# Telemetry-Aware Add-on Recommendation for Web Browser Customization

Full Paper

Martin Lopatka
Mozilla
mlopatka@mozilla.com

Victor Ng
Mozilla
vng@mozilla.com

Ben Paul Miroglio
Mozilla
bmiroglio@mozilla.com

David Zeber
Mozilla
dzeber@mozilla.com

Alessio Pierluigi Placitelli
Mozilla
aplacitelli@mozilla.com

Laura Thomson
Mozilla
lthomson@mozilla.com

## ABSTRACT

Web Extensions (add-ons) allow clients to customize their Web browsing experience through the addition of auxiliary features to their browsers. The add-on ecosystem is a market differentiator for the Firefox browser, offering contributions from both commercial entities and community developers.

In this paper, we present the Telemetry-Aware Add-on Recommender (TAAR), a system for recommending add-ons to Firefox users by leveraging separate models trained to three main sources of user data: the set of add-ons a user already has installed; usage and interaction data (browser Telemetry); and the language setting of the user's browser (locale). We build individual recommendation models for each of these data sources, and combine the recommendations they generate using a linear stacking ensemble method. Our method employs a novel penalty function for tuning weight parameters, which is adapted from the log likelihood ratio cost function, allowing us to scale the penalty of both correct and incorrect recommendations using the confidence weights associated with the individual component model recommendations. This modular approach provides a way to offer relevant personalized recommendations while respecting Firefox's granular privacy preferences and adhering to Mozilla's lean data collection policy.

To evaluate our recommender system, we ran a large-scale randomized experiment that was deployed to 350,000 Firefox users and localized to 11 languages. We found that, overall, users were 4.4% more likely to install add-ons recommended by our ensemble method compared to a curated list. Furthermore, the magnitude of the increase varies significantly across locales, achieving over 8% improvement among German-language users.

## CCS CONCEPTS

• **Information systems** → **Personalization**; **Web applications**; **Recommender systems**; • **Human-centered computing** → *Web-based interaction.*


*UMAP '19, June 9–12, 2019, Larnaca, Cyprus*

ACM ISBN 978-1-4503-6021-0/19/06.
https://doi.org/10.1145/3320435.3320450

## KEYWORDS

Web Browser Personalization, Recomender Systems, Web Extensions, Linear Ensemble

**ACM Reference Format:**
Martin Lopatka, Victor Ng, Ben Paul Miroglio, David Zeber, Alessio Pierluigi Placitelli, and Laura Thomson. 2019. Telemetry-Aware Add-on Recommendation for Web Browser Customization: Full Paper. In *27th Conference on User Modeling, Adaptation and Personalization (UMAP '19), June 9–12, 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3320435.3320450

## 1 INTRODUCTION

Modern Web browsers enable user customization via extensions, which add non-core features or allow a user to personalize the browser to their individual needs. Some of the most popular classes of extensions for the Firefox browser are ad blockers, password managers, and download helpers [36]. As well as benefiting users directly through expanded functionality or convenience, the extension ecosystem provides indirect value to Mozilla in the form of increased user engagement [26] and decreased churn associated with browser customization.

As such, it is desirable to ensure that Firefox users are able to easily discover extensions that are most relevant to them. However, until recently, this task has primarily been up to the users themselves. Extensions (known as "add-ons" in the Firefox environment) are principally available from addons.mozilla.org [36] (abbreviated as AMO), which may be visited as a normal website or from the "Get Add-ons" panel within Firefox's Add-ons Manager (accessible by navigating to about:addons). Extensions are generally discovered by searching this site, although recommendations are sometimes offered in the form of hand-curated lists or collections featured on AMO or around the Web. This has likely presented a barrier to engagement with the extension ecosystem. Despite the wealth of extensions available, we have found that a significant proportion of Firefox users do not have any extensions installed, and of those who do, the vast majority only have very few. Furthermore, the distribution of extensions that are installed is substantially skewed towards the most popular offerings. In this paper, we seek to remedy this situation by developing a multi-faceted system for recommending extensions to users, and demonstrating that providing discoverable, personalized extension recommendations does in fact lead to increased extension installation.

Recommender systems typically determine relevant recommendations on the basis of similarity either between items (content-based filtering) or between users (collaborative filtering). Since extensions have only limited relevant metadata, a collaborative approach is a better fit in our setting. However, collaborative methods are known to suffer in the presence of sparsity, when a majority of users have interacted with only few items or vice-versa. This is often addressed by augmenting the user-item matrix with external information on user preferences to improve the similarity calculation. However, the principal source of user data available to us, Firefox browser Telemetry [35], provides only a limited window into a user's qualitative preferences. In accordance with Mozilla's privacy policy [33], it consists mainly of technical usage measurements, such as the user's computing environment (e.g. OS, hardware, installed add-ons), as well as interaction and usage data, including time active and number of pages loaded. Additionally, Firefox's privacy settings enable granular control over what information is shared via Telemetry. This leads to both sparsity and variability in the range information available for selecting recommendations.

To overcome these challenges, we develop a recommender system for extensions centered on robustness to varying degrees of data sparsity. We adopt a modular approach, implementing multiple recommenders, each tailored to a particular subset of the available data. The first is a traditional collaborative filtering model which uses extension installations as implicit ratings, trained using a standard matrix factorization technique. As a fallback for cases where extension installations are lacking, we employ a novel approach to assess similarity between users in terms of other Telemetry features. We first determine groups of users with "similar interests" by clustering them on their item ratings.

We then compute pairwise similarities between users in terms of their Telemetry features. A likelihood ratio criterion determines which of these similarity values are "large enough" for the purpose of providing recommendations, based on how likely a pair of users are to belong to the same interest cluster. As well as bypassing the issue of choosing a similarity threshold, this approach entails a form of fuzzy matching where the set of similar users may span multiple clusters. A final recommender module offers simple demographic filtering for browser localization (language setting), drawing recommendations from the most popular extensions in the user's locale. To address potential privacy implications for locales with very few users, the extension frequencies are computed in a differentially private manner.

