Auto-Scoring of Personalised News in the Real-Time Web: Challenges, Overview and Evaluation of the State-of-the-Art Solutions

Paris Carbone, and Vladimir Vlassov
KTH Royal Institute of Technology, Stockholm, Sweden
Email: {parisc, vladv}@kth.se

***Abstract*—The problem of automated personalised news recommendation, often referred as *auto-scoring* has attracted substantial research throughout the last decade in multiple domains such as data mining and machine learning, computer systems, e-commerce and sociology. A typical recommender systems approach to solving this problem usually adopts content-based scoring, collaborative filtering or more often a hybrid approach. Due to their special nature, news articles introduce further challenges and constraints to conventional item recommendation problems, characterised by short lifetime and rapid popularity trends. In this survey, we provide an overview of the challenges and current solutions in news personalisation and ranking from both an algorithmic and system design perspective; and present our evaluation of the most representative scoring algorithms while also exploring the benefits of using a hybrid approach. Our evaluation is based on a real-life case study in news recommendations.**

***Index Terms*—Auto-scoring; recommender systems; scoring algorithms; data mining; machine learning**

I. Introduction

The online web serves today as the main resource provider for live news articles with an increasing rate of tens of thousands of articles per hour. Individual users of the web throughout the world are identically different and have the need to get exposed both to news that are mostly relevant to their interests but also important throughout the communities they are members of. The problem of automated personalised news recommendation (often referred with the term "auto-scoring") and its underlying challenges have attracted substantial research throughout the last decade in multiple domains such as data mining and machine learning, computer systems, e-commerce and sociology. Various techniques and tools have been investigated in both academic studies and industrial use cases, each of them tackling specific challenges.

A typical *recommender systems* approach to solving this problem usually adopts *content-based* scoring, *collaborative filtering* algorithms or more often a hybrid approach. *Content-based* scoring scores articles based on features extracted from their content that best match user profiles assembled individually per user or user group. *Collaborative filtering* on the other hand focuses solely on the users' explicit ratings or implicit actions to associate them with similar users and provide article recommendations based on such similarities.

Due to their special nature, news articles introduce further challenges and constraints to conventional item recommendation problems, characterised by short lifetime and rapid popularity trends. It is therefore often important to be able to score news based on their predicted lifetime and popularity in order to feed users with important information as early as possible. Expected lifetime can also be beneficial for technical purposes such as backend cache invalidation strategies of article data. Finally, due to the vast amount of user generated content in the web, the quality of articles should also be considered in order to provide high quality recommendations. For that purpose several classifiers have been considered that evaluate the quality of articles that can further aid scoring.

In this survey, we provide an overview of the challenges and current solutions to news personalisation and ranking from both an algorithmic and system design perspective; and present our evaluation of the most representative scoring algorithms along with their alternatives while also exploring the benefits of using a hybrid approach. Our evaluation is based on a real-life case study in news recommendations.

A. News Recommendations: Requirements and Challenges

The problem of news recommendations introduces additional challenges to traditional recommender systems. Below we enumerate some of the major challenges that we believe make news recommender systems a special case on its own.

Short Lifetime: Most news articles are items that are of interest for consumption throughout roughly 24 hours. Old articles are in most cases not considered "news" anytime after. A news recommendation system should therefore categorise and recommend articles in a short-term basis compared to a movie recommendation system for example where items can be cached, recommended and consumed for decades.

Rapid Popularity Trends: News can follow extremely rapid popularity trends. For example, upon the wedding or death of a celebrity millions of clicks involving relevant articles could be produced within an hour. That signifies that a news recommender system should detect such trends in time in order to recommend users the "hot" topics and articles as early as possible.

Lifetime Decay: An orthogonal problem to popularity prediction in news articles is to also be able to predict their lifetime that is the amount of time certain articles might gather user attention. There are cases of articles that attract users long

978-1-4673-9566-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ICCAC.2015.9

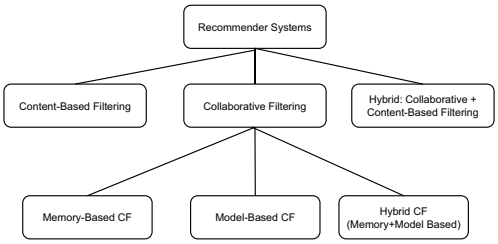

Fig. 1: An Overview of Recommender Systems

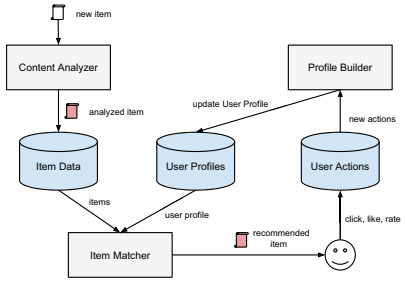

Fig. 2: Content Based Recommendation System Overview

after their creation, perhaps some big news that are not time-sensitive such as the invention of a new system or device. Life decay can be useful not only for recommendations but also for caching strategies at the backend of a recommendation system.

Locality Awareness: Many articles are only interesting to a class of users that are locally collocated, such as an extreme weather report for Stockholm. Furthermore, users that are topologically collocated often express similar interests. A recommendation system should be able to classify users by their location and expose them to potential crucial and location-specific news articles.

Article Quality: The web is full of sources of articles from well-known publishers to popular independent user blogs, thus, the quality of articles might vary significantly. A recommender system should evaluate and also regard the article quality in any of its recommendation strategies.

Serendipitous Discovery: Over-personalisation is generally not good when it comes to news articles. Studies [18], [13] show that users are interested in unexpected news of categories that might not necessarily fall into their favoured topics. Thus, a news scoring algorithm should take into consideration this property, often called "serendipitous discovery" to make news recommendation more interesting and relatively unexpected to users.

II. Recommender Systems: An overview

Recommender systems are *smart* systems that expose items to users based on their expectations that can be either reflected through their past history and interests or based on what other users prefer as a whole. These two main directions in recommender systems are also known as *Content-Based Filtering* and *Collaborative Filtering*. In Fig. 1 we present all known types of recommenders in a single hierarchy.

Content-Based Filtering focuses on modelling user interests and filtering items that fall into the preferred categories for each individual user. Most traditional approaches to recommendations fall into this category and their origins come from retrieval systems theory and artificial intelligence. *Collaborative Filtering* on the other hand is a more modern approach that is based on the assumption that a recommender system is being used by thousands or millions of users so that usage patterns can be extracted from their actions. Usage patterns can be retrieved implicitly from simple actions such as *clicks* or from explicit user actions such as item *ratings*.

In next sections, we will focus on techniques falling on each of these two categories and explain in more detail the main concepts and algorithms used today.

