. Chang and V. Vapnik. (1992) LIBSVM: a library for support vector machines. [Online]. <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- [17] M. Claypool, P. Le, M. Wased, and D. Brown, "Implicit Interest Indicators," in *Intelligent User Interfaces (UIU '01)*, Santa Fe, New Mexico, USA, Januari 14-17, 2001.
- [18] Viv Cothey, "Web-crawling reliability," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 14, pp. 1228-1238, December 2004.
- [19] A. Das, M. Datar, and A. Garg, "Google News Personalization: Scalable Online Collaborative Filtering," in *World Wide Web (IW3C2)*, Banff, Alberta, Canada, 2007.
- [20] B. David, R. Conejo, and A.A. David, "METIOREW: An Object Oriented Content Based and Collaborative Recommending System," *Lecture Notes in Computer Science*, vol. 2266, p. 310, 2002.

- [21] DIGG. [Online]. <http://www.digg.com>
- [22] Ensembl. [Online]. <http://ensembl.com/>
- [23] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine*, vol. 17, pp. 37-54, 1996.
- [24] (1999) Foundation for Intelligent Physical Agents (FIPA). Specifications. [Online]. <http://www.fipa.org>
- [26] C. Gena and S. Wiebelzahl, "Usability Engineering for the Adaptive Web," in *The Adaptive Web.*, 2007, ch. 24, pp. 720-762.
- [27] Genio. [Online]. <http://www.genio.com>
- [28] N. Good, J.B. Schafer, J. Konstan, A. Borchers, and B. Sarwar, "Combining Collaborative Filtering with Personal Agents for Better Recommendations," in *Proceedings of Sixteenth National Conference on Artificial Intelligence (AAAI)*, 1999, <http://www-users.cs.umn.edu/~herlocke/aaai-99.pdf>.
- [29] Google news. [Online]. <http://news.google.com>
- [30] Google reader. [Online]. <http://reader.google.com>
- [31] GROUPLens research group. [Online]. <http://www.grouplens.org>
- [32] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, "A Practical Guide to Support Vector Classification," Department of Computer Science, National Taiwan University, Taipei, 2009.
- [33] Internet Movie Database (IMDB). [Online]. <http://www.imdb.com>
- [34] Jave Agent DEvelopment Framework (JADE). [Online]. <http://jade.tilab.com/>
- [35] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," in *Proceedings of the European Conference on Machine Learning*, Berlin, 1998, pp. 137-142.
- [36] T. Joachims, "Transductive Inference for Text Classification using Support Vector Machines," in *Proceedings of ICML-99, 16th International Conference on Machine Learning*, 1999, pp. 200-209.
- [37] T. Joachims, D. Freitag, and T. Mitchell, "WebWatcher: A Tour Guide for the World Wide Web," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
- [38] M. A. Johnson. (2009, September) SVM.NET. [Online]. <http://www.matthewajohnson.org/software/svm.html>
- [39] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604-632, Sept 1999.
- [40] A. Kobsa, "Privacy-Enhanced Web Personalization," in *The adaptive web*. Berlin: Springer, 2007, ch. 21, pp. 628-670.
- [41] last.fm. [Online]. <http://www.last.fm>
- [42] Y. Li, L. Lu, and L. Xuefeng, "A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in E-Commerce," *Expert Systems with Applications*, vol. 28, no. 1, pp. 67-77, January 2005.
- [43] J. Martinez. (2010, Feb 16) nBayes. [Online]. <http://nbayes.codeplex.com/>
- [44] T. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, inc., 1997.
- [45] H. Nwana, "Software Agents: An Overview," *Knowledge Engineering Review*, vol. 11, no. 3, pp. 1-40, 1996.
- [46] Open Directory Project. [Online]. <http://www.dmoz.org>