For a given user, recommendations are obtained from each of these component modules and combined using a linear ensemble method. Each component recommender is designed to return confidence weights along with its recommendations, reflecting their relevance. These scores are aggregated across components in the ensemble to produce final scores for each recommendation, the topmost of which are surfaced to the user. The ensemble weights are selected to optimize a penalty function known as the log likelihood ratio cost, which to our knowledge has not previously been used to train a recommender system.

We have implemented this recommender system as a Web service[1] available from the "Get Add-ons" panel of the Firefox Add-ons Manager. To evaluate its performance relative to hand-curated recommendations, we deployed a large-scale randomized experiment which reached 350,000 Firefox users. The results presented in Section 3.3 show improved installation rates, i.e. acceptance of recommendations, in the ensemble study branch compared to both a partially-curated (hybrid) and fully-curated (control) list of add-ons. Earlier results comparing the ensemble strategy against the performance of individual component models also demonstrate superior installation rates for the ensemble branch. This suggests that weaker information leveraged by the component models not utilizing the client extension installations can in fact serve to refine recommendations provided by a standard collaborative filtering method.

## 1.1 Related Work

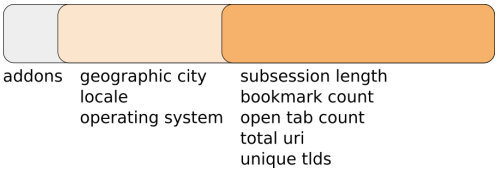
The task of locating extensions is desirable from the perspective of improving the user experience through personalization [58] and increasing the diversity of interactions with the overall ecosystem. However, top-N item recommendation [14] is a nuanced task complicated by both the vastness of the search space and the weak signal present in the implicit measurements available in the Telemetry data.

Collaborative filtering methods [13] leverage the relationship between users and the items to be recommended, in addition to item-item relationships, in order to compute meaningful recommendations for users. User-item relationships can be modelled either through explicit ratings (e.g. numerical scoring) or implicit ones (e.g. was the item ever used?) [37]. While explicit rating is convenient, it might not always be available.

Prior work on alleviating the cold start problem in collaborative filtering has focused on incorporating additional data sources to supplement the standard user-item matrix [50, 51, 55]. This includes cross-domain recommender systems, which tend to draw on ratings expressed by the same users for different types of items [5, 11, 54], and approaches leveraging social media [22, 46]. External data is particularly useful in domains where user-item ratings tend to be sparse, such as recommending apps [2, 52] or locations of interest [8, 12, 53]. Such approaches typically combine all data into a single model. However, in accordance with Mozilla's lean data collection practices, the correlation of independent data sources with browser telemetry is an undesirable solution, as it may erode client privacy. The use of ensembles for improving recommendations has also been explored [1, 14, 19, 45, 48], although these generally apply multiple models to the same or related datasets. Previous collaborative filtering models have also incorporated clustering methods [16, 39, 40, 42], where clusters are used to define the user similarity neighborhoods. Finally, the log likelihood ratio cost metric originated in [3] and is commonly used in forensics [24, 27]. However, as noted above, we are not aware of it having been used previously in training ensemble weights for a recommender system.

[1] https://github.com/mozilla/taar

**Figure 1: Client feature vector**




## 2 METHODOLOGY

In order to determine the features most relevant to recommending extensions, we individually evaluate the predictive power of a large set of Telemetry measurements via an iterative tree classification method, so as to assess univariate efficacy in yielding stable and high precision recommendations [6]. The usual pitfall of threshold selection is mitigated by the fact that the number of features we can realistically incorporate is bounded by practical online performance constraints. We retain the top 8 performing features, in addition to the add-on installation information, yielding the client feature vector depicted in Figure 1.

The retained Telemetry measurements are:

- Categorical features:
  - geographic city: the city closest to the origin IP address (best guess)
  - locale: the application localization identifier based on a combined country/region[18] and language[17] indicator.
  - operating system: the host operating system
- Continuous features:
  - subsession length: the length of the current browser subsession in seconds
  - bookmark count: the number of registered bookmarks saved to the client profile
  - open tab count: the number of open browser tabs
  - total uri: the number of URIs the user has visited
  - unique TLDs: the number of unique TLDs (Top Level Domains) the client has visited [2]
- Add-on features:
  - add-ons installed

More information on these fields, as well as the full set of available measurements, is provided in the Firefox Probe Dictionary [34].

As the add-on ecosystem contains a variety of extensions from a diverse community of contributors, code quality can vary. In order to safeguard the user experience, we restrict recommendations to a manually curated whitelist of 171 add-ons. This approach ensures that only high quality Web Extensions are recommended to clients: whitelisted add-ons have undergone complete code review by Firefox engineers to ensure code quality and compliance with AMO Policies [32]. Additionally, the whitelist will play a key role in the privacy protections employed by the locale recommender described in Section 2.3.

The validation and tuning of our recommendation models is complicated by the fact that ground truth remains unknown for users in the test set. Previous works have focused on developing evaluation metrics for recommender systems in an industrial context [7, 25, 44]. Instead, we adopt the following approach: we select a subset of clients with at least 4 add-ons installed, and for each client, we obfuscate a subset of their add-ons by masking. We can then generate recommendations based on the unmasked add-ons, as well as the other Telemetry features, taking the masked add-ons as ground truth. Recommendations are evaluated as either *correct* or *incorrect* based on their presence in that client's set of masked add-ons. This approach is used for both the feature selection outlined above and parameter tuning of the individual recommender modules described in Sections 2.1–2.4.

### 2.1 Add-on-based Recommender

This recommender module adopts a standard collaborative filtering approach with implicit ratings, where users are assumed to be similar if they have similar add-ons installed. We build a latent factor model using a matrix factorization technique, alternating least squares (ALS) [21], to decompose the user/add-on matrix. This approach is chosen as it proves to be as good as other techniques in the literature, while being scalable to very large datasets by design [56]. Moreover, the ALS algorithm is readily available in off-the-shelf distributed computing packages such as Apache Spark [49].

To compute the model, a master list of valid add-ons is generated by querying the addons.mozilla.org API and filtering according to a set of baseline validity criteria. We then construct a model matrix $M$, setting $M_{i,j} = 1$ if user $i$ has installed add-on $j$ from the master list and 0 otherwise. Since users typically install only few add-ons, this matrix will be sparse. We then map the user-item interactions represented in $M$ to a latent space by decomposing it into the product of two matrices using ALS: one representing how much users like each latent feature, and the other how each feature is relevant to the add-on. Our model is retrained using these steps on a weekly basis.