III. Content-Based Filtering

A. Overview

One of the main and oldest paradigms in recommendation systems is to offer users items based on their past preferences. The main idea behind it is to first build an interest profile per user that can be later "matched" with candidate items, one by one and in terms of fitness. This type of *information filtering* is often called *content-based filtering* since all internal attributes of an item are considered for matching with the respective user profile attributes. Thus, a basic, often expensive requirement is to analyse the content of such items and extract a feature vector for each. Additionally, it is required to incrementally *learn* each user's profile by logging their actions and preferences throughout the usage of the recommendation system. An overview of a content-based recommender system can be seen in Fig. 2.

There we can identify the following three main components: The *Content Analyzer*, the *Profile Builder* and an *Item Matcher*. Additionally, we can see three types of data, the *Item Data* which consist of items along with their corresponding feature matrices, *User Profiles* which is a collection of stored user profile attributes for each individual user in the system. Finally, *User Actions* consist of all individual user actions logged throughout the system's life. In more detail:

- **Content Analyzer:** Raw items can be anything from news articles to sale products or movies. A content Analyzer's role is to analyze the content of such items and extract all relevant features in a structured representation. For example, news article items text analysis would result in a feature vector with tagged features such as *topic, source, body length, title length, number of pictures etc.*.
- **Profile Builder:** The role of the profile builder is to analyse user actions such as clicks, ratings or likes logged by the live production system by applying machine learning techniques to create a user model for each user. In the above example, the profile builder should be able to infer the degree each user prefers long article titles, each specific topic or many pictures and store each such measure in a vector. Profile building is usually done offline resulting in an updated profile per user.

- **Item Matcher:** This component usually runs live in production on any recommender system and its role is to "match" user profiles to item features matrices in order to infer the degree of fitness of each item to the user. The result of the item matcher can often be an ordered list of items scored by their fitness to a user's profile.

We show well-known algorithms and techniques associated with each component of content-based recommendation.

B. Content Analysis

The core model in content analysis is the *vector space model*. Based on this model a document collection, such as a collection of articles, is represented by a *document set* $D = \{d_1, d_2, \ldots, d_n\}$ and a set of k terms $T = \{t_1, t_2, \ldots, t_k\}$, often called the *term dictionary*. The *dictionary* terms can be fixed keywords associated with text analysis, entities or any type of terms that can be extracted from documents e.g. by applying pattern matching techniques. Each document d_i is represented in the *vector space model* as a vector $d_i = \{w_{1i}, w_{2i}, \ldots, w_{ki}\}$ where each w_{li} represents the a *weight* of term t_l in document d_i.

Assigning weight terms in documents, especially text documents is a highly researched topic since it is the basic measure used for similarity metrics and retrieval. A dominant scheme for weighting terms in documents is known as *Term Frequency-Inverse Document Frequency* ($TF - IDF$). As its name hints two statistics are considered to weight individual terms, the *term frequency* $tf(t, d)$ and the *inverse document frequency* $idf(t, D)$:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

The *term frequency* measures the frequency of term t in a document d. In its simplest form $tf(t, d) = f(t, d)$ where $f(t, d)$ is the raw frequency of term t in d, however the normalised frequency is generally preferable to minimise the frequency bias in long documents against the maximum of all term frequencies in the same document as such:

$$tf(t, d) = \frac{f(t, d)}{max\{f(w, d) : w \in d\}}$$

The *inverse document frequency* measures the *information gain* of each term t throughout the document set D, also known as "term specificity" and is expressed as the logarithmically scaled fraction of documents in the set that include each term as such:

$$idf(t, D) = log\frac{n}{|\{d \in D : t \in d\}|}$$

where n is the number of documents in the set and $|\{d \in D : t \in d\}|$ denotes the number of documents in the set that term t appears at least once.

C. User Profiling

The most interesting part of content-based filtering is creating user profiles. Based on the *vector space model* a User Profiling component should create a content-based profile per user represented as a weighted feature vector where weights signify the amount a user potentially "likes" each feature. Several techniques used for profiling aim to infer feature weights from individual content vectors generated through user actions. Machine learning techniques are usually applied for such tasks such as *Bayesian classifiers* or *Cluster Analysis* to "train" each user's preference model from their past actions. A more simplified way for modelling user interests is as a set of disjoint categories or topics $C = \{c_{like}, c_{dislike}\}$. The problem of document recommendations can be therefore reduced to binary classification by first estimating the main topic c a document d belongs into and then recommending it to the user if $c \in c_{like}$.

1) Probabilistic modelling of User Interests: Document classification can be achieved by different types of classifiers. Naive Bayes are a simple class of classifiers that build a probabilistic model based on a given sample, also known as *training data*. Thus, with inductive learning Naive Bayes classifiers can estimate $P(c|d)$, that is the probability of a given document d to belong to topic c, as $\frac{P(c)P(d|c)}{P(d)}$.

The conditional probability $P(d|c)$ cannot be computed per se since documents are unique, however, we can overcome this limitation in the vector space model taken that a document is represented by a feature vector. By making the *naive* assumption that the occurrence of tokens is *conditionally independent* given class c the naive Bayes approximation of $P(c|d)$ can be re-stated as follows.

$$P(c_k|d) = P(c_k) \prod_{t \in V_d} P(t|c_k)$$

2) Relevance Feedback and Refinement: Another approach is to use user feedback in order to refine user recommendations, a technique known as *relevance feedback*. This approach originates from information retrieval systems, namely the SMART system [23] which employed *Rocchio's algorithm* and is based on the assumption that users are aware of their information needs and can evaluate results back to the recommendation system that can therefore refine future results.

There are various variations considered, however, the general idea stays the same. According to *Roccio's algorithm* user profiles are classifiers synthesised in a linear way in the *vector space model*. Classifiers can be built either for the whole user profile or for each topic or class per se. Upon the usage of the system the user can classify documents as related or non-related resulting to the two documents sets D_r and D_{nr}. Thus, the problem of measuring the relevance of a document d to a category $c_i \in C$ is reduced to building a classifier $\overrightarrow{c_i} =< w_{1i}, w_{2i}, \ldots, w_{|T|i} >$ where T is the set of terms and each w_{ij} represents a weight of term $t_j \in V$ to class $c_i \in C$. The weight of each term can be computed as a linear

combination of the mean term weight vectors (e.g. using TF-IDF measures as weights) of the document sets D_r and D_{nr}:

$$w_{ki} = r_{pos} \cdot \sum_{d_j \in D_r} \frac{w_{kj}}{|D_r|} - r_{neg} \cdot \sum_{d_j \in D_{nr}} \frac{w_{kj}}{|D_{nr}|}$$

The weights r_{pos} and r_{neg} are used for controlling the modified vector towards being more sensitive to either negative or positive user feedback.

We can build one classifier per profile and measure the similarity of each document to the profile vector as a means of scoring documents directly to user profiles. In the case of assigning classes to documents we can pick the class with the highest similarity $c_d = argmax_{c_i} sim_{\vec{d}, \vec{c_i}}$. In its testing phase the Roccio's classifier resembles clustering techniques (e.g. nearest neighbour or nearest centroid) where we assign a given document the label of the class whose samples' mean is closest. The difference in the training phase is the users' contribution to incrementally build positive and negative document sets. This technique has been used in several news recommender systems, e.g. Newsfeeder and Fab [21], [3].

