DD2476 Search Engine Project
News Recommendation

Federico Baldassarre, Sandra Bäckström, Sebastian Bujwid, Zimin Chen

May 21, 2017

Abstract

In today's society a news user is often overwhelmed by the amount of available articles, spread over thousands of different topics. Reading the front page of a news site, will most likely result in articles that we do not find relevant, and explicitly searching for a topic is often an effort that we do not want to make. News recommendation services therefore aim to provide articles that the user wants to read, without the need to search for them.

In this work we explore the effectiveness of integrating a user reading history in a recommendation engine, in the form of articles that the user has liked or visited. Furthermore, we consider different sources of meta-data that can better describe the content of an article for the purpose of producing recommendations.

Following a prototype implementation, the recommendation system is evaluated. The evaluation study shows that the recommendations are much better than random ones, and tend to improve with more user history. However, the recommendation system still suffers from a *cold start* problem, as we only have limited data about user and the recommendation results lacks diversity.

1 Introduction

Recommendation systems can be a part of almost any technical product. Today there are recommendation systems in popular products such as Netflix. whose service helps the users finding movies or series that they are likely to enjoy, rather than having to search for themselves. By increasing customer satisfaction, companies find business value in recommendation systems making these vital for their products. [1]

This also applies to news services, as their users want to read about topics they are interested in. In today's society, there are many ongoing parallel conflicts, issues and developments worldwide and the users is often overwhelmed by the amount of available articles, spread over thousands of different topics. Reading the front page of a news site, will most likely result in articles that we do not find relevant and explicitly searching for a topic is often an effort that we do not want to make. News recommendation services therefore aim at providing articles that the user wants to read, without the need to search for them.

This work explores the effectiveness of integrating a user reading history in the recommendation engine, in the form of articles that the user has liked or visited. To clarify, this is different from a profiling system that holds information about the user, such as location, age, sex and much more. In this work we create a news recommendation engine that only considers the user history in order to suggest articles.

This approach has been proven useful in search engines, to refine user queries through relevance feedback, e.g a user that searches for 'America' and selects news about 'American football' should be presented documents about sports in future searches. In the same spirit, this can be applied to a recommendation engine, where users' queries are not explicitly expressed, but we try to predict their need of information.

2 Related Work

In the field of recommendation engines, one of the most popular approaches is collaborative filtering. Its effectiveness has been proved on the field through many successful implementations[2]. This method is based on the assumption that people who had similar preferences in the past, will continue to behave similarly in the future. However, this requires large datasets about users and their actions and there are situations where the *cold start* problem prevents a successful outcome. This issue occurs when little or no data is available to base the recommendations on and it is likely to make news recommendation system particularly troublesome, because of the very short life-span of news.

3 Method

3.1 Our Solution

Given the aforementioned constraints of no user data and short life-span of the content, we decide not to base our predictions on a collaborative filtering approach. Instead, we decide to work without any historical data and use only the user activity from the current session. Our method is inspired by the Rocchio algorithm for relevance feedback, where

the original query is expanded with terms from the documents marked as relevant by the user [3].

In our system, when a person uses the service for searching the news, a user profile is created with the history of articles that the user found interesting during the current session. We then use this information to recommend articles without any explicit query. In doing this, we are not limiting the recommendations to the last search, instead we draw them from a potentially broader spectrum of different topics. In other words, the user feedback is kept throughout the session and used as the query itself for to retrieve the recommendations, while in the context of Rocchio, it would only contribute to expand the original query.

In order to get the most information content from the few pieces of news that a user might select, we enrich the documents in store with meta-data. We then base our recommendation queries on information such as the author, published date and publishing source, along with the full-text of the article. Furthermore, thanks to more advanced text-analysis tools, we are able to extract relevant entities and keywords from an article, which we believe are of key importance in representing the gist of a piece of news.

3.2 Evaluation

A *normal* information retrieval system and a recommendation system are similar in many aspects, but they differ in the need they are trying to fulfill. In its simplest version, a search engine has to retrieve documents that are relevant with respect to an explicit user query. In this case, evaluating the result of a query consists in comparing the relevance assigned by the system to a document against the relevance the user would assign. On the contrary, a recommendation system deals with an unexpressed need of information, that has to be learned using the available information about the user and that for this reason will result in being much broader that a single explicit query.

