Kibitz: A Framework for Creating Recommender Systems

by

Brian John Sun

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author ................................................................

Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by .........................................................

David R. Karger
Professor
Thesis Supervisor

Accepted by ...........................................................

Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses
Kibitz: A Framework for Creating Recommender Systems

by

Brian John Sun

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 2017, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Recommender systems are one of the most vital and ubiquitous parts of the modern web. They are used by many major internet services such as Facebook, Google, and Amazon. However, there is a wealth of content and data that remains untapped by mainstream commercial recommender systems. We have designed and implemented Kibitz, a framework that allows anyone to create a recommender system on top of an arbitrary collection of items. We have developed a web application that facilitates the creation, customization and deployment of standalone websites for browsing and rating items as well as receiving item recommendations. We have also created a set of libraries for embedding rating and recommendation functionality into other websites. Partnering with local bookstores, we evaluated the process of using Kibitz to build recommender systems for their communities.

Thesis Supervisor: David R. Karger
Title: Professor
Acknowledgments

I would like to thank Professor David Karger for his support, guidance, and mentorship throughout the duration of this project; Amy Zhang, Quanquan Liu, and everyone from the Haystack group for their support and inspiration; Justin Anderson, Albert Carter, and everyone working on the Datahub project for their constant help and advice. Lastly, I want to thank my family and friends for their love.
# Contents

1 Introduction 11

2 Related Work 15
   2.1 Recommender Systems ............................... 15
   2.1.1 Relevant Projects ................................. 16
   2.1.2 Challenges in Building a Recommender System .......... 17

3 Kibitz 19
   3.1 End Users ........................................... 20
   3.2 Creators ............................................. 23
      3.2.1 Main Page ....................................... 23
      3.2.2 Recommender Creation ........................... 24
      3.2.3 Main Settings and Customization .................. 28
      3.2.4 Miscellaneous Settings .......................... 34
   3.3 Widget Libraries ................................... 36
   3.4 Kibitz Version One ................................ 38

4 Implementation and Architecture 43
   4.1 Datahub .............................................. 43
   4.2 Kibitz Server ....................................... 44
      4.2.1 Technologies Used ............................... 44
      4.2.2 Application Architecture ........................ 44
   4.3 Front End Applications ............................. 46
5 User Studies
5.1 Bookstore Owners
  5.1.1 Interviews
  5.1.2 User sketches
  5.1.3 Tasks
  5.1.4 Post-Task Interviews
5.2 End User Studies

6 Future Work and Conclusion
6.1 Item Categories to Explore
6.2 Conclusion

A User Interview Questions, Tasks, and Surveys
A.1 Creator Interview
A.2 Creator Post-Task Survey
A.3 End User Survey
List of Figures

3-1 End User Home Page ................................................. 20
3-2 Text Based Item View ............................................. 21
3-3 Sort/Filter Side Nav ............................................. 22
3-4 Creator Login Page .............................................. 23
3-5 Creator Home Page ................................................ 24
3-6 Recommender Creation Modal ............................... 25
3-7 Creation Modal, CSV File Upload .......................... 26
3-8 Creation Modal, Datahub ........................................... 27
3-9 Settings and Customization Page, General Tab ............. 28
3-10 Templating Tab Page ............................................ 29
3-11 Item Template Popup ........................................... 30
3-12 Rating Template Popup .......................................... 30
3-13 Template Design, List Display .............................. 31
3-14 Rating Icon Selection .......................................... 32
3-15 Item Field Selection GUI ..................................... 33
3-16 CSV Upload Tab .................................................. 34
3-17 Creation Flow on Landing Page ............................. 39
3-18 Creation Flow on Home Page ................................. 40
3-19 Landing Page with Overlays .................................. 40
3-20 Original Kibitz Item Display ................................. 41

5-1 Design 1 .............................................................. 51
5-2 Design 2. Interactive Item Similarity Map .................. 53
Chapter 1

Introduction

Recommender systems are a vital part of the modern web. Many services incorporate some form of content recommendation. Facebook recommends posts in a user’s News Feed, Youtube recommends videos based on a user’s viewing history, Netflix recommends movies, and Amazon recommends products. However, there is a wealth of data that remains untouched by mainstream commercial recommender systems. Creating a recommender system is costly, time consuming, and usually involves some amount of development resources. Smaller populations and communities that do not have these resources will not be able to enjoy the benefits of using a recommender system.