D. Item Matching

The basic requirement for matching items expressed in the *vector space model* is to consider a basic *similarity measure* between items as a distance metric. As stated earlier both items and user profiles are expressed as feature vectors, thus, the "closeness" between items should consider the amount of matches between such features. A widely used similarity metric is the *cosine similarity* since it measures the cosine of the angle between two vectors A and B:

$$sim_{A,B} = cos(\theta)_{A,B} = \frac{A \cdot B}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Such a similarity metric can be applied for example between a document and a user profile feature vector. Similar measures are the *Sørensen-Dice similarity* and *Jaccard coefficient*:

$$dicesim_{A,B} = 2\frac{|A \cdot B|}{|A|^2|B|^2}$$

$$jacsim_{A,B} = \frac{dicesim_{A,B}}{(2 - dicesim_{A,B})}$$

E. A Use-Case of Content-Based Filtering

One of the most direct actions used for building profiles are the users' clicks. A use case of such an approach is the strategy of Google News [19] which employed a click-based strategy to predict the distribution of clicks between different topics for each user. Google's recommendation system for news makes use of both the history of user clicks and the general click distribution in the geographical area of the user. That is due to the fact that user interests are constantly changing and the their respective click distributions are influenced by the local news trends. For example, in Spain a significant number of users might look for news related to sports during the euro cup even when some of them are not generally interested in them.

User actions considered by the recommendation system as feedback are mainly their clicks which are translated as a positive vote for the category each click belongs to. Given a fixed predefined set of article categories $C = \{c_1, c_2, \ldots, c_n\}$ the system logs for each user, one can determine their respective click distribution per topic in every period t as follows.

$$D(u,t) = \left(\frac{N_1}{N_{total}}, \frac{N_2}{N_{total}}, \ldots, \frac{N_n}{N_{total}} \right)$$

where $N_{total} = \sum_i N_i$. The same metric is also logged on each period per location denoted as $D(t)$. The predicted future click distributions per user are therefore computed in three steps:

1) The system computes each users current *genuine news interests* regardless of the news trend, from their past click distributions;
2) Each user's general genuine news interests model is being updated (incrementally) to reflect any recent changes;
3) The users' near future predicted interests are being computed by combining their general genuine news interests and the current news trend in their location.

The *genuine interest* of a user in topic category c_i over a time is modelled as the probability of the user clicking on an article from that category. Since article recommendation can be based on the predicted behaviour of each individual user in the near future the final step is to predict the *click distribution* of a user in the next time period, e.g. one hour. This method has being reported to work well in combination with a collaborative filtering approach since it can be applied to rather new articles without the need for gathering significant user clicks to be accurate. It can also optimised in terms of performance by applying only incremental updates of the final interest model by incrementally adding only each period's genuine user interest classifier to the total classifier.

F. Applications and Limitations

Content-Based Filtering is one of the most fundamental approaches to item recommendations. It is still considered today a crucial part of any recommender system since it deals well with the *first-rater problem*, thus, it can give value to bootstraping recommender system with no significant user activity. Especially in cases where new items are published with a high rate, a content-based algorithm can still offer them as accurate recommendations to users without the need to gather user actions in extended time periods. However, content-based filtering comes with a high cost of profiling individual user interests. Furthermore, it maintains tight coupling in the model with the domain that user profiles are trained on. Finally, over-specialisation does not really encourage serendipitous discovery which is a preferred measure in many use cases such as music recommendations.

IV. Collaborative Filtering

In our *data-driven* era user behavior is being constantly logged and used for improving the quality of service in various applications and services. Thus, a dedicated class of recommenders has been established that builds mainly on user actions, known as "Collaborative Filtering". Such recommenders do not apply any content inspection on the items they are recommending, in contrary, they base their recommendations on what other similar users have rated or clicked. One of the closest prior approaches to collaborative filtering coming from the *AI* field, is the nearest neighbor approach for classification where the k nearest users to a queried user are used too infer interests.

A. Overview

Collaborative Filtering (CF) techniques exploit the interest similarity between users of a system in order to offer them future recommendations rather than analysing the content of items. Such techniques regard user ratings or clicks to find *implicit groups* of users or items (based on user activity) that are closely related and thus require their *active participation* for higher accuracy in their predictions. Furthermore, one of the main benefits of such a content-agnostic approach is the higher granularity (item-level) of recommendations in contrast to the topic/term-based level of content-based methods. In this section we will analyse the different types and algorithms used in collaborative filtering, several recent developments and also provide use cases in the context of the news recommendation problem.

In general, CF techniques vary based on their approach, however they conceptually fall into one or both of the following two categories [26]:

Memory-Based CF: This category of CF identifies user recommendation as a similarity computation problem and thus is focused on first finding the *top-K* most similar users to a given user to provide recommendations and then recommend that user the *top-N* items based on aggregate ratings within the *top-K* user group. This is the simplest and mostly deployed form of CF and can be highly effective especially when rating matrices are not significantly sparse.

Model-Based CF: A more fundamental way of providing recommendations with CF is to train the recommender system with machine learning techniques in order to infer hidden behavioural patterns within users of which the behaviour (clicks, ratings etc.) is being modelled. User modelling allows for further *compression* techniques such as the *Singular Value Decomposition (SVD)* and hidden latent semantic modelling for finding hidden factors that group users while also dealing with sparse data.

B. Memory-Based Filtering

The core function in Model-Based CF is to compute and sort users or items by similarity metrics on weight matrices. The process of applying Memory-Based CF has two closely related variations: *item-centric* and *user-centric* similarity computation. In the case of *item-based* computation the goal is to create a similarity *item-item* matrix with respective weights between each set of items that reflect the co-rating ratio among users. To achieve this a set of all users that have rated each set of items is considered to apply the similarity computation. This approach is often adopted by e-marketplaces and the weight matrix is used for item recommendations based on items bought (eg. users who bought/viewed item i also bought/viewed items j and k). In contrast, on the *user-centric* approach the user similarity is being computed first between users who have rated each set of items to generate a *user-user* similarity matrix. In both cases certain similarity measures are needed to be adopted that are applied on user actions per se.

1) Similarity Metrics for Memory-Based Filtering: The mostly used similarity metrics for this purpose are *Correlation-Based* such as *Pearson's correlation* and the *Vector Cosine-Based similarity*. Pearson's correlation between two variables X and Y measures the degree of linear dependence between them, assigning a value v in $[-1, 1]$ where $v = 0$ translates as no correlation, $v > 0$ shows positive correlation and $v < 0$ negative correlation. In its general form for a given sample n Pearson's correlation between X and Y is the following:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

The cosine similarity between two vectors has been introduced in III-D. A benefit of using Pearson's correlation coefficient over the cosine similarity metric is that it applies normalisation to the vector similarity metric and thus it is preferred when user rating sizes vary.