Apart from this inherent difference, another parameter to take into account is the way of retrieving *feedback* from the users. Feedback can be either asked explicitly in the form of a yes/no question (*"do you consider this document relevant?"*) or a numerical evaluation (*"how relevant do you consider this document?"*), or implicitly extrapolated from the user actions (*"did the user click on a result?", "how much time did the user spend on a document?"*). In a normal operation mode, all the feedback should preferably be implicit, but for evaluation purposes we can ask the users from a test group to express an explicit feedback.

In the context of evaluating an information retrieval systems, two key metrics are *precision* and *recall*. The former measuring the ratio of relevant documents over the total of retrieved documents, the latter measuring the ratio of selected relevant documents over the total of relevant documents in storage.

In this evaluation we consider recommendation as a ranking task, which means that our goal is to retrieve a relatively low number of news articles in the hope that the user will find them interesting to read. To keep the evaluation simple we set the limit of our *top-K* recommendation system to five.

When it comes to evaluating a ranking task, two popular metrics are the Mean Average Precision and the Normalized Discounted Cumulative Gain.

3.2.1 MAP - Mean Average Precision

The Mean Average Precision method for scoring a ranking algorithm treats the problem as binary, i.e. either a document is relevant or not. Furthermore, the contribution of a document to the final score is also linked to its position in the ranking, where a relevant document on top of the list is worth more than one at the bottom. To compute the MAP over a set of queries we simply take the mean of the Average Precision of each query. Using $P(k)$ and $R(k)$ to represent respectively the precision and the recall at k, the Average Precision can be expressed as:

$$AP = \sum_{k=1}^{K} P(k) \cdot \Delta R(k)$$

The meaning of $\Delta R(k)$ is the change in recall given by the document in position k, using $r = \min(R, K)$ to mark the minimum between the total number of relevant documents and the number of retrieved documents:

$$\Delta R(k) = \begin{cases} 0 & \text{if the document } k \text{ is } \textbf{not} \text{ relevant} \\ 1/r & \text{if the document } k \text{ is relevant} \end{cases}$$

3.2.2 NDCG - Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain builds on the same concept of MAP of decaying the score for lower-ranked documents, but adds a numerical component to the scores, so that their relevance can be expressed on a scale.

Summing up the document scores along a list would give us the *cumulative gain*. In order to consider the position of a document inside the list we can simply divide its score by a number that grows as its ranking increases. For a list of K rankings, using $rel(k)$ to indicate the relevance of the document in position k, the DGC can be computed as:

$$DCG = \sum_{k=1}^{K} \frac{rel(k)}{\log_2(k+1)}$$

Given a query, every permutation of the list can be scored through DCG and we can tweak the search engine towards the highest scores. In an ideal scenario, where the document storage contains at least K documents with the highest relevance, the maximum DCG achievable is simply:

$$\max DCG = \max_k rel(k) \sum_{k=1}^{K} \frac{1}{\log_2(k+1)}$$

However, if the document storage does not contain enough relevant documents to *fill* the list for a given query, the highest possible DCG will be lower. For this reason, before comparing different queries it is custom to normalize the DCG by dividing by the highest score for that query. This is what gives the *normalized* discounted cumulative gain.

In evaluating our recommendation system, it is not feasible to normalize the scores. In fact, it would mean that every user we include in the study would have to rank the whole

dataset of documents based on his interests. To overcome this issue, we can work under the assumptions that the news database is fairly large and the number of recommendations is small enough compared to the many interests of a user. Hence, there will always be at least K documents with the highest relevance score that can populate the recommendation list and we can compare their unnormalized DCG*s*.

4 Implementation

To test and evaluate these methods, we have implemented our recommendation system: from the news ingestion pipeline, to the search engine itself and a web interface for the user study. The ingestion pipeline, written in Python, takes care of collecting articles from a web API and enriching them with meta-data. The search engine is powered by an Elasticsearch instance, which is in charge of indexing and retrieving articles in response to our queries. The web front-end is mainly written JavaScript and allows us to easily browse the news database, formulate custom queries and track the user behavior.

4.1 Crawling

Any news recommendations task begins with collecting articles. The objective is to retrieve news from several sources, extract various meta-data, perform text-analysis and index the articles into a search engine. To make a realistic collection, this project makes use of nearly two thousands pieces of news from various sources such as CNN, Reuters and NY-times.