The central goal of this project is to explore making recommendation systems more easily and readily available to a wider audience, particularly to an audience that does not have the time or resources to develop software applications.

There are many software packages for implementing a recommendation engine. Examples include LensKit [2], Apache Mahout [3], and Spark MLlib [7]. However, these tools require significant programming experience. Users of these packages are essentially developing their own software applications from scratch. This is akin to using a web framework like Django [8] to create a personal website. This is not necessarily a viable option for every user. We hypothesize that there are many users with no programming experience who have data they want to use to build a recommender system.
To address this, we have built Kibitz, a framework which allows anyone to create a standalone website for browsing and rating any collection of items and receiving item recommendations. It requires no development knowledge or expertise and works with any given collection of items. A user can customize and style the website according to a preexisting set of parameters as well as CSS.

The standalone recommender website provides item browsing and rating functionality, sort and filter options, full-text search, the ability to hide non-relevant items, and two different layouts, image grid and text-based.

To create such a website, users can upload a collection of items by CSV file or Databhub table [1]. They can customize the look of the item by selecting which item fields to display and in what order as well as the format of the display, using pop ups or simply listing out the values. The rating widget for an item can also be customized by color, font size, and icon image.

We have also designed and written libraries for embedding rating and recommendation functionality into any website. Users can specify the item ID in the rating widget and include it next to an item on their website. Users can also use a list widget to incorporate browsing and recommendation function on their website.

We evaluate our framework by having real world users, who are not programmers, build recommender systems with it. Overall, the users thought that the tool for building the recommender system was intuitive and easy to use. We conducted NASA Task Load Index surveys and found that using the tool was not very demanding and did not require much effort [12]. We are conducting ongoing surveys with the end users of these built recommender systems to evaluate their usability and overall user experience.

The thesis is outlined as follows:

In Chapter 2, we discuss previous work and related projects.
In Chapter 3, we describe the core interfaces and functions of Kibitz.
In Chapter 4, we describe the essential implementation level details of Kibitz.
In Chapter 5, we discuss evaluation studies for Kibitz and findings with real world users of recommender systems.
In Chapter 6, we conclude and talk about future work.
Chapter 2

Related Work

2.1 Recommender Systems

Recommender systems are central to this project so we describe them here. A recommender system is essentially a tool for filtering information by predicting a user preference for an item. Some of the most common items found in commercial recommender systems are movies, music, news, books, articles, and products. Recommender systems attempt to generate a list of items that are relevant according to certain criteria. Usually, this criteria is relevance to a given user or a similarity to certain items. There are two major types of recommender system approaches, collaborative filtering and content-based filtering.

In collaborative filtering, the recommender system looks at a user’s past behavior and compares it to other users [16][17]. The information analyzed could be explicit, such as a rating or purchase history, or implicit such as viewing an item. With this information, the system compares a given user to other users with similar behavior. If a similar user showed preference for an item, that same item might be recommended to the given user. An example of this is Amazon, which compares users by looking at purchase history, views, and ratings and then recommends products preferred by similar users. The benefit of this approach is that the system does not need to understand anything about the item explicitly. This allows collaborative filtering algorithms to be implemented across many domains without any expertise about that
domain. One downside is that a large amount of user data is usually needed for the recommender system to begin making meaningful recommendations, also known as the cold start problem.