2) Computing Similarity Matrices: The basic construct considered in memory-based CF is $user \times item$ matrix for user ratings. This ratings matrix contains all user ratings per item and thus, in most cases it is rather sparse.

For **user-based** filtering the similarity between each pair of users i and j should be computed first in order to create the *user-user* matrix $w(u_i, u_j)$. Each cell (i, j) in the matrix should therefore show the similarity $w_{i,j}$ between users i and j based on their rating vectors. By applying Pearson's correlation coefficient between the rating vectors of users i and j we have:

$$w_{i,j} = \frac{\sum_{k \in I}(r_{i,k} - \bar{r_i})(r_{j,k} - \bar{r_j})}{\sqrt{\sum_{i \in I}(r_{i,k} - \bar{r_i})^2}\sqrt{\sum_{i \in I}(r_{j,k} - \bar{r_j})^2}}$$

where I is the set of items rated by either i or j and $\bar{r_i}, \bar{r_j}$ are the average ratings among the co-rated items of i and j respectively.

Alternatively, an **item-based** approach aims to create an *item-item* matrix $w(item_i, item_j)$, also referred as *co-visitation matrix* between each pair of items based on user rating vectors. Thus each cell (i, j) should reflect the degree of co-visitation or correlation $w_{i,j}$ between items i and j. By applying the Pearson's correlation in this context we have:

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r_i})(r_{u,j} - \bar{r_j})}{\sqrt{\sum_{u \in U}(r_{u,i} - \bar{r_i})^2}\sqrt{\sum_{u \in U}(r_{u,j} - \bar{r_j})^2}}$$

where U is the set of users that have rated either i or j and $\bar{r_i}, \bar{r_j}$ are the average ratings of items i and j respectively.

Since finding the top-K neighbours in large sets of items and users can be often too expensive, several techniques have been developed that apply dimensionality reduction principles in order to ease the process of neighbourhood estimation. One of the most popular techniques is called *Locality-sensitive Hashing* which achieves that goal in sublinear time.

3) Item Scoring Prediction: Given that a the top-K closest user set U has been computed for each individual user u, a scoring function should therefore predict the suitability of each item i (eg. an article) rated by users within U to that user. In its general form a rating represents an aggregate of other user ratings within U given an aggregate function f as $r_{u,i} = f_{u' \in U} r_{u',i}$

For example a simple average of the $top - K$ user ratings could be the following:

$$r_{u,i} = \frac{\sum_{u' \in U} r_{u',i}}{K}$$

A more adopted approach is to add user similarities as weights in the average as such:

$$r_{u,i} = t \sum_{u' \in U} sim(u, u') r_{u',i}$$

where t is a normalisation factor.

C. Model-Based Filtering

1) Clustering CF: Perhaps the dominant model-based approach to collaborative filtering today is applying clustering of similar users based on their interests or items based on associated user actions/interests. Clustering resembles memory-based top-K approaches where the "closest" items or users are needed to be found in order to consider only intra-neighbourhood ratings for item recommendations. We will continue this section from an item-centric point of view, however, the process is similar for finding close users. To motivate the need for efficient clustering techniques we will provide some further background on the general problem of finding closest neighbours.

In our model we can represent each item I as a set containing users that are interested in that item , eg. $I = \{u_4, u_{10}, u_{244}\}$. For example, in a news recommender service those can be the users that have actually clicked and read a specific article. Since items are sets, we can measure the similarity between two items, eg. I and $K = \{u_5, u_{10}\}$ using the *Jaccard coefficient* $J(I, K) = \frac{|I \cap K|}{|I \cup K|}$

In our example that would yield $J(I, K) = \frac{1}{5}$. In general two sets are identical when the Jaccard similarity is 1 and disjoint when it is 0. In order to find the nearest neighbours between items within our data set we should therefore calculate the Jaccard similarity between all pairs of existing items in an $Item \times Users$ matrix, an operation that would be inefficient when users or/and items are too many (eg. 10^6 scale) and thus such a matrix is too sparse. ***MinHash*** is a technique that is widely used among other use cases to estimate the Jaccard similarity of sets by also reducing the computational complexity of this clustering problem in terms of number of comparisons.

*a) **MinHash Clustering**:* The MinHash technique is practically an estimator of the Jaccard similarity, a scheme invented by Andrei Broder et al [7]. Its applications vary from detecting duplicates to clustering similar itemsets in sublinear time. In each iteration a hash function h is used to to hash all members of each two sets A and B to distinct integers. For each set S let $h_{min}(S)$ be the minimum hash value $min_{x \in S} h(x)$. When $h_{min}(A) = h_{min}(B)$ (the minimum hashed item lies within $A \cap B$). That yields that $P[h_{min}(A) = h_{min}(B)] = J(A, B)$.

Each set S with n items can be therefore reduced to a smaller vector consisting of m minima where each minima is determined by an h function $S' = [s_1, s_2, \ldots, s_m]$ where $s_i = h_{min,i}(S)$. We call S' the signature of S and compute the estimated Jaccard similarity of signatures A' and B' as $P[A'_i = B'_i]$. The problem of computing the similarity between all existing sets of items still persists especially when we are only interested in similar itemsets or sets that bear a minimum bound of similarity. *Locality Sensitive Hashing* offers a solution to the problem by only considering the computation of strongly similar itemsets (ie. nearest neighbours), a technique that is widely used for computation reduction (over parallelism) in collaborative filtering.

*b) **Locality Sensitive Hashing (LSH)** :* The basic idea of Locality Sensitive Hashing, first introduced by Indyk and Motwani [11], is to apply *bucketing* in the signature matrix of all items and reduce the similarity computation only to items that fall into the same bucket. The main property needed for bucketing in this case is to use a hashing function that puts similar itemsets into the same bucket with high probability. In order to increase the probability of two strongly similar itemsets falling into the same bucket the signature matrix is therefore partitioned vertically into a fixed number of *bands*. For each band a hash function is chosen that hashes the intra-band signatures into a dedicated bucket array per band. The correctness of this method relies on the assumption that the more identical two item signature vectors are the more probable it is to fall into the same bucket on at least one band. In terms of accuracy LSH relies strongly on the number of bands chosen to partition the signature matrix. Given that two items A and B have similarity s, for r number of rows partitioned in each band the probability that all rows in a band are equal is s^r while the probability that at least one row in a band is unequal between A and B is $1 - s^r$. Thus, the probability that no band resolves into equal rows for A and B is $(1 - s^r)^b$. This yields that that the probability of A and B falling in the same bucket in at least one band is $1 - (1 - s^r)^b$. Therefore, we can adjust b to trade-off

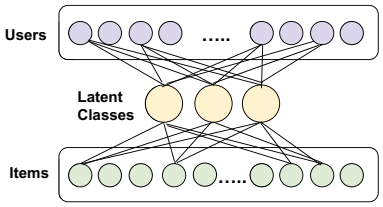

Fig. 3: Probabilistic Latent Semantic Indexing

between lower computational complexity and accuracy by adjusting b accordingly. Since the aforementioned probability follows a sigmoid curve (S-curve) relative to the similarity distance threshold we want to consider for bucketing the desired threshold can be approximated by $(1/b)^{1/r}$ [11].