We first retrieve a news from the *News API* service, that acts as an aggregator for different news websites[4]. The API also returns some simple meta-data, such as author, publishing date, source and a short description. However, it does not provide the full content, so the next step in the pipeline is to scrape the text with an HTML parser. Afterwards, we use the natural language processing services from *IBM Watson Developer Cloud* to extract keywords and entities from the full text[5]. Finally, the news are persisted in a database and indexed into Elasticsearch, ready for the user's queries.

4.2 Elasticsearch

Elasticsearch allows us to index, store and search the articles. This allows us to easily use formulate custom queries based on the user profile to retrieve recommendations, as well as performing simple searches among the articles.

In Elasticsearch every article is indexed using this structure:

Field	Content
Title	The title as it appeared on the newspaper
Author	The author of the article
Publishing date	The publishing date of the article
Keywords	A list of keywords extracted by Watson
Entities	A list of entities extracted by Watson
Full text	The full text of the article

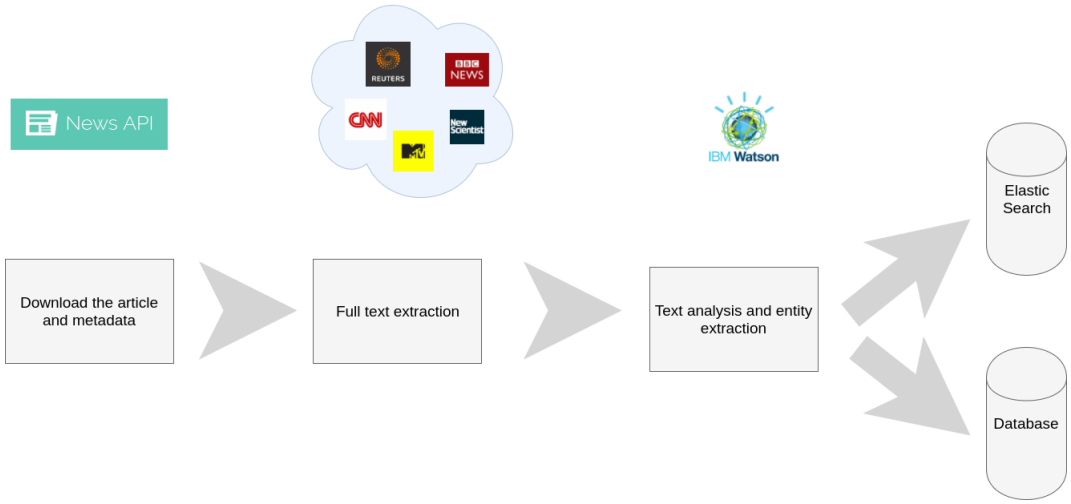


Figure 1: Pipeline of crawling.

When producing recommendations for a user, we take into consideration its profile, in particular the attributes of articles he has read in the past. In doing this, we are not limited to query for similarities in matching fields. For instance, we use a selection of keywords from previous articles to query against the full text of the articles in the index. This is where the great flexibility of Elasticsearch comes in handy, in fact we are also able to express the relative importance, or *weight*, of every sub-query with respect to the overall query. E.g. we can instruct the search engine to score matching entities double the value of matching sources.

In detail, here is how we use the contents of the user's history to produce custom recommendations. Each element is used to produce a sub-query, targeting some field in the index. Bear in mind that in each one of the sub-queries, the query elements might have different weights, but the resulting scores are normalized before aggregating them in the final result. The numbers in parenthesis represent the weights that each sub-query gets after normalization.

Keywords (25%)
Keywords from previously read articles are used to search the full text of other articles. The matches are not required to be perfect, as we allow Elasticsearch to perform some text operations such as pluralization, stemming or *n-gram* searches. Moreover, keywords that appear more than once in the user's history are weighted on the base of their frequency.

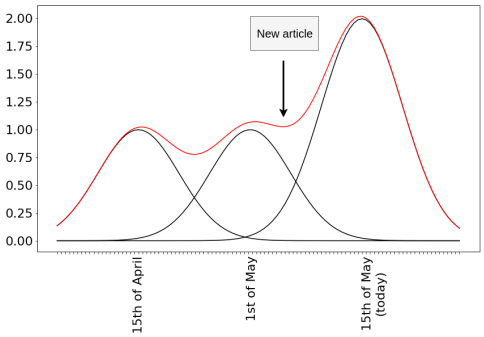
Entities (25%)
Entities from previously read articles are matched against the entity field from the index, the more matches an article gets the better its score. In this case we disable the text analysis functionality, because the entities extracted from the Watson API are already standardized. As in the keywords case we weight the individual entities with the number of occurrences in the user's history.