An important advantage to this approach is that recommendations will be accompanied by a confidence weight quantifying their relevance for the candidate user. We will use this weight when combining this module with other recommendation strategies as discussed in Section 2.4. However, as this model is based purely on the add-ons portion of the user Telemetry data, it has a few shortcomings: new users might receive less interesting recommendations, and new add-ons might not get recommended. Indeed, users must have at least one add-on installed in order to obtain recommendations from this model. Nonetheless, cases where this approach performs poorly can be mitigated when combined with the other recommendation modules in the ensemble.

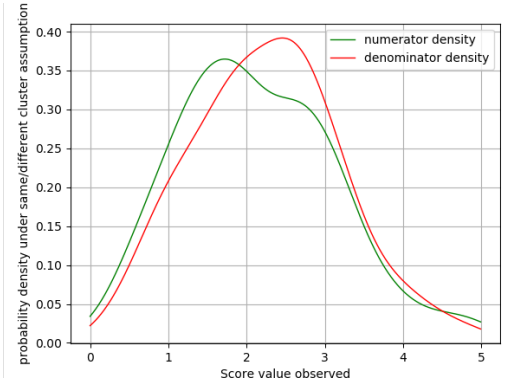
### 2.2 Telemetry-based Recommender

Given a candidate client for which recommendations are sought, the Telemetry-based recommendation module seeks to identify clients which are similar in terms of their non-add-on Telemetry features (Figure 1). We refer to clients with installed add-ons against which candidate clients are compared (via the non-add-on Telemetry features) as *donors*.

A standard approach in this scenario is to determine the $K$ nearest neighbor donors to the candidate client, and to recommend the

[2]No specific website information is ever stored. This value corresponds to the **count** computed on the client, sent via Telemetry as an `Integer` value.

**Figure 2: Probability density of same vs. different cluster membership as a function of pairwise similarity computed on non-add-on features**




donors' add-ons to the candidate. However, in order to ensure diversity in the set of donors surfaced for a candidate, and to calibrate the notion of "sufficiently similar" to the topology of the feature space, we match the candidate with all donors who are "more similar than dissimilar" as determined by a likelihood ratio-based methodology.

In order to obtain a baseline for how close two clients with similar add-on preferences tend to be in terms of their Telemetry features, we begin by grouping donors into distinct clusters based on their installed add-ons. This is accomplished by applying a bisecting K-means clustering algorithm [43] (a form of divisive clustering) to the add-on portion of the client feature vectors. Donor clients belonging to the same cluster are thus deemed to have similar add-on preferences, and conversely for clients in different clusters. Next, we compute similarity scores in the Telemetry feature space between all pairs of clients in a sample of donors. We then pool scores for pairs belonging to the same cluster and for those belonging to different clusters, and represent the inter- and intra-cluster score distributions using kernel density estimates as illustrated in Figure 2. This generalization of pairwise similarity computed for intra-group and inter-group relationships allows us to specify a model for deciding whether a candidate has add-on preferences coinciding with or distinct from a given donor, given their similarity in the non-add-on Telemetry feature space, using likelihood ratios.

We define similarity between two clients in terms of their Telemetry features using the following distance metric:

$$d(x, y) = [c(x, y) + \phi] \cdot h(x', y'),$$

where the $h$ is the Hamming distance [38] between the categorical portions $x'$ and $y'$ of the client feature vectors:

$$h(x', y') = \sum_j |x'_j \neq y'_j|;$$

$c$ is the Canberra distance [38] computed between the continuous features $x$ and $y$ for a client pair as:

$$c(x, y) = \sum_j \frac{|x_j - y_j|}{|x_j| + |y_j|};$$

and $\phi$ is a constant floor adjustment term applied to the continuous portion of the feature vector to prevent 0 values. This use of the $\phi$ parameter explicitly prioritizes similarity among the categorical variables versus the continuous variables in the distance metric.

Given a candidate client requesting recommendations, we compute the distance $d$ between it and all donors, and recommend add-ons pooled across donors which are more likely than not to belong to the same cluster as the candidate. Representing this as a general likelihood ratio (LR) model gives us a very natural quantification of the chances that an add-on a donor has installed may be interesting to a candidate client with a particular similarity to that donor. Add-ons surfaced as recommendations from a particular donor inherit the LR value determined from that donor's similarity with the candidate. Recommendations are then based on the sorted list of add-on/LR pairs, sorted on LR. Additionally, add-on donors are re-sampled weekly, ensuring fresh sampling of the add-ons ecosystem, and allowing the possibility of new pattern discovery and the inclusion of new add-ons in the recommendation pool.

Under our model, donors which are most similar to the candidate in terms of $d$ have a higher likelihood of their add-ons surfaced as recommendations, thereby ensuring their relevance. However, our method has two important advantages over a standard clustering or nearest neighbors approach. We have avoided the issue of selecting a threshold for donors being "similar enough", instead learning an appropriate criterion from the data itself. Also, we have provided for enhanced diversity by fuzzy-matching candidates with the donor clusters: rather than getting assigned to a single one of our add-on clusters, a candidate is matched with all donors which could conceivably belong to the same cluster as the candidate, regardless of which clusters the donors themselves belong to.

## 2.3 Locale-based Recommender

A number of add-ons offer functionality specific to the user's linguistic locale, [17, 18] such as local service interaction or language assistance. It is thus no surprise that, in the feature-selection phase of our analysis, we found browser locale to be a highly effective predictor of add-on preference. Additionally, locale is a property determined on browser installation rather than by user interaction, meaning that a locale-based recommender allows us to surface relevant recommendations even for users with limited Telemetry usage information.

To compute recommendations, the TAAR Locale recommender generates a table of add-on installation frequency counts by locale. The counts are converted to relative frequency weights, and recommendations are selected as the add-ons with the top $K$ highest weights. Since a number of locales and add-ons have very few installs, publishing top-$K$ lists may pose a risk to the privacy of users with rare locale/add-on combinations. To mitigate such risks, we compute the initial frequency counts in a differentially private [9, 10] manner.