2) Probabilistic Latent Semantic Indexing for CF: Probabilistic Latent Semantic Analysis (PLSA or PLSI) is a statistical technique originally derived from latent semantic analysis and aims to model the probability of co-occurrences as a combination of conditionally independent multinomial distributions. For example, in the case of estimating $P(u|i)$ between user and item co-occurrences $(u|i)$ the PLSI approach is to introduce a hidden variable Z also denoted as the *latent class* with domain $\mathcal{Z}$ where $||\mathcal{Z}|| = L$.

The core of the idea of introducing a hidden (latent) variable is to make the two variables studied for co-occurrence (in this case users and items) conditionally independent as it is visually expressed in Fig. 3 . The modelled probability would therefore be $p(i|u;\theta) = \sum_{z=1}^{L} p(z|u)p(i|z)$

The model is purely controlled by parameter θ which represents the cumulative probability distributions (CDFs) of $p(z|u)$ and $p(i|z)$. Another way of viewing this technique is to consider it as a clustering approach, thus, p(z—u) would reflect the distance of u to cluster z where the number of clusters is finite (L). Probabilistic clustering methods such as PLSI are also referred to as *soft clustering* techniques since each data point or pair of points is studied for fitness over all classes or clusters along with a proportional probability or weight for each in contrast to *hard clustering* techniques (e.g. K-Means) where each point belongs to one cluster with probability 1. In practice, the CDFs in θ can be estimated in an iterative function using the *Expectation Maximization* (EM) algorithm to learn the maximum likelihood parameters of the model as described below.

a) The Expectation-Maximization (EM) Algorithm: EM is a widely used iterative method that aims to find the maximum likelihood estimates of parameters in statistical models where latent variables are included. The two iterative steps performed in EM are the *Expectation* and the *Maximisation* step, also denoted as E and M.

The technique is as follows: Given a dataset D we want to estimate parameters θ where θ' is a dummy variable and D' is the denoted dataset with latent variables included. We consider an initialisation step for θ where $\hat{\theta}^{(0)}$ is the initial value and a series of iterations of an E-step (expectation) and an M-step (maximization) as follows:

- Define initial parameters $\hat{\theta}^{(0)}$
- Repeat E,M steps until convergence or for a fixed number of iterations T
  - **E-step:** Compute $Q(\theta', \hat{\theta}^{(i)}) = E(f(\theta'; D')|D, \hat{\theta}^{(i)})$
  - **M-step:** Estimate $\hat{\theta}^{(i+1)}$ as $max_{\theta'} Q(\theta', \hat{\theta}^{(i)})$

3) Alternating Least Squares: Alternating Least Squares or *ALS*, is a variant to matrix factorisation that performs well in large scale deployments [32][12]. In summary *ALS* provides an optimisation of the known factorisation problem : $P = X \times Y^T$ by applying iterative row parallelisation. Another benefit is its ease to consider implicit datasets, eg. binary actions that is the usual case. On each parallel iteration either X or Y factorization matrix is assumed to be fixed and the optimisation is being solved for the other.

A typical least-squares approximation works as following: Given some regularization weights $c_{ui} = I + ar_{ui}$ we iterate to minimize:

$$\sum c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum ||x_u||^2 + \sum ||y_i||^2)$$

The ALS variation of least-squares approximation assumes a fixed Y to compute the optimal X (and vice versa). Furthermore, each row x_u is independent and C_u is the diagonal matrix of c_u, the user strength weights. Thus, it yields $x_u = (Y^T C_u Y + \lambda I)^{-I} Y^T C_u p_u$

4) Bayesian CF: One of the first approaches to CF involves creating a probabilistic model using Bayesian Networks [6]. A Bayesian network consists of nodes representing items (or users respectively) in a domain, having states that reflect the user ratings or a "no-rating" state. A learning Bayesian network algorithm would therefore generate a network that reflects a hierarchy of item dependencies. The parents of each item in the hierarchy are considered the best predictors for its ratings and can be used further for predicting user scores. The model resembles a decision tree structure from which all conditional probabilities can be derived for each node.

D. Applications and Limitations

Collaborative filtering is widely used today in production for movies, news, music and other recommendation use cases. One of the greatest benefits of this type of recommenders is that they are generic enough to be used in multiple use cases. It is often the case that people who bear similarities in one subject might probably be similar in other things as well like drink preferences. This makes collaborative filtering really powerful, however, it is usually not used as a single technique. In most cases collaborative filtering is used in combination with content-based approaches to deal with the problem of bootstrapping but also to enrich the recommendations quality.

V. Special Cases in News Recommendation

In this section we analyse domain-specific cases for recommender systems on personalised news recommendation. As described in Section I-A news articles introduce special temporal and spatial properties. It is therefore important to address such properties and reflect these features as a part of a final

scoring algorithm for user recommendations. The dominant characteristics of news articles, addressed either independently or on conjunction [10] for more successful recommendations, are the *popularity, freshness* and *user interests*.

A. Hybrid Content-Based and Collaborative Filtering Approaches

It is evident that content-based approaches themselves cannot reflect trends and user behaviour in general while collaborative filtering techniques rely heavily on the availability of user logs. Kirshenbaum et al [15] conducted a comparative study upon both historical and live data to identify the combination of recommendation techniques that can offer the best click-through rate (CTR) among common alternative scoring methods. The most effective method was a hybrid of item-item collaborative filtering and content-based TF-IDF method offering 37% improvement over baseline article recommendation methods. Google News adopted a similar hybrid strategy to enhance their news recommender system by adding content-based scoring, mentioned in section III-E to their existing collaborative filtering recommendation approach. They further showed an improvement of 30.9% in terms of CTR compared to their existing recommendations via live AB testing.

B. Article Popularity Measures and Prediction

The popularity of articles is an important factor when it comes to recommendations and has already been addressed indirectly both in the content based filtering and collaborative filtering techniques presented before. In this section we will see in more detail how we can quantify popularity explicitly from user actions (eg. comments or clicks) and incorporate it in any scoring algorithm. Antoniadis et. al [29], [27], [28] have proposed a comment-based approach to measuring and therefore predicting popularity of new articles or topics as well as lifetime expectancy estimation which is an orthogonal problem. Alternative approaches [17], [4], [2] analyse social activity for forecasting "hot" news and topics and thus assign them higher score weights.