In content-based filtering, the attributes of an item are analyzed explicitly. The recommender system attempts to characterize a given item and then recommends other items that are similar to it. In the context of user recommendations, it might recommend items that are similar to items already preferred by a user. An example of this is Pandora Radio, a music recommendation service. It analyzes a song by a set of predefined attributes and then finds other songs with similar attributes.

2.1.1 Relevant Projects

There are many research projects and open source packages that aim to make recommender systems easier to build. Examples include LensKit [2], Apache Mahout [3], LibRec [4], and MyMediaLite [5], among many others. Most of these projects attempt to make it easier for developers or programmers to build recommender systems. However, there are very few that attempt to make it easy for any kind of user to build a recommender system.

One commercial system that does attempt to bring recommender systems to a wider audience is Yusp by Gravity R&D [6]. Yusp is a SaaS recommendation engine that is aimed at businesses, both online and offline. They offer recommendation and personalization solutions in industries ranging from eCommerce, classifieds, marketplaces, content sites, dating platforms, job portals, and traditional, brick-and-mortar businesses, among others. Their solution involves embedding tracking code into their customer websites. This code then collects information about user behavior as well as item data and metadata to build a recommendation model. Their model is a hybrid of collaborative and content-based filtering.
2.1.2 Challenges in Building a Recommender System

Kibitz uses collaborative filtering for its recommendation algorithm. One of the challenges in using collaborative filtering is the cold start problem. This refers to the initial stage of building the recommender system when data about users is scarce. There is usually a large amount of user behavior data needed when compared to other approaches. The benefit of this approach is the ability to recommend a very wide range of items.

There are a number of solutions to cope with the cold start problem. One solution is using active learning and selectively choosing which data points to include in the model based on how useful that data point is to performance [9]. This is also known as rating elicitation [10][11]. Other solutions include using a hybrid of collaborative and content-based filtering. When the system is still coping with data scarcity in the initialization stages, content-based filtering can be used to assist the recommendation model as it builds the collaborative filtering component. Kibitz attempts to cope with the cold start problem by helping users retrieve their data (such as item ratings) from various sources that can be input into the recommender system.
Chapter 3

Kibitz

The Kibitz framework consists mainly of a web application for users to create recommendation systems. These recommendation systems are realized in the form of a web application as well. We will call users who want to create recommendation systems creators. The web application that is deployed will be used by end users. Creators have a collection of items that they would like to input into the recommender system for end users to browse or get recommendations on. Throughout this thesis, we will refer to the item columns (in the relational table sense) as fields, attributes, or parameters interchangeably. They are the field values that represent the items if each item were a row or record in a SQL table. This chapter describes the interfaces of this framework.

The goal of the end user interface is to allow users to browse and rate items as well as view recommended items. We provide a general standalone website that displays items provided by the creator. The user can sort and filter these items as well as do full-text search. The user can also view recommended, rated, and not interested items.

The goal of the creator interface is to allow users to upload a collection of items and customize the look and feel of their standalone website. Users can provide item data through a CSV file or a Datahub table. After creating the website, they can customize sorting and filtering of items, the rating widget, item layout, and which item fields are shown and in what order.
3.1 End Users

We now describe the interface displayed to end users. After the user logs in, he or she will be redirected to the home page. The top left element is the recommender name which the creator has customized. Next to the recommender name is the search bar. The user is able to do full text search on the items. There is fuzzy search enabled which allows the user to search for items without having to make an exact match. Below the search bar are the three main tabs: Home, Recommendations, and Your Rated Items.

The Home tab is where all items get displayed, as shown in Figure 3-1. They are shown according to the order given by the CSV file or the table order in Datahub. The user can also specify the default item order by selecting an item field to sort on. By default, the user is displayed a grid of items with individual templates customized by the creator.

![Figure 3-1: End User Home Page](image)

Each template is equipped with a button that allows the user to mark the item as "not interested." This removes the item from the display and then adds it to a list of "not interested" items that the user can edit. These items will no longer show up in any page on the recommender system except the "not interested" page. This user
can access this page by clicking through the top right icon on the home page.