Downstream computation of recommendations will then inherit the privacy guarantees, provided it does not rely on private data beyond the protected frequency counts. The Laplace Mechanism [9] is an established method for releasing a table of frequency counts while preserving $\epsilon$-differential privacy. We adapt this technique to generate add-on installation frequency tables for each locale according to the following procedure:

- Limit each client to contributing $m$ of their whitelisted installed add-ons to the frequency counts
- Using the limited data, compute the frequency counts for each add-on in each locale
- Generate noisy counts by locale for each add-on in the whitelist by adding independent Laplace($m/\epsilon$)-distributed noise to the raw counts
- Report the add-ons per locale together with their noisy frequency counts

In the Laplace mechanism, the amount of noise required to achieve a given privacy guarantee depends on the amount by which a single user can influence the outcome of the query. The threshold $m$ determines the tradeoff between bias and variance in generating the privacy-preserving counts, and can be tuned to optimize their accuracy. As most users tend to have few add-ons installed, this is in fact achieved using the stringent limit of only 1 randomly selected reported add-on per client.

Note that the use of the whitelist, as described in Section 2, is central to our use of the Laplace mechanism for this problem. While clients' add-ons are restricted to those in the whitelist on the one hand, we report noisy counts for every whitelisted add-on on the other, even those which no client had installed. Since the whitelist is determined in advance, independently of the per-locale frequency counts, we avoid the additional privacy cost of discovering the list of frequently installed add-ons itself from the private data.

This process yields a simple data structure of recommendations per locale sorted by their weights, which are computed from the privacy-preserving frequency counts:

```
{'zh-CN': [('guid_01', 0.75),...,('guid_02', 0.05)],
 'fr-FR': [('guid_03', 0.24),...,('guid_04', 0.01)],
 ...,
 'en-US': [('guid_04', 0.18),...,(guid_05', 0.02)]}
```

One implication of generating the recommendations based on the noisy counts is that the system may recommend add-ons which were not actually installed by any client in the corresponding locale. However, the weights for such add-ons will either by dwarfed by those of the most frequent ones, or will be approximately uniformly distributed if no add-ons occurred particularly frequently in the locale (or if the locale has very few users). Thus, such recommendations will be effectively muted by the subsequent ensemble module. In this manner, we provide a carefully calculated balance between utility and privacy, even for locales with few clients.

## 2.4 Ensemble Recommender

The ensemble method proposed herein is an implementation of a hybrid sort. Unlike a conventional Linear Stacking [47] wherein individual base models operate over a common feature space but yield different recommendations depending on the statistical characteristics of the models, our base models leverage different subsets of the available Telemetry fields. For example, the add-ons-based recommender described in Section 2.1 is included as a base model, but it is only applicable to a particular subset of the feature space: it may only contribute recommendations when the candidate client has other add-ons installed.

Ensemble methods typically optimize their relative model weights based on an objective function such as MAP (Mean Average Precision) [23] for comparable ranked retrieval problems [21, 41, 47] or RMSE (Root mean square error) [7]. This is usually measured over a ground-truth set of known (client, recommendation) tuples. However, this approach is hindered in our setting by the fact that the distribution of overall add-on installation rates is severely skewed on one hand (popular add-ons are installed much more frequently that niche ones), and the fact that clients typically install very few add-ons on the other. Indeed, both MAP and RMSE metrics lead to poor convergence and substantial bias towards the most popular add-ons. In order to achieve generalizable ensemble weights, we instead establish a novel metric centered on the *confidence weight* of a recommendation, which are returned alongside recommendations by each of our base recommenders.

The log likelihood ratio cost (cLLR) is a metric developed for calibrating likelihood ratio curves [3] which captures the gradient of a set of likelihood ratios derived from test data. It is defined as

$$C_{LLR} = \frac{1}{2}\left[\frac{1}{N_C}\sum_{i=1}^{N_C}\log_2\left(1+\frac{1}{W_i^C}\right)+\frac{1}{N_E}\sum_{j=1}^{N_E}\log_2\left(1+W_j^E\right)\right],$$

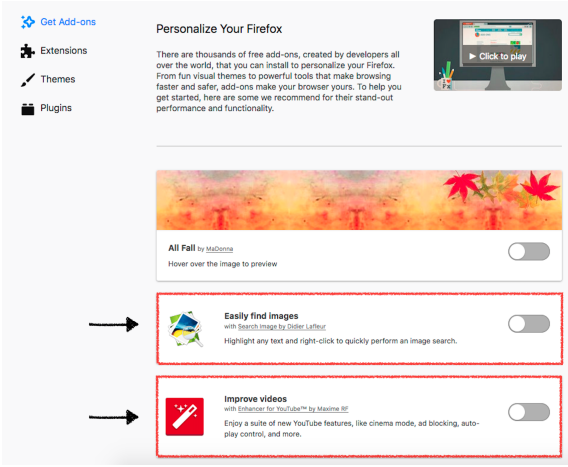
where $N_C$ is the number of correct recommendations provided by a component model in the top $k$ ranked recommendations and $W_i^C$ is the confidence weight attached to the $i$-th correct recommendation produced. Likewise, $N_E$ and $W_j^E$ are the number and confidence weights of erroneous recommendations among the top $k$ surfaced recommendations.

Substituting the normalized confidence weights generated by each of the component models for true likelihood ratios enables the evaluation of a set of recommendations in terms of quality by the cLLR metric, such that a low cLLR indicates a good recommendation set, while a higher cLLR indicates a high cost for error. When ranked recommendation confidence weights are set to equal 1.0, the cLLR score correlates to the precision. However, when weighted recommendations are available, the cLLR penalizes scores which provide strong support for incorrect recommendations and minimizes the penalty when low confidence recommendations are erroneously surfaced. In this way, additional information pertaining to recommendation confidence is leveraged when compared to ranking quality metrics [20]. Using the cLLR as the penalty function, we determine a set of optimal model ensemble weights over the base models using scikit-learn's grid-search implementation [38] passing the cLLR as a custom performance metric. Manual observation of intermediate recommendation results show satisfactory recommendation performance when high confidence and low popularity add-ons are included in the top $k$ ranked recommendations.