1) Popularity Prediction with Activity Monitoring: The authors in [29], [27] propose the time interval between an article's creation and the last user comment made on that article as a metric for article popularity and further observed that the cumulative distribution function (CDF) of that metric fits the power law distribution [9]. Therefore, the most suitable prediction model for popularity proposed in the same study is *linear regression on a logarithmic scale* (LinearLog) which was also employed in the past by popular web content recommendation services [30]. The *LinearLog* model $\hat{N}_s^{LN}$ can be trained per article s at a time t_i, given the current number of comments $N_s(t_i)$ to predict the number of comments at any time t_r where $t_i < t_r$ as such:

$$\hat{N}_s^{LN}(t_i, t_r) = exp\left(ln(N_s(t_i)) + \beta_0(t_i, t_r) + \frac{\sigma_0^2(t_i, t_r)}{2}\right)$$

where β_0 is a coefficient obtained through maximum likelihood estimation while σ_0^2 denotes the variance of the residuals on a logarithmic scale. An example scoring algorithm that can be derived from such estimator would be a ranking of the predicted popularity in 24h after each article's publishing time $\hat{N}_s^{LN}(t_i, 24)$.

2) Popularity Prediction with Content Inspection: In contrast to activity monitoring several studies have been conducted that base their popularity approximations of news articles solely on their content even prior to their publication by inspecting their content in correlation with current trends from the social web. In [4] the authors consider a multi-dimensional feature space that can be derived from article content and train their model using both regression and classification methods. They further show from an inspection from Twitter posts that a 84% accuracy of articles popularity prediction can be achieved solely by content inspection. The main characteristics considered in the study were 1) the artice source, 2) the main topic, 3) the subjectivity of the language used in the article and 4) named entities mentioned in the article. To train their models the authors introduced an article nominal class characterising the type of each article based on its popularity (i.e. number of tweets).

The accuracy of machine learning algorithms was evaluated over a set of 10000 articles with the results seen in Table I.

Algorithm	Accuracy
Bootstrap Aggregation (Bagging)	83.96%
Decision Trees (J48)	83.75%
SVM	81.54%
Naive Bayes	77.79%

TABLE I: Accuracy Evaluation [4]

C. Quality-Centric Scoring

Another important factor that is employed in several scoring algorithms is the quality of an article. As Bendersky et al. advocate in [5] quality-based measures further improve scoring over document ranking techniques in retrieval systems that are solely link-based such as PageRank [22]. Link-analysis in scoring documents is according to the same authors equivalent to collaborative-filtering techniques, thus, missing the important contribution of content-based methods on ranking. The proposed *quality-biased* ranking function for a document D can be combined in a linear fashion with existing scores and consists of a combination of feature scores multiplied with their respective parameter: $\sum_{L \in \mathcal{L}} \lambda_L f_L(D)$. The estimation of the feature parameters can be made using the coordinate ascent algorithm [20] or any other linear parameter estimation algorithm such as RankSVM [14].

VI. Evaluation of Scoring Algorithms

A. Overview

The correct evaluation of scoring algorithms and recommender systems accuracy in general is of great importance since it quantifies their predictability when it comes to user interaction with a system.

B. Methods

There are generally two main methods for evaluating recommender systems: the *offline* and *online* evaluation [25], [10]. *Offline* evaluation is more common and easier to conduct since no interaction with the user is necessary, although, special attention should be given to the bias of the user choices/ranking. It is generally preferable to conduct offline evaluation of recommendation algorithms across user data generated using a baseline algorithm to minimise such bias. The *online* evaluation exposes a pool of "unaware" users directly to a recommendation system under evaluation and instantly measures its accuracy. Given a good sample selection, the online evaluation is considered more trustworthy and closer to reality.

1) Offline Evaluation: In general, *offline* evaluation is useful for evaluating the predictive power of a recommendation algorithm. In most cases this method is used first to compare algorithms on a given data set consisting of user data prior to the deployment of the algorithm. There are several ways of assesing the predicted scores of an algorithm given a multi-user dataset that spans over a long period of time [25].

Consistent Simulated Time Evaluation: A "realistic" approach to simulating scores across different algorithms over a time history is to train a scoring algorithm in temporal order for each user. Since the re-computation of a training model is a computationally intensive operation a common approach is to sample the users and all the time intervals to assess the scoring capability of each algorithm on sampled intervals. In that case, we only consider the user data prior to each interval to train each algorithm and use it to predict user rankings until the next random interval. Since time consistency is kept across users in this approach, it is only useful when absolute time is important for the given recommendation problem.

Random Simulated Time Evaluation: This approach is similar to the prior one with the main difference that time is not kept consistent across users. Thus, we consider different time samples per user and use these to compute its consecutive scoring approximation of a given algorithm. This approach is less close to a realistic evaluation but it can reduce the time bias of the train data in use while also being less computationally intensive.

Time Agnostic Evaluation: In cases where the time metric is not important we can totally ignore it during evaluation and simply sample users and items into two distinct sets for training and test. Thus, each scoring algorithm is trained based on a subset of items ranked by each user and evaluated for each predictability on the test data set of the rest of the items.

2) Online Evaluation: The most effective method to assess the quality of a scoring algorithm is to use it directly on a recommender application and observe its effect on the user behavior. To effectivelly evaluate different scoring algorithms [16] at the same time in a live deployment, it is usually prefferable to direct users to different recommender systems without their awareness and assess the partial accuracy measures of each algorithm for comparison purposes. In most cases online evaluation is preferrable when algorithms have been priorly tested upon an offline evaluation in order not to cause a negative impact to the recommender system.

C. Measures

Perhaps the most important property of a recommender engine is its ability to predict user behavior *accurately*. Thus, *accuracy* is considered as the default metric to assess how "good" a recommender is, under the assumption that users prefer results who match their expected interests. There are generally three main metrics used for recommendation accuracy assessment that are also associated with scoring algorithms in general:

Rating Prediction Accuracy : For measuring how close the predicted ratings match the actual user ratings. A typical measure is the *Root Mean Squared Error* (RMSE) and its derivatives.

Usage Prediction Accuracy : For measuring the ability of the recommender to predict the usage of items (eg. clicks) by its specific user. Typical measures are the *Precision, Recall* and *ROC curves*.

Ranking Accuracy : For measuring the quality of the order the items appear to the user based on a recommender algorithm. One measure used often is the *Normalized Distance-based Performance Measure* (NDPM).

We will continue with a more detailed descriptions of the most fundamental measures used on recommendations, as mentioned above.

1) The Root Squared Mean Error (RSME): The RMSE expresses the degree that the predicted ratings $\hat{r}_{ui}$ vary from the actual (hidden) user ratings r_{ui}. The RMSE is considered among a *test set* $\mathcal{T}$ as such:

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} (\hat{r}_{ui} - r_{ui})^2}$$

Similarly, the *Mean Absolute Error* (MAE) is given as such:

$$MAE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} |\hat{r}_{ui} - r_{ui}|}$$

It is often recommended to normalise the RSME and MAE measures to the scoring range or use their average across users especially when the distribution of items is unbalanced [1].

2) Usage Precision Recall and ROC: Many recommender systems such as news article recommenders or dating services focus more on a simple binary prediction of the usage (or absense) of items by users rather than explicit ratings whatsoever. In such cases more appropriate measures are the Precision, Recall and the False Positive Rate of the recommender.