In addition to a grid based view, a user can view items in a more minimal display. We have provided a text based view that contains only the item title, the rating component, as well as item fields selected by the creator. This can be seen in Figure 3-2. This allows the user to view more items in a smaller amount of space.

![Figure 3-2: Text Based Item View](image)

We have a collapsible side navigation for sorting and filtering options. This can be seen in Figure 3-3. The user can sort on a set of numerical fields that the creator has chosen. The user can also filter items with a range slider for numerical fields and using checkboxes for categorical fields.

Lastly we have the Recommendations and My Ratings tabs. The recommendations page displays recommended items, which are computed and fetched from the Kibitz server. Items that are rated by the user will be moved from the recommended page automatically to the ratings page. The ratings page displays a user’s rated items which belong to the current recommender system.
Figure 3-3: Sort/Filter Side Nav
3.2 Creators

3.2.1 Main Page

We now describe the interface used by creators. The user is first greeted by a login/signup page where he or she creates a user account with username, email, and password. This can be seen in Figure 3-4. We have given creators the option to use their Datahub accounts to sign up or sign in to Kibitz. We have chosen this design because creators will automatically be given a Datahub account on sign up. This is for the purpose of storing and managing item data provided by the creator. We foresee that creators will eventually use Datahub directly to manage their data. Given that the core function of Kibitz relies on user-generated data collections, having such a tool for account management is a natural feature.

![Figure 3-4: Creator Login Page](image)

Next we describe the home page of the creator web application. After the user logs in, he or she is displayed a list of recommendation systems created by the user.
in summary format. This can be seen in Figure 3-5. If no recommender has been created yet, the user is taken directly to the creation process. If there is only one recommender, the user is taken directly to the settings and customization page. The user is shown the name of the recommender system created, the active or paused status of the system, as well as the URL of the deployed website. In the left side bar, there is a button to a profile page. Currently, this page displays only username and email. We anticipate for future designs that this profile may contain more information about the user and aggregate metrics of all recommender systems created by the user.

Figure 3-5: Creator Home Page

3.2.2 Recommender Creation

To create a recommender system, the user clicks on the create button below the list of recommender systems. This leads to a modal dialog as shown in Figure 3-6. For inputting data into the recommender system, the user has the option to choose between uploading a CSV file of item data, as seen in Figure 3-7, or using data from a Datahub repository, as seen in Figure 3-8. In either case, the user must provide a recommender name which is displayed on the deployed website. Currently, the Kibitz generated websites are hosted on a default URL (http://kibitz.csail.mit.edu/) with a URL ending that is provided by the user. If the user specified a URL name of "books," the corresponding website URL would be "http://kibitz.csail.mit.edu/books."
If the user chooses to upload a CSV file, they can select the CSV headers which correspond to title, description, image link, and a universal identifier such as an ISBN (for books) or UPC (for products). These are basic parameters by which most items can be described. We have attempted chose the smallest set of parameters possible. Image link is necessary to display an item in a grid style layout. If it is not provided, items will default to a text based layout. A universal identifier such as an ISBN or UPC is useful in identifying an item across multiple recommender systems or retrieving information about the item. Description may or may not be included. Primary key field can also be specified; otherwise, they are generated automatically.

Figure 3-6: Recommender Creation Modal

If the user chooses to create a recommender system through the Datahub platform,
he or she must choose that repository and table in which the items are stored. The repositories are tables are fetched from the user’s Datahub account and belong to either the user or have the user as a collaborator (as set in Datahub). The primary key field can be selected or added to the table.

After selecting the parameters, the user clicks on the create button in the modal and the recommender system is automatically setup and deployed. The user can visit the website from the link provided in the list summary page.