Authors (19%)
Authors from previously read articles are matched against the corresponding field in the index. Fuzzy matching is used to capture the various ways a name can be written in.

Sources (19%)
Sources from previously read articles are matched against sources from the articles in the index in the same way entities are.

Published Dates (12%)
For this field we use a more advanced weighting system. We measure how close the publishing date of an article is to one of the dates of a previously read article. In detail, each article from the history casts a Gaussian curve centered on its publishing date and the score of a new article is the sum of the values of these curves in correspondence of its publishing date. In addition, we also add a Gaussian around today's date, to boost the most recent articles.

Here we show an example of boosting an article on the base of its distance from today and from the publishing dates of the articles in history



4.3 Frontend

Once the news ingestion pipeline and the recommendation engine are in place we need to be able to assess the quality of our recommendations. The two main functions of the front-end are answering to user-initiated queries and provide recommendations. These are reflected in the structure of the website, which is divided in two sections.

In the first section the user can query the system for articles, that are retrieved based on the similarity between their text and the user query. All the articles retrieved carry the same information stored in the search engine (keywords, entities, author, source and published date) and when a user selects/likes an article, these informations are added to the user profile. As the user searches for different things, his profile will be continuously updated with more and more values. Once again, we remark that the user profile only maintain the reading history and that data such as the user's age is not known.

On the other hand, the recommendation section of the website periodically pulls up recommended articles. Previously visited articles are excluded from the recommendations list, as the user probably does not want to read these articles again. When building the recommendation query we disregard the content of any user-initiated query and we only use information from the user profile. At first, the recommendations are likely to random, as there is no information in the user profile. Later on, the more the user visits results from the search function, the better and more specific the recommendations get.

As a technical detail, the front-end is written in Javascript and the queries are built on client-side, using the domain-specific language of Elasticsearch to express structure and the weights discussed in the previous section.

5 Experiments and Results

5.1 The Evaluation Study

Our evaluation is based on the DCG scores assigned by the users of the recommendation system as a function of the information we hold on their preferences.

In detail, the protocol we are following can be broken down into these steps:

1. Create N_Q queries to be used in the news search engine, these queries should cover as many topics as possible to reflect each user's interests, but at the same time they should be *realistic* to reflect the ability of each user to search for news
2. Every user participating in the study should then sequentially execute these queries:
   (a) Run a query in the news article search engine
   (b) Consider the list of articles retrieved and choose the first 2 documents that he or she finds interesting starting from the top
   (c) Look at the $K = 5$ articles that appear in the recommendation list and assign them a relevance score based on the aforementioned table
   (d) Run another randomly selected query

In this way we are simulating the actual working conditions of the recommendation system, that is based on an implicit binary positive feedback coming from the users' *clicks* or *likes*. At the same time we are explicitly asking the user to express a numeric feedback on the recommended articles. This is then used to compute the DCG of every recommendation list proposed to the user as he consumes the queries and as we accumulate knowledge about his interests. Once all the users have completed the test sequence, we can express the final result for this group as the average of individual results.

In order to assert that having more information about a user allows for better recommendations, we need to compare our system against the baseline given by a recommendation system that does not make use of the user's information. For this reason we need to have a second group of people undergoing the same process, for which however the we disable the user profiling in the recommendation engine.

5.2 Results

The ranking systems used in the study is based on the following three scores:

Gain	Relevance	The news article...
0	Uninteresting	does not appeal the user's interest
1	Somewhat interesting	seems interesting, but the user would not read it immediately
2	Interesting	catches the interests of the user that would read it on the spot

These are the results obtained for the test group (11 people) and the control group (11 people) over a total of 10 set queries.

As presented in figure 2, DCG of control group never changes. This is because users from control group are always presented with the same recommendations, since they do not give any feedback about the content they see.