## 2.5 Implementation

Each of the three recommender models are precomputed using Apache Spark, and stored in JSON format in Amazon S3. They are updated on a weekly basis to incorporate the latest Firefox client Telemetry data. As these data models do not require low latency access, they are simply loaded and stored in their entirety in memory at process start up. All of the Telemetry data required for generating recommendations for a candidate client is housed in Amazon DynamoDB, indexed by client ID. This datastore is updated

**Figure 3: The "Get Add-ons" section of about:addons**




on a daily basis so that TAAR has at most 24 hour old data to use for add-on recommendation.

The recommendations themselves are surfaced in "Get Add-ons" section of the Firefox built-in about:addons page. When a user navigates to this page, the front-end sends a message to the TAAR server with the user's client ID. The ID is used to retrieve the client's Telemetry from DynamoDB, which is then passed as input to the recommendation service. The server responds to the front-end with a list of 10 recommendations identified by add-on ID, and these results are finally rendered in a user-friendly way within the "Get Add-ons" page. Since Telemetry data for all users is already present on Mozilla's servers, the only piece of information transmitted to the TAAR service is a hashed client ID, reducing latency and supporting Mozilla's lean data policies.

## 3 EVALUATION

In this section we present the results from the most recent experiment we deployed, designed to highlight the role that TAAR plays in the add-on discovery experience. Past experiments we have run using similar designs demonstrated positive results. Most notably, we conducted a preliminary study to assess the performance attained when information coming from non-add-on Telemetry features was used, via the ensemble method, to refine recommendations provided by a collaborative filter method. The ensemble branch showed a 0.8% increase in users' overall add-on installation probability, and a 1.2% increase for non-en-US users, thus informing our decision to move forward with the current design.

The experiment we discuss here best reflects the "production-ready" implementation of TAAR. A key distinction from past experiments is the limited pool of 171 possible add-ons TAAR can recommend to the user due to the use of the whitelist, which we discuss further in Section 3.2.

### 3.1 Shield

The TAAR experiment described below was implemented and distributed using Mozilla's Shield service. Shield [28] is an internal user testing platform that allows for the evaluation of new features through statistical randomized experiments, referred to internally as Shield Studies [29]. Common applications of Shield Studies include changing preferences, displaying messaging or distributing surveys. Firefox ships with a system add-on, the *shield-recipe-client*, which receives instructions from the Shield server [31] and loads a study if the client meets the study-specific targeting criteria. Shield allows for opt-in and opt-out studies, depending on the type of data collection involved [30].

### 3.2 Experiment Design

We tested the efficacy of TAAR on real users using an opt-out Shield Study. The study is limited to new users, to focus on those who are less likely to have preconceptions of add-ons, and to better understand a user's entry into the add-ons ecosystem. We consider a new user to be one whose profile was created between 21 and 2 days before the study enrollment date.

The study follows a between-subjects design with 3 cohorts. Upon navigating to the about:addons page, we show users a cohort-dependent list of 4 add-ons generated by the following processes:

- *control*: Manually curated list of add-ons based on a user's locale, browser version, and other high-level browser characteristics. This is the standard, non-experimental experience.
- *ensemble*: Weighted combination of all eligible models, as described in Section 2.4. Each of the 4 displayed add-ons is a recommendation from the TAAR service.
- *hybrid*: Identical to the *ensemble* cohort, with $C$ curated add-ons interleaved among the recommended add-ons. Here, $C \in \{1, 2, 3, 4\}$ and is selected uniformly at random. This is an attempt to simulate add-on promotional campaigns, which is common practice for the about:addons page. (The *ensemble* cohort has $C = 0$.)

Each user in the study is randomly assigned to one of the three possible cohorts with probabilities of 0.25, 0.25 and 0.5 for the *control*, *ensemble* and *hybrid* branches, respectively. The *hybrid* branch has a higher sampling weight to allow for sufficient quantities of each observable value for $C$. We do not include analysis for values of $C$ in this paper, but rather focus on comparing the *ensemble* cohort to the *control* and *hybrid* cohorts.

The about:addons URL is stored as a browser preference and is referenced each time the page is requested. We changed this URL for users in the *hybrid* and *ensemble* branches to direct requests to the TAAR service, inserting personalized recommendations into the relevant page elements highlighted in Figure 3. When a user with the altered URL requests the about:addons page, TAAR generates and serves recommendations that are rendered to the user in approximately 70 milliseconds.

We localized all add-on content for 11 different locales to better understand TAAR's performance in different regions and contexts, which required us to limit the pool of available recommendations to a whitelist of 171 add-ons. Since the standard about:addons page has localized content, it is imperative that the treatment pages do as well, in order to keep the user experience uniform across cohorts. While the whitelist reduced the cost of the localization tasks by design, it also provided security assurances, since each add-on in the list was hand-picked and verifiably non-malicious.

**Figure 4: Popup in English**

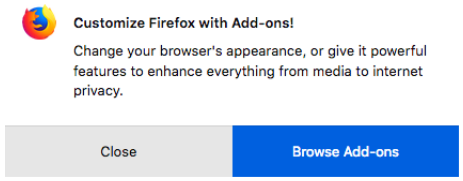



**Table 1: Number of Unique Users per Cohort after Validation**

| Cohort ($c$) | Unique Users ($n_c$) |
|---|---|
| control | 152, 554 |
| ensemble | 101, 734 |
| hybrid | 203, 585 |
| Total | 356, 365 |

To augment traffic to the about:addons page, and to ensure more of the study sample is exposed to one of the page variants, we displayed the popup shown in Figure 4, linking to the about:addons page through the "Browse Add-ons" button. Once entered into the study, users saw this messaging after three successful page loads. A user can, of course, navigate to this page organically by typing "about:addons" in the address bar, or by clicking the *Add-ons* button from the browser menu.