- [47] Larry Page, Sergey Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Library Technologies Project, Technical report, <http://dbpubs.stanford.edu/pub/1999-66>, 1998.
- [48] M. J. Pazzani and D. Billsus, "Content-Based Recommendation Systems," in *The Adaptive Web.*: Springer Berlin / Heidelberg, 2007, ch. 10, pp. 325-341.
- [49] M.F. Porter, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 40, no. 3, pp. 211-218, 2006.
- [50] PostRank. [Online]. <http://www.postrank.com/>
- [51] reddit. [Online]. <http://www.reddit.com>
- [52] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," 1994.
- [53] P. Resnick and H.R. Varian, "Recommender Systems," *Communications of the ACM*, vol. 40, no. 3, March 1997.
- [54] G. Salton and M. J. McGill, *Introduction to modern information retrieval.*, 1983.
- [55] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM computing surveys*, vol. 34, no. 1, pp. 1-47, 2002.
- [56] A. Shepitsen, J. Gemmel, B. Mobasher, and R. Burke, "Personalized Recommendation in Social Tagging Systems Using Hierarchical Clustering," in *Recommender Systems 2008 (RecSys '08)*, Luasanne, Switzerland, October 23-25, 2008.
- [57] showfilter. [Online]. <http://www.showfilter.com/>
- [58] A. Singhal, C. Buckley, and M. Mitra, "Pivoted Document Length Normalization," in *ACM SIGIR conference on Research and development in information retrieval*, 1996.
- [59] TechCrunch. [Online]. <http://www.techcrunch.com/>
- [60] S. Tong and D. Koller, "Support Vector Machine Active Learning with Applications to Text Classification," *Journal of Machine Learning Research*, pp. 45-66, November 2001.
- [61] V. N. Vapnik, *The nature of statistical learning theory*. New York: Springer, 1995.
- [62] M. Wooldridge, *An Introduction to MultiAgent Systems.*: Wiley, 2005, ISBN: 0-471-49691-X.
- [63] Z. Xia, Y. Dong, and G. Xing, "Support vector machines for collaborative filtering," in *Proceedings of the 44th annual Southeast regional conference*, Melbourne, Florida, 2006, pp. 169-174.