The basic metrics used in such measures are the True/False Positives (TP, FP) and True/False Negatives (TN, FN) which are counters for item recommendations falling in the categories specified in Table II .Based on the measures above Precision, Recall and False Positive Rate can be computed as seen in Table III.

	Recommended	
	Yes	Not
Used	TP	FN
Not Used	FP	TN

TABLE II: Recommendation Usage Classes

Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
False Positive Rate	$\frac{FP}{FP+TN}$

TABLE III: Recommendation Usage Classes

The precision and recall offer a significant trade-off. *More* recommendations can lead to improved Recall but also reduced Precision. A common way to visualize the relationship between Precision and Recall is by using ROC curves (true positive-false positive rate) or precission-recall curves.

3) The Normalised Distance-based Performance Measure (NDPM): Most ranking accuracy evaluation measures assume a reference ordering of items for each user. For example, a reference ranking of videos for a user could be the videos the user has ranked so far in descending order. Since there are discrete scoring values in most cases, it is possible to have equal reference scores among items. The Normalised Distance-based Performance Measure (NDPM) does not give any penalty when there is a tie of the scores between items in the reference and actual set and is given as $NDPM = \frac{C^- + 0.5C^{u0}}{C^u}$ where:

$$C^+ = \sum_{ij} sgn(r_{ui} - r_{uj}) sgn(\hat{r}_{ui} - \hat{t}_{uj}) \quad (1)$$

$$C^- = \sum_{ij} sgn(r_{ui} - r_{uj}) sgn(\hat{r}_{uj} - \hat{t}_{ui}) \quad (2)$$

$$C^u = \sum_{ij} sgn^2(r_{ui} - r_{uj}) \quad (3)$$

$$C^{u0} = C^u - (C^+ + C^-) \quad (4)$$

When the ranking matches the reference ranking then NDPM gives a score of 0, otherwise it gives a higher score with a 1 score representing the worst ranking.

4) Other Measures: There are cases where accuracy measures do not fully reflect the quality of a recommender system (eg. when the user is more interested into novel recommendations such as in the case of music recommendation). For that reason there are two alternative measures that can be considered such as *Novelty* and *Serendipity*.

Novelty expresses the freshness of a recommendation from the user's perspective, i.e., whether recommended items have been suggested before to the user. Several studies [24], [8] give a penalty to recommendations that include popular items since popularity is statistically inversely proportional to novelty.

Serendipity on the other hand measures the element of surprise in recommendations. Randomness is sometimes mistakenly correlated with serendipity, however, it is more broad and does not focus on items that the user is the least expecting to see. In other words, serendipity values more recommendations that don't match the profile of a user. For that purpose recommendations with distance metrics can help to achieve good serendipity scores.

D. Conclusions

Recommendation algorithms can be evaluated in both offline and online tests. Offline tests are generally preferred for general comparisons, however, online tests are more effective on measuring the actual effectiveness of an algorithm upon the behavior of the user. Accuracy measures are common and easy to use in both offline and online evaluations and cover most evaluation needs for recommender algorithms. Apart from accuracy measures novelty and serendipity are also noteworthy since they quantify the innovation in recommendations which can be critical in various use cases.

VII. A CASE STUDY IN NEWS RECOMMENDATIONS

As described in section V, news recommendation and scoring algorithms have to deal with a very special case of items. News articles are not considered persistent items and they are being updated at minimum on a daily basis. Since users seek to read new articles every day a recommender should be able to rank fresh items for each user based on their profile.

Given that this is one of the most challenging use cases of recommender systems, we conducted an experimentally driven analysis of state-of-the-art scoring algorithms and ideas in the context of news ranking.

A. Experimental Study

In our experimental study we used news article user logs collected throughout a span of 17 days from *Sumline AB*, a Swedish commercial news aggregator service during the period of November and December 2014. The data consisted user actions (mainly clicks) on news articles and associated metadata.

In summary, we ran our evaluation for 12000 events spreading over the whole period, with an average of 700 events per day. That dataset was the result of filtering by a single country, Sweden, where most of the active users were coming from.

1) Evaluation Method: Since we conducted this study on existing user logs, we only considered an offline analysis of the selected scoring algorithms. Among the methods described in Section VI we chose two:

A Consistent Simulated Time Evaluation to simulate the performance of each scoring algorithm in the same time intervals that it would be used if it would be running online. We refer this method *incremental*.

A Time Agnostic Evaluation by applying cross-validation over different periods we discretised our data on. This method generally offers lower variance and is generally preferred for evaluating collaborative filtering algorithms. We will refer to this method as *cross* for the rest of the section.

As a *basic period* to discretize our data, we chose a day (24h) since it also corresponds well to the periodicity of the news articles. Our evaluation process for every scoring algorithm and method is summed up in the following steps:

Algorithm	Type	Description (see Section)	Identifier Identifier
Topic-Based (Click Behavior)	Content-Based	III-E	*content*
Topic-Based (Click Behavior) with Trend Influence	Content-Based	III-E	*content_pub*
ALS - Exclusive	Col. Filtering	IV-C3	*als-excl*
ALS - Inclusive	Col. Filtering	IV-C3	*als-incl*
ALS + Topic-based	Hybrid	III-E, IV-C3	*als-content*

TABLE IV: Algorithms considered

1) **Sampling:** Select the training and test datasets as such:
   - For the *incremental* version select all events up to period k to be included in the training set and put all events of period $k+1$ in the test set.
   - For the *cross* version select k random periods of which to create the training dataset and one random period for the test set.

   In some special cases we had to include the test set into the training set as well as we will make clear later while describing the different algorithms.
2) **Training:** Train each scoring algorithm by feeding it all events from the training set given.
3) **Scoring:** Generate a $User \times Item$ matrix as a result of the cartesian product of all distinct users and items in the test set and use the scoring algorithm to predict the score for each combination.
4) **Ranking**: Select the top K articles per user for each algorithm based on their assigned score and compute the True Positive Rate (also known as *Recall* or *Sensitivity*)

We chose to follow this methodology for conducting our offline evaluation since it is suitable for implicit binary ratings (eg. user clicks). By ranking and selecting the top K articles per user we can easily check the predictability of each respected scoring algorithm by just computing the original clicks that are included in each recommended set, thus, eliminating the need to normalise or transpose scores.

2) Algorithms and Implementation: In our evaluation we decided to select and implement the most representative algorithms from each recommender system category along with their alternatives while also exploring the benefits of using a hybrid approach. All algorithms considered in our experimental comparison are summed up in table IV. Among the existing content-based algorithms we chose to implement Google's topic-based algorithm along with its variant that also includes the public trends in the predicted probability distribution function of the recommended topics per user. We have implemented the algorithms in Python using Apache Spark to enable distributed processing. For the ALS variants we used the MLLib implementation that comes with Apache Spark [31]. We considered two variants for ALS, an inclusive and an exclusive version, referring to the inclusion or not of the actions of the test set to the training set. As it was mentioned before collaborative filtering techniques suffer from the bootstrapping problem, thus, making predictions about items that do no exist in the model impossible. Finally, the hybrid algorithm simply selects articles as such: $topK(predicted_{ALS} \times predicted_{content})$.