*3.2.1 Cohort Validation.* Due to the nature of the experiment design, we must restrict our study sample to users who navigated to the about:addons page at least once; otherwise the user was never exposed to the treatment or the control. Additionally, we must verify that users in the *hybrid* and *ensemble* branches were successfully served TAAR recommendations, since in the case of failure they are served the standard (control) about:addons page to avoid user experience breakages. We accounted for this by extracting the list of users that are present in the TAAR application logs and associated with a successful recommendation entry from either the hybrid or ensemble model, and dropping users from the study that *ever* experienced a failure (as they would have seen both the treatment and the control). Table 1 shows the resulting cohort sizes for each cohort $c$, after imposing these restrictions on our study sample, denoted by $n_c$. Note that the initial sampling weight structure (0.25/0.25/0.5) is not preserved due to the additional filtering criteria for the cohorts exposed to TAAR, although the *hybrid* and *ensemble* cohorts approximately keep their intended 2:1 (0.5/0.25) ratio.

## 3.3 Results

We compare the estimated probability of installing an add-on $\hat{\pi}_c$ for each cohort $c$, across the three cohorts, where $\hat{\pi}_c = a_c/n_c$, $a_c$ being the number of users in cohort $c$ that installed an add-on from the about:add-ons page. The recommendation page seen in Figure 3 displays precisely 4 recommendations per visit. We favour the rank-agnostic performance metric $\hat{\pi}_c$ as it directly measures the success of this recommender system in a realistic production context where all recommendations are given prominent visibility.

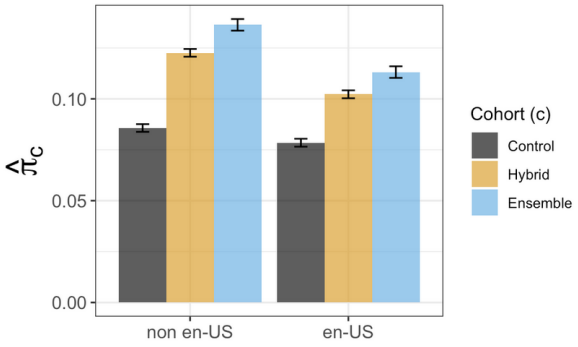
**Table 2: $\hat{\pi}_c$ by Cohort**

| Cohort ($c$) | $\hat{\pi}_c$ |
|---|---|
| control | 0.0829 |
| hybrid | 0.1141 |
| ensemble | 0.1266 |

**Table 3: Proportion Test Results for $\hat{\pi}_c$ with 95% Confidence Intervals**

| Comparison ($c$) | $\Delta_c$ | CI low | CI high | p-value |
|---|---|---|---|---|
| control | 0.0438 | 0.0413 | 0.0462 | $\approx 0$ |
| hybrid | 0.0125 | 0.01 | 0.015 | $\approx 0$ |

**Figure 5: $\hat{\pi}_c$ by Cohort and Locale with 95% Confidence Intervals**

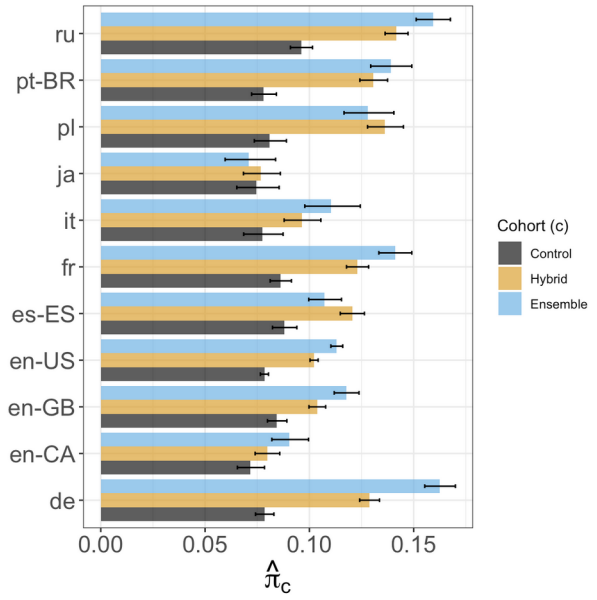



Since $\hat{\pi}_c \in [0, 1]$, we ran a two-sided proportion test comparing $\hat{\pi}_c$ for the *ensemble* cohort against the *hybrid* and *control* cohorts, where $\Delta_c = \hat{\pi}_{\text{ensemble}} - \hat{\pi}_c$ and represents the expected change in add-on installation probability for the *ensemble* cohort compared to cohort $c$. When considering $\hat{\pi}_c$, pairwise two-sided proportion tests offer more interpretable and granular results than some of the alternatives (i.e. a Chi-Square Test for Independence).

We observe a positive effect that is statistically different from 0 for the *ensemble* cohort compared to the other two cohorts as seen in Table 3, implying that the ensemble model is surfacing more relevant add-ons than the hybrid model and the standard curated list. Additionally, we compare the same effects restricted to users with an `en-US` locale, and users with non-`en-US` locales. Users with an `en-US` locale make up 45% of our study sample. Figure 5 demonstrates that the positive effects we observe are greater in magnitude for users in non-`en-US` locales. The corresponding values for $\Delta_{\text{control}}$ are 5.1% for non-`en-US` and 3.5% for `en-US`. Figure 6 further breaks down $\hat{\pi}_c$ by individual locales, restricting $a_c$ and $n_c$ to the subset of users in the relevant locale.

The *ensemble* cohort has a statistically significant, positive effect over the *control* cohort in 91% of locales, and a positive effect over both the *control* and *hybrid* cohorts in 45% of locales. We observed only one null effect over the control for the Japanese locale (`ja`), while Russian (`ru`) and German (`de`) locales notably stand out with

**Figure 6: $\hat{\pi}_c$ by Cohort and Locale with 95% Confidence Intervals**




$\Delta_c$ values of 6.3% and 8.4% respectively. These summary stats, coupled with Figure 6, demonstrate the performance consistency of TAAR across different languages and regions.

## 4 DISCUSSION

The Telemetry-based model described in Section 2.2 entails an additional set of weakly held assumptions regarding the use of unlabeled data at the stage of divisive clustering. Unsupervised clustering of a set of clients based exclusively on add-on installations is expected to ensure diversity sampling in terms of the **add-ons** represented. This approach does not address client characteristic biases present in the non-add-on portion of the client feature vectors. Furthermore, the use of cluster membership labels as a surrogate for client similarity in the Telemetry space implies an independence between the Telemetry features space and the particular set of add-ons installed per client on the one hand, while the later generalization of intra- and extra-cluster similarity across these assumes that a dependence exists on the other.