# 7 Appendix

## 7.1 Original system proposals

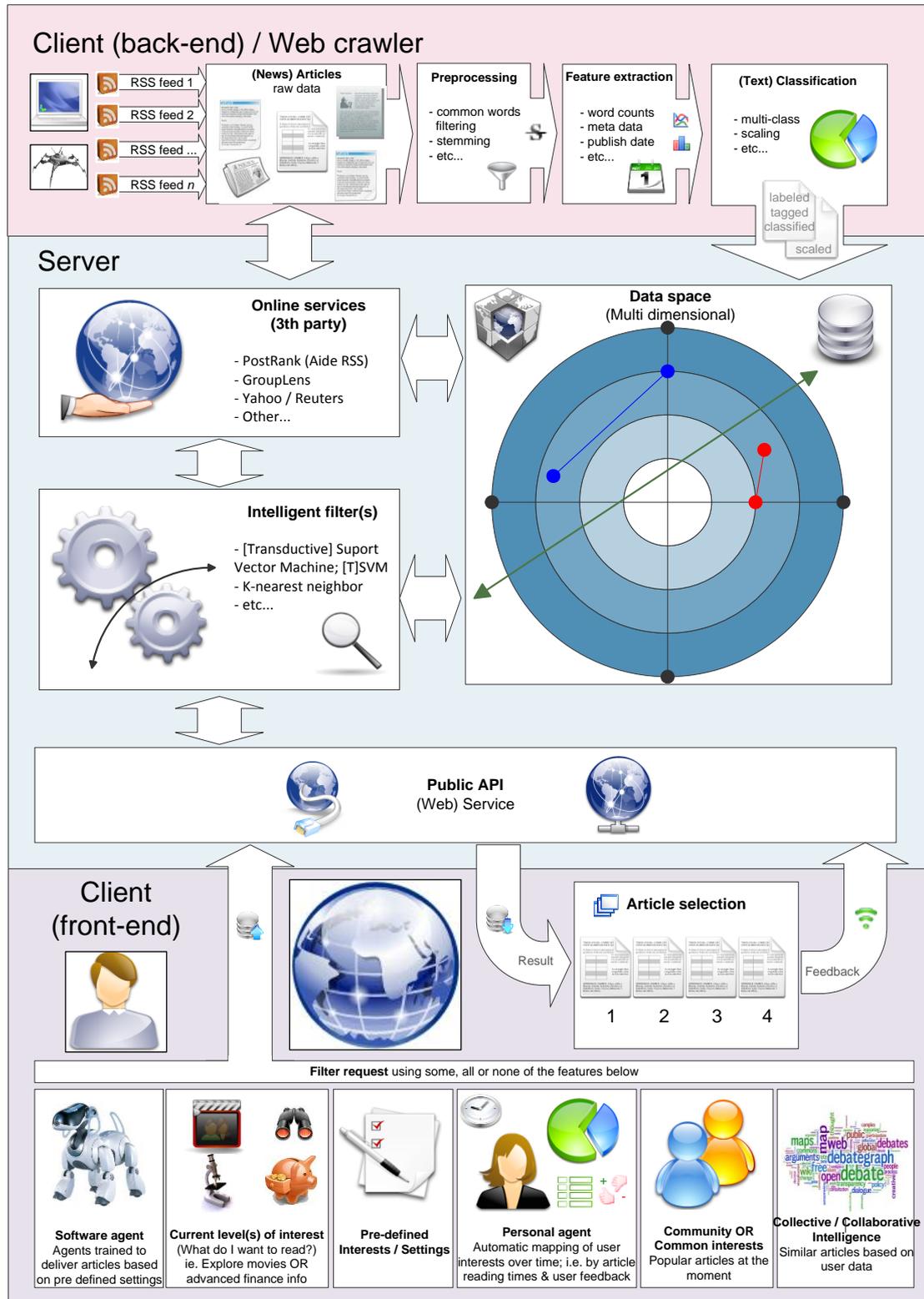


Figure 45: First framework proposal

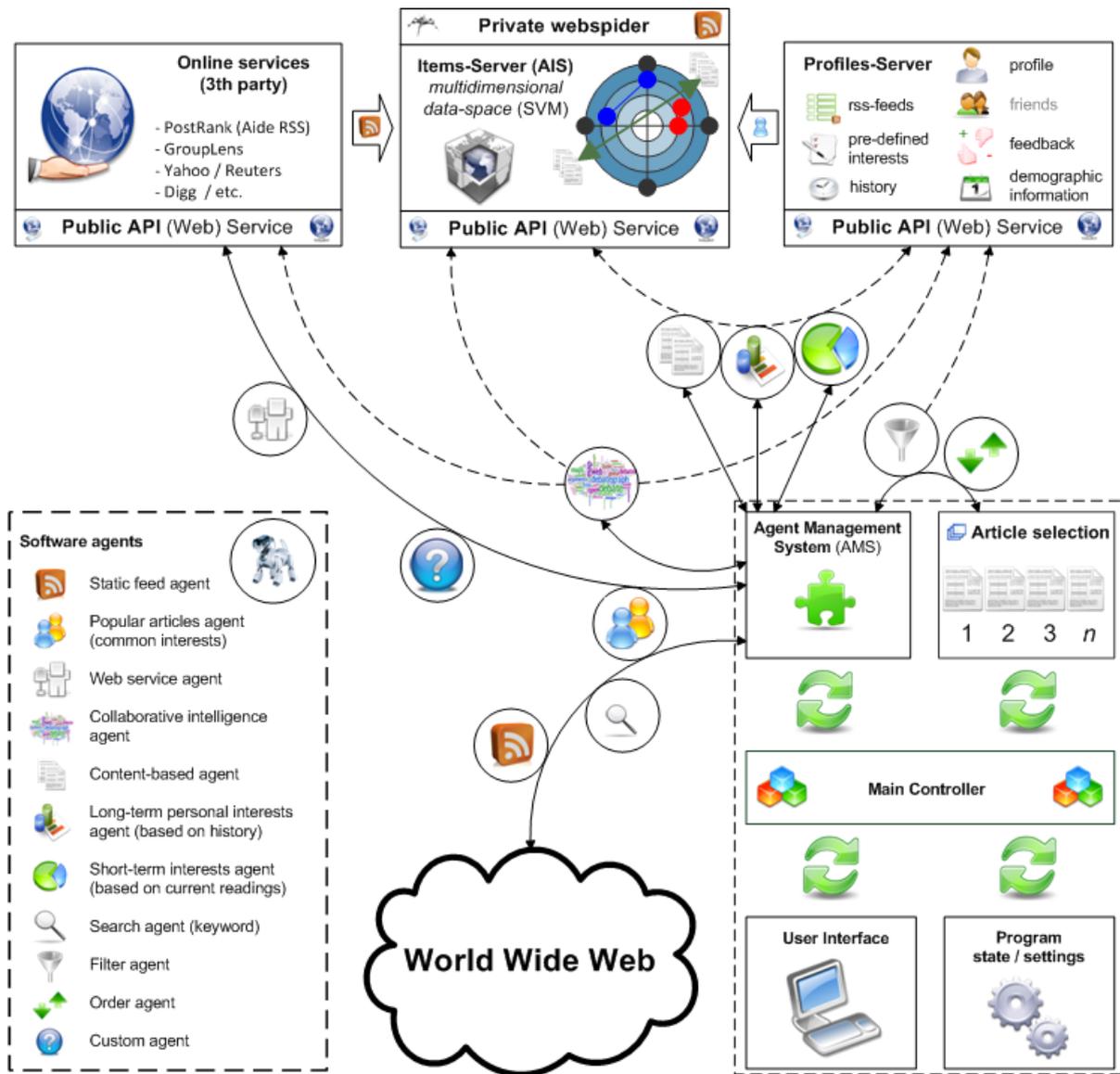


Figure 46: Original system setup (prior development)

## 7.2 Usage instructions

### Introduction

iNewsReader is a netnews recommender system based on RSS feeds. The goal of this website is to recommend personalized news articles. The system is implemented for an ongoing master research project at the department of **Artificial Intelligence**; University of Groningen (RuG); The Netherlands. Due to research purposes this is a highly experimental and incomplete service.

### Instructions

1. First **READ THE INSTRUCTIONS AND NOTIFICATIONS THOROUGHLY!** This page contains important information about the workings of the system, its drawbacks and the research behind.
2. If not done so already, create an account [here](#) and log in to the system (top right).
3. Next you can **add personal RSS feeds** to the home page to directly follow your sites of interest. Many websites provide RSS feeds to their content, links to these feeds are mostly indicated by an icon of this  kind. **Note:** Do never include feeds with personal content (facebook, tweets etc.), because all added feeds automatically become available for ALL users!!!
4. Now you're ready to follow the news. Therefore open articles from the main recommendation at the home page, use your personal feeds or search for them. After reading, provide feedback (vote  ) to the system. This is important! The system needs at least a few votes, preferably both positive and negative, to be able to recommend anything! The more you vote, the better the recommendation will suit your personal interests.

*Note: Only articles opened for reading will be included in the recommendation process. So 'not opening' an article does NOT tell the system you're not interested.*

### Notifications

Because this is a research project, there are some notifications and drawbacks to mention about the system:

1. For statistical measures, sometimes 'random' recommendations are provided; these may not fit your interests at all.
2. It takes about 8 to 10 'rounds' to include a vote into a new recommendation. This is caused by the setup for testing multiple system configurations.
3. A vote is registered if the icon turns its represented color (green: like; red: dislike). Votes are overridden, so if the same article is voted twice, only the last vote remains.
4. The recommendation process itself is executed 'offline'. This means the system queues recommendations and tries to provide new ones on the fly, but sometimes there may be no new recommendations available. Just wait a few minutes or come back later.
5. The same article can be recommended multiple times if different configurations think it's an item of interest. Once an article is read, it won't be included in a new recommendation, but can still be already included in queued recommendations.
6. A recommendation will be displayed only once. To recall previously recommended articles, use the [history](#) page.
7. Speed: The system can be very slow some times (be patient :-). This is caused by the platform and network the system is working on.
8. Uptime: The system may be out of reach for longer periods of time.
9. Feeds: Not all RSS feeds can be read by the system. It's known some feeds (i.e. smashing magazine) cause errors.
10. Articles: Not all websites (i.e. NY Times) allow being loaded in frames (which are used to display the toolbar on top of the page). Therefore sometimes an extra step (click on link) needs to be taken to display the contents in a new window and still be able to provide feedback after reading.
11. Language: The recommendation is optimized for 'English', but other languages are not excluded.
12. Privacy: All user data will be used for this research only, an indication of the usage see the [Charts](#) page.
13. Errors: The system isn't fully optimized and may contain errors. If errors are displayed, ignore them at first, if they reappear: [contact me](#).

## 7.3 Naïve Bayes Classifier implementation

```
1 public class Classifier
2 {
3     private Index first; // WordCounts for first class (Index[token] = count)
4     private Index second; // WordCounts for second class (Index[token] = count)
5     private double priorFirst; // Prior probability for first class
6     private double priorSecond; // Prior probability for second class
7     private double pFirst; // Final (log-)probability for first class
8     private double pSecond; // Final (log-)probability for second class
9     private double firstTotal; // Total terms in first class
10    private double secondTotal; // Total terms in second class
11    private double minimal; // Minimal probability (prevent 0.0 probabilities)
12
13    public double Alpha { get; private set; } // Tuning parameter, influence final probabilities
14    public double Epsilon { get; private set; } // Tuning parameter, influence unknown term probabilities
15    public float Tolerance { get; set; } // Classification tolerance
16
17    public Analyzer( Index first
18                  , Index second
19                  , double priorFirst = .5d
20                  , double priorSecond = .5d
21                  , float tolerance = .05f
22                  , double alpha = 1.0d
23                  , double epsilon = 1.0d
24    ){ // initialize
25        this.first = first;
26        this.second = second;
27        this.priorFirst = priorFirst;
28        this.priorSecond = priorSecond;
29        this.Alpha = alpha;
30        this.Epsilon = epsilon;
31        this.Tolerance = tolerance;
32        // Global counts
33        this.firstTotal = (double)first.EntryCount;
34        this.secondTotal = (double)second.EntryCount;
35        this.minimal = Epsilon / (firstTotal + secondTotal);
36    }
37    // Categorize Article
38    public CategorizationResult Categorize(Dictionary<string, int> tokens, out double prediction)
39    {
40        // initial probabilities
41        pFirst = Alpha + Math.Log10(priorFirst);
42        pSecond = Alpha + Math.Log10(priorSecond);
43
44        // Calculate Term probabilities
45        foreach (KeyValuePair<string, int> token in tokens)
46        {
47            // Term counts
48            double firstCount = (double)first.GetTokenCount(token.Key);
49            double secondCount = (double)second.GetTokenCount(token.Key);
50
51            // skip unknown terms
52            if (firstCount == 0 && secondCount == 0) { continue; }
53
54            // calculate term probabilities P(W_i|C_first) & P(W_i|C_second)
55            double pTokenFirst = (firstCount == 0) ? minimal : firstCount / firstTotal;
56            double pTokenSecond = (secondCount == 0) ? minimal : secondCount / secondTotal;
57
58            // Update log-probabilities P(First|Article) && P(Second|Article);
59            // [SUM probabilities x times termCount]
60            for (int i = 0; i < token.Value; i++)
61            {
62                pFirst += Math.Log10(pTokenFirst);
63                pSecond += Math.Log10(pTokenSecond);
64            }
65        }
66        // Final prediction
67        prediction = (pSecond / pFirst) - 1;
68
69        // Determine class
70        if (prediction <= this.Tolerance) { return CategorizationResult.Second; }
71        if (prediction >= this.Tolerance) { return CategorizationResult.First; }
72        return CategorizationResult.Undetermined;
73    }
74 }
```