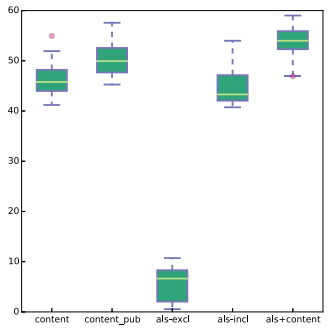

Fig. 4: True positive rate of each scoring algorithm - incremental

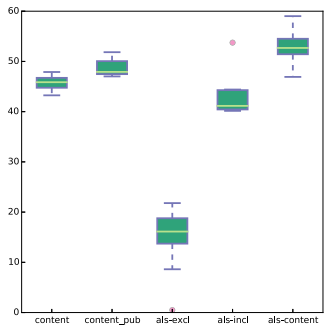

Fig. 5: True positive rate of each scoring algorithm - cross

3) Experiment Setup: Our experimental setup is a single machine with 2.2GHz Intel Core i7 and 16GB Memory. Furthermore, we made the following assumptions: For *content_pub* we used 10 virtual clicks per topic and in the case of ALS we chose to use 5 latent factors and 20 iterations as its default parameters. During the Ranking phase we chose the top 20 articles per user which was enough for evaluating our scores given that the average number of clicks per day in the service is four to five. The evaluation was mainly written using Python Notebook to allow for some decent exploratory data analysis experience. The results of the *incremental* method for each specific algorithm variant can be seen in Fig. 4.

4) Discussion: Among the simple algorithm versions the content-based variant with trend influence seemed to achieve the highest TPR while the worst accuracy was excibited by the exclusive version of ALS, making it clear that collaborative filtering can only make an impact when we have enough data available. The incluse version of ALS did achieve a reasonably good predictive performance, however, by itself it could not beat the predictiveness of topic-based recommendations. Finally, the hybrid version achieved the highest scores when it comes to predictiveness. The improvement over the existing topic-based approach can be clearly seen and explained based on the highly quantized rankings of content-based techniques. When all articles with the same content-id are put together with the same score, upon ranking we would have to include all of such articles together and then select the top K. That would result in most article predictions falling in the same categories for a user, belonging to the top one or two topics

for this user. The collaborative filtering predicted score of ALS can be therefore used to score articles with the same topic id and thus improve the probability to include recommended items from articles that are not part of the top topics for a user. The *cross* variant is depicted in Fig. 5. This variant bears similar characteristics to the incremental one, with only a smaller general variance and a better performance for the exclusive ALS version. That could be credited to the inclusion of subsequent periods than the one contained in the test set each time, some of which containing article click events originating at the test set.

VIII. Conclusion

There are often trade offs over selecting to do content-based or a collaborative filtering-based ranking of items. In our experimental analysis we observed that the optimal true positive rate can only be achieved by a hybrid score obtained by both methods. Of course, collaborative filtering would only be sensible to use when there are enough events at hand. Thus, we could conclude that the optimal case lies in-between. Perhaps an online deployment would be interesting to consider gradually switching towards a hybrid scoring algorithm from purely content based recommenders based on trends. The adaptive inclusion of collaborative filtering scorings to final scorings makes a good application of stream processing and would be an interesting use case for future work.

IX. Acknowledgements

This research was supported in part by Sumline AB via a project funded by VINNOVA (contract 2014-00486). We thank Qi Cao and Magnus Bergman from Sumline AB for giving us opportunity to work in this project, for their support.

References

[1] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. Analyzing user modeling on twitter for personalized news recommendations. In *User Modeling, Adaption and Personalization*, pages 1–12. Springer, 2011.

[2] Mohamed Ahmed, Stella Spagna, Felipe Huici, and Saverio Niccolini. A peek into the future: predicting the evolution of popularity in user generated content. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 607–616. ACM, 2013.

[3] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[4] Roja Bandari, Sitaram Asur, and Bernardo A Huberman. The pulse of news in social media: Forecasting popularity. In *ICWSM*, 2012.

[5] Michael Bendersky, W Bruce Croft, and Yanlei Diao. Quality-biased ranking of web documents. In *Proceedings of the 4th ACM international conference on Web search and data mining*, pages 95–104. ACM, 2011.

[6] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[7] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[8] Òscar Celma and Perfecto Herrera. A new approach to evaluating novel recommendations. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 179–186. ACM, 2008.

[9] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[10] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proc. of the 8th ACM Conference on Recommender systems*, pages 169–176. ACM, 2014.

[11] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[13] Leo Iaquinta, Marco De Gemmis, Pasquale Lops, Giovanni Semeraro, Michele Filannino, and Piero Molino. Introducing serendipity in a content-based recommender system. In *Hybrid Intelligent Systems, 2008. HIS'08. 8th International Conference on*, pages 168–173. IEEE, 2008.

[14] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[15] Evan Kirshenbaum, George Forman, and Michael Dugan. A live comparison of methods for personalized article recommendation at forbes. com. In *Machine Learning and Knowledge Discovery in Databases*, pages 51–66. Springer, 2012.

[16] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181, 2009.

[17] Kristina Lerman and Tad Hogg. Using a model of social dynamics to predict popularity of news. In *Proceedings of the 19th international conference on World wide web*, pages 621–630. ACM, 2010.

[18] Lei Li, Ding-Ding Wang, Shun-Zhi Zhu, and Tao Li. Personalized news recommendation: a review and an experimental investigation. *Journal of Computer Science and Technology*, 26(5):754–766, 2011.

[19] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 31–40. ACM, 2010.

[20] Donald Metzler and W Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[21] Mehran Nadjarbashi-Noghani, Jie Zhang, KMH Sadat, and Ali A Ghorbani. Pens: A personalized electronic news system. In *Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*, pages 31–38. IEEE, 2005.

[22] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[23] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[24] Guy Shani, Max Chickering, and Christopher Meek. Mining recommendations from the web. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 35–42. ACM, 2008.

[25] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[26] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[27] Alexandru Tatar, Panayotis Antoniadis, Marcelo Dias De Amorim, and Serge Fdida. Ranking news articles based on popularity prediction. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 106–110. IEEE Computer Society, 2012.

[28] Alexandru Tatar, Panayotis Antoniadis, Marcelo Dias De Amorim, and Serge Fdida. From popularity prediction to ranking online news. *Social Network Analysis and Mining*, 4(1):1–12, 2014.

[29] Alexandru Tatar, Jérémie Leguay, Panayotis Antoniadis, Arnaud Limbourg, Marcelo Dias de Amorim, and Serge Fdida. Predicting the popularity of online articles based on user comments. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 67. ACM, 2011.

[30] Manos Tsagkias, Wouter Weerkamp, and Maarten De Rijke. News comments: Exploring, modeling, and online prediction. In *Advances in Information Retrieval*, pages 191–203. Springer, 2010.

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[32] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.