Finally, it is worth discussing the effects of widespread use of recommender systems in terms of the ecosystem diversity. An increase in the installation rates of recommended add-ons may in turn lead to a higher likelihood of the recommendation of those same add-ons. These polarizing effects may be further exacerbated by a situation where variable information is available for candidate recommendation items [15]. This necessitates that particular attention is placed on ensuring diversity sampling to ensure that new add-ons are promoted in the ecosystem.

### 4.1 Future Work

Prior to the experimental launch of the recommendation system described here, a curated list of add-ons was surfaced on each visit to the Firefox about:addons page. As such, interaction information pertaining to add-on installation events and add-on discovery was unavailable. Future versions of this system may now draw on anonymous interaction data available from the application logs as a more realistic *ground truth* for tuning ensemble weight parameters. Actual individual add-on recommendation and installation/non-installation event data shall be substituted for the surrogate approach described in Section 2.

The ensemble approach may be further refined by the additional parametrization of weights applied to the individual features. The evaluation of a feature-weighted stacked ensemble [47] may be of interest to further refine recommendation quality. Optimizing on feature weights at the level of recommendation model contribution may also allow a meta analysis of feature utility. This has the potential to further reduce the number features required to provide a personalized experience.

The results described herein demonstrate a proof-of-concept implementation, deployed over a subset of the Firefox population. In order to achieve stable and long-term production deployment, additional infrastructure is needed to ensure stability and security while rigorously safeguarding user privacy [4, 57].

## 5 CONCLUSIONS

A recommender system is developed and demonstrated (operating at scale) to yield high value add-on recommendations to users of the Mozilla Firefox web browser. Only existing data sources are used, in alignment with a lean data collection policy, and no personally identifying information is needed to provide recommendations.

The weighted ensemble of component models is demonstrated to provide recommendations better than a curated list, even in cases where clients have opted to limit the information they choose to share with Mozilla via browser Telemetry preferences.

The results of a randomized controlled trial show a significant improvement in the probability of add-on installation in both en-US localized populations and non-English localized browsers. In both cases, the ensemble method outperforms both control conditions, a partially- and fully- curated list of feature add-ons, which is the current Firefox add-ons discovery experience.

## ACKNOWLEDGMENTS

The authors acknowledge and thank Florian Hartmann who completed a research internship at Mozilla, during which he contributed prototype analyses aiding the TAAR project's later development.

We would also like to thank the numerous individuals who participated in code review and coordination in building the TAAR system. In particular: the Add-ons team for immense support and camaraderie, the Engineering Operations team for assisting with scaling our deployment, the Quality Assurance team for their rigorous testing of early prototypes, and the community contributors who participated in the localization of content.

Finally, the authors thank all the Firefox users who continues to trust us with their data by enabling extended Telemetry. Special thanks to all the Firefox users who's contribution of extended data collection via participation in studies allows us to do the best job we can at prototyping and refining Firefox features.

# REFERENCES


[1] Ariel Bar, Lior Rokach, Guy Shani, Bracha Shapira, and Alon Schclar. 2013. Improving simple collaborative filtering models using ensemble methods. In *International Workshop on Multiple Classifier Systems*. Springer, 1–12.

[2] Matthias Böhmer, Lyubomir Ganev, and Antonio Krüger. 2013. Appfunnel: A framework for usage-centric evaluation of recommender systems that suggest mobile applications. In *Proceedings of the 2013 international conference on Intelligent user interfaces*. ACM, 267–276.

[3] Niko Brümmer and Johan du Preez. 2006. Application-independent evaluation of speaker detection. *Computer Speech & Language* 20, 2 (2006), 230 – 275. Odyssey 2004: The speaker and Language Recognition Workshop.

[4] Joseph A. Cal, Ann Kilzer, Arvind Narayanan, Edward W. Felten, and Vitaly Shmatikov. 2011. You Might Also Like: Privacy Risks of Collaborative Filtering.

[5] Xuezhi Cao and Yong Yu. 2017. Joint User Modeling Across Aligned Heterogeneous Sites Using Neural Networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 799–815.

[6] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46. https://doi.org/10.1177/001316446002000104

[7] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-N recommendation tasks. In *RecSys'10 - Proceedings of the 4th ACM Conference on Recommender Systems*. 39–46.

[8] George Drosatos, Pavlos S Efraimidis, Avi Arampatzis, Giorgos Stamatelatos, and Ioannis N Athanasiadis. 2015. Pythia: A privacy-enhanced personalized contextual suggestion system for tourism. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, Vol. 2. IEEE, 822–827.

[9] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[10] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and TrendsÂő in Theoretical Computer Science* 9, 3âĂŞ4 (2014), 211–407. https://doi.org/10.1561/0400000042

[11] Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. 2012. Cross-domain recommender systems: A survey of the state of the art. In *Spanish Conference on Information Retrieval*. sn, 24.

[12] Huiji Gao, Jiliang Tang, and Huan Liu. 2015. Addressing the cold-start problem in location recommendation using geo-social correlations. *Data Mining and Knowledge Discovery* 29, 2 (2015), 299–323.

[13] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.

[14] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (Dec. 2015), 19 pages.

[15] Ido Guy, Sigalit Ur, Inbal Ronen, Adam Perer, and Michal Jacovi. 2011. Do you want to know?: recommending strangers in the enterprise. In *In Proceedings of the ACM Conference on Computer supported Cooperative Work (CSCW*.

[16] Mei-Hua Hsu. 2008. A personalized English learning recommender system for ESL students. *Expert Systems with Applications* 34, 1 (2008), 683–688.

[17] ISO639-1:2011(E) 2011. *Codes for the Representation of Names of Languages*. Standard. International Organization for Standardization, Geneva, CH.

[18] ISOISO3166/MA 2013. *Codes for countries and their subdivisions*. Standard. International Organization for Standardization, Geneva, CH.

[19] Michael Jahrer, Andreas Töscher, and Robert Legenstein. 2010. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 693–702.