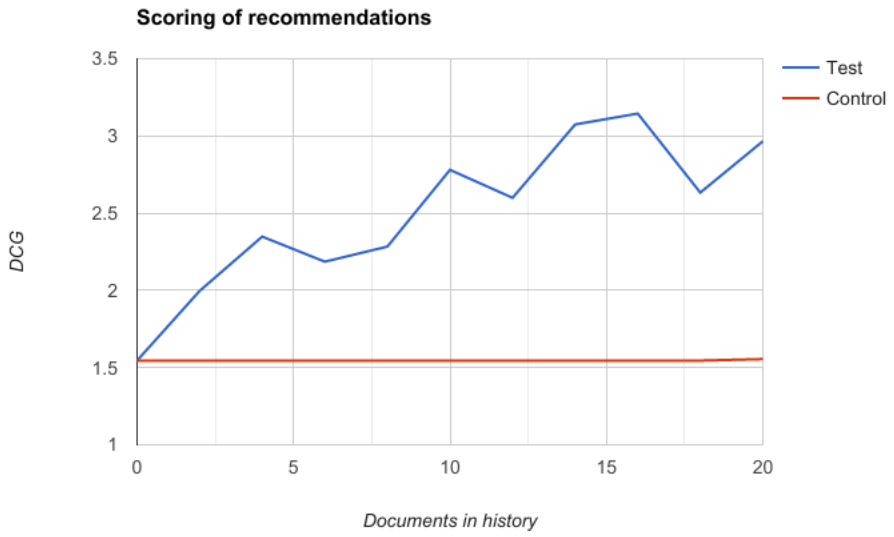


Figure 2: Evolution of DCG score for the test and control groups

6 Discussion and Conclusion

Based on the results, we can see that user profiling widely improves the quality of recommendations. However, the recommendation engine still suffers from the cold start problem. In several cases, we noticed that the recommended news tend to remain the same in the short term, changing only after some iterations over the queries. The reason behind this could be that the first few selected news are similar and they dominate the search for recommended news. Therefore, only when more news are selected and the user profile starts to represent his general interest, the recommended news start to change.

Another issue that we acknowledged is the lack of diversity in the recommendation results. Today's media is dominated by current public figures and topics such as Korea or Donald Trump. If a user shows interest in many Trump-related articles and a fewer articles of other topics, the recommendation will be dominated by Trump.

A possible solution to the problem of non-diversity is doing separate queries for the different topics and then aggregating their results. However, this introduces the issue of classifying different areas in a user's interests.

Future work on this topic would involve carefully selecting additional features from the articles and tweaking their weight in the recommendation query. Furthermore, it would be interesting to see the effect of allowing the user to rank the recommendations (this could be done implicitly, by observing his click behavior), to provide another source of feedback for the system.

A risk with news recommendation systems is them providing only articles you want to read, without ever challenging you with *'unpleasant'* news. Many news publishers have different political views and area of focus, so reading only from your favorite source is will most likely narrow your perspective on the matter.

In order to challenge the readers and make people more aware of other people's perspectives, it would be interesting to incorporate different articles into these services. Perhaps articles within the same topic but from another source, or a completely random article

exposing the reader to new areas and opinions. The world is quite narrow-minded today: we read what we want to read and we see what we want to see. Perhaps what we really need from a news recommendation system is to be helped in understanding different perspectives, in order to create a more welcoming world.

References

[1] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19, December 2015.

[2] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan 2003.

[3] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. 2009.

[4] News api.

[5] Ibm watson developer cloud.

A Ranking metrics

A.1 Example: Mean Average Precision

Consider a dataset of five documents, out of which $R = 4$ are relevant. We are asked to retrieve the top $K = 3$ documents:

Doc	Rel
A	✓
B	✓
C	✓
D	
E	✓

i	Doc	Rel	$P(i)$	$\Delta R(i)$
1	A	✓	1	1/3
2	D		1/2	0
3	C	✓	2/3	1/3
				5/9

i	Doc	Rel	$P(i)$	$\Delta R(i)$
1	A	✓	1	1/3
2	B	✓	1	1/3
3	C		2/3	0
				6/9

Table 1: Example of MAP calculations

A.2 Example: Discounted Cumulative Gain

Here is an example of the evolution of the DCG score over the list of our *top-5* recommendations. For every one of the 10 queries, we add two documents to the user history.

k	$\frac{1}{log_2(k+1)}$	<no query>	Game	America	Science	Money	Food	Company	Election	Development	Best university	Environment issue
1	1.00	0	1	1	1	1	1	1	1	1	1	1
2	1.58	2	1	0	0	0	1	1	1	1	1	2
3	2.00	0	0	1	0	1	1	1	1	1	1	1
4	2.32	0	0	0	1	1	1	1	1	2	1	1
5	2.58	1	0	0	0	1	0	0	1	1	2	1
	DCG	1.65	1.63	1.50	1.43	2.32	2.56	2.56	2.95	3.38	3.34	3.58

Table 2: Example of DCG calculations