[20] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446. https://doi.org/10.1145/582415.582418

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. https://doi.org/10.1109/MC.2009.263

[22] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2013. Addressing cold-start in app recommendation: latent user models constructed from twitter followers. In *Conference: The 36th international ACM SIGIR conference on Research and development in information retrieval*. 283–292.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[24] Didier Meuwly, Daniel Ramos, and Rudolf Haraksim. 2017. A guideline for the validation of likelihood ratio methods used for forensic evidence evaluation. *Forensic science international* 276 (2017), 142–153.

[25] Frank Meyer, Françoise Fessant, Fabrice Clerot, and Eric Gaussier. [n. d.]. Toward a New Protocol to Evaluate Recommender Systems. In *CEUR Workshop Proceedings, co-located with ACM RecSys 2012* (2012).

[26] Ben Miroglio, David Zeber, Jofish Kaye, and Rebecca Weiss. 2018. The Effect of Ad Blocking on User Engagement with the Web. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 813–821. https://doi.org/10.1145/3178876.3186162

[27] Geoffrey Stewart Morrison. 2011. Measuring the validity and reliability of forensic likelihood-ratio systems. *Science & Justice* 51, 3 (2011), 91–98.

[28] Mozilla. 2017. Firefox Shield. Retrieved January 21, 2019 from https://wiki.mozilla.org/Shield

[29] Mozilla. 2017. Firefox Shield Studies. Retrieved January 21, 2019 from https://wiki.mozilla.org/Firefox/Shield/Shield_Studies

[30] Mozilla. 2017. Firefox/Data Collection. Retrieved January 21, 2019 from https://wiki.mozilla.org/Firefox/Data_Collection

[31] Mozilla. 2017. Normandy. Retrieved January 21, 2019 from http://normandy.readthedocs.io/en/latest/

[32] Mozilla. 2018. AMO Policies. Retrieved January 21, 2019 from https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/AMO/Policy

[33] Mozilla. 2018. Mozilla/PrivacyPolicy. Retrieved January 21, 2019 from https://www.mozilla.org/en-US/privacy/

[34] Mozilla. 2018. Mozilla/Probe Dictionary. Retrieved May 4, 2018 from https://telemetry.mozilla.org/probe-dictionary/

[35] Mozilla. 2018. Telemetry. Retrieved January 21, 2019 from https://wiki.mozilla.org/Telemetry

[36] Mozilla. 2019. Add-ons for Firefox. Retrieved January 21, 2019 from https://addons.mozilla.org

[37] Douglas W Oard, Jinmook Kim, et al. 1998. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*. Menlo Park, CA: AAAI Press, 81–83.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[39] Andre Luiz Vizine Pereira and Eduardo Raul Hruschka. 2015. Simultaneous co-clustering and learning to address the cold start problem in recommender systems. *Knowledge-Based Systems* 82 (2015), 11–19.

[40] Manh Cuong Pham, Yiwei Cao, Ralf Klamma, and Matthias Jarke. 2011. A clustering approach for collaborative filtering recommendation using social network analysis. *J. UCS* 17, 4 (2011), 583–604.

[41] Sam Reid and Greg Grudic. 2009. Regularized Linear Models in Stacked Generalization. In *Multiple Classifier Systems*, Jón Atli Benediktsson, Josef Kittler, and Fabio Roli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 112–121.

[42] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2002. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, Vol. 1. 291–324.

[43] Sergio M. Savaresi and Daniel L. Boley. 2001. On the performance of bisecting K-means and PDDP. In *Proceedings of the First SIAM International Conference on Data Mining (ICDM-2001*. 1–14.

[44] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. 2011. Setting Goals and Choosing Metrics for Recommender System Evaluations. 811 (01 2011).

[45] Jiwan Seo, Seungjin Choi, Mucheol Kim, and Sangyong Han. 2013. The Method of Personalized Recommendation with Ensemble Combination. *JoWUA* 4, 4 (2013), 108–121.

[46] Bracha Shapira, Lior Rokach, and Shirley Freilikhman. 2013. Facebook single and cross domain data for recommendation systems. *User Modeling and User-Adapted Interaction* 23, 2-3 (2013), 211–247.

[47] Joseph Sill, Gábor Takács, Lester W. Mackey, and David Lin. 2009. Feature-Weighted Linear Stacking. *CoRR* abs/0911.0460 (2009). arXiv:0911.0460 http://arxiv.org/abs/0911.0460

[48] Bruno Souza Cabral, Renato Dompieri Beltrao, Marcelo Garcia Manzato, and Frederico Araújo Durão. 2014. Combining multiple metadata types in movies recommendation using ensemble algorithms. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*. ACM, 231–238.

[49] Apache Spark. 2018. ApacheSpark/ALS. Retrieved January 17, 2018 from https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html

[50] K Tanaja, Koichi Hori, and Masato Yamamoto. 2010. Development of a Recommender System based on Extending Contexts of Content and Personal History. *Journal of Emerging Technologies in Web Intelligence* 2, 3 (2010), 197–203.

[51] Maria Trujillo, Marta Millan, and Edward Ortiz. 2007. A recommender system based on multi-features. In *International Conference on Computational Science and Its Applications*. Springer, 370–382.

[52] Xiao Xia, Xiaodong Wang, Xingming Zhou, and Tao Zhu. 2014. Collaborative recommendation of mobile apps: A swarm intelligence method. In *Mobile, Ubiquitous, and Intelligent Computing*. Springer, 405–412.

[53] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. 2011. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 325–334.

[54] Qian Zhang, Dianshuang Wu, Jie Lu, Feng Liu, and Guangquan Zhang. 2017. A cross-domain recommender system with consistent information transfer. *Decision Support Systems* 104 (2017), 49–63.

[55] Zi-Ke Zhang, Chuang Liu, Yi-Cheng Zhang, and Tao Zhou. 2010. Solving the cold-start problem in recommender systems with social tags. *EPL (Europhysics Letters)* 92, 2 (2010), 28002.

[56] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. *Lecture Notes in Computer Science* 5034 (2008), 337–348.

[57] Xue Zhu and Yuqing Sun. 2016. Differential Privacy for Collaborative Filtering Recommender Algorithm. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics (IWSPA '16)*. ACM, New York, NY, USA, 9–16.

[58] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving Recommendation Lists Through Topic Diversification. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*. ACM, New York, NY, USA, 22–32.