Figure 3-7: Creation Modal, CSV File Upload
Figure 3-8: Creation Modal, Datahub
3.2.3 Main Settings and Customization

We now describe the settings and customization page. The user clicks on a recommender system in the list page and is redirected to the page as shown in Figure 3-9. This is the general tab, which is the default tab. The user is able to edit the recommender name on this page. This is the name displayed on the top left corner of the deployed website. It is similar to a custom logo.

The user is also able to choose which fields can be used to sort and filter items on their website. A book seller for example may want to have their end users filter books by author. We provide two item types for filtering, numerical and categorical. Numerical type is for item fields that are integers or decimals. Examples would be publication year of a book or number of players for a game or a price value. Categorical type is a generic type for fields that take on string values. Examples would be book genre or game categories like RPG.

![Settings and Customization Page. General Tab](image)

For numerical fields, the server automatically finds the minimum and maximum of all values for the selected item field and then displays it to the end user for filtering. A range slider given to the end user and only items in that range will be included.
For categorical fields, the server finds the most frequent values for a field and displays those to the end user. The values can be ordered by alphanumerical order or by frequency. For example, a store’s books might all fall in the categories of adult, children, or young adult. This would allow an end user who is only interested in children’s books to filter on one of those categories.

The second tab in the settings page is the templating tab as seen in Figure 3-10. Users are able to customize the way an individual item is displayed to the end user. There are two default templates to choose from. The first is a template design which utilizes popups. There is a popup for item details, which has the item fields, as shown in Figure 3-11. The second popup is a ratings popup which shows the distribution of ratings for the item, as shown in Figure 3-12. The second design as seen in Figure 3-13 has all the item fields listed out. This offers a design with less complex visual logic as is suitable for users who want a more minimal display.

![Figure 3-10: Templating Tab Page](image-url)
Figure 3-11: Item Template Popup

Figure 3-12: Rating Template Popup
Figure 3-13: Template Design, List Display
In addition to customizing the item template, we wish to give creators the ability to customize the rating widget. This gives them the freedom to determine the rating icon font size, color, and image without writing any HTML or CSS.

In the right column of Figure 3-10, the user is given a form of GUI widgets to use to style their item template. The user is able to customize the look of the rating widget in the template. The icon color on hover over and hover out can be customized. Icon size can be selected as well. In addition, the user is able to choose what kind of icon is displayed, where the default is filled and empty stars. This can be seen in Figure 3-14. The user can choose to edit individual icons or edit the icons collectively all at once. Selected and unselected icon can be customized as well.

![Editing Rating Icons]

Figure 3-14: Rating Icon Selection
At the bottom of the templating page, the user is able to edit which item fields are displayed to the end user on the website. We utilize a drag and drop interface that has a click highlighting function. This allows the user to specify which item fields are included as well as the order of the display, both of which are important. The user is able to save all the changes by clicking on the save button at the bottom of the page.

Figure 3-15: Item Field Selection GUI
3.2.4 Miscellaneous Settings

In the data tab, the user is able to re-upload a CSV file of items to their website as seen in Figure 3-16. Similar to the flow for creating a recommender system with CSV file upload, the user selects a CSV file, chooses which file headers correspond to the essential item headers: title, description, image link, and universal identifier (e.g., ISBN). The user then uploads by clicking on the upload button. This will repopulate the items in the recommender system. This function is useful, for example, for store owners who periodically update their inventories. We add some consistency checks such as making sure file headers have not changed. Otherwise, a popup will ask the user to confirm the collection update.

![Figure 3-16: CSV Upload Tab](image)

The next tab is the widget tab, where the user can copy and paste HTML code that can be used to embed kibitz functionality into their website. Currently there is the rating widget which users with some programming experience can insert next to an item on their own website. The requirement would be to be able to specify and item id that is recognizable to the Kibitz server and database as well as having the user logged in with Kibitz. There is also a list widget which displays items and item recommendations to end users. The only requirement is for the end user to be logged in with Kibitz.

Finally, there is the advanced tab. Users can reselect which repository and tables
in Datahub that the recommender system pulls from as well as which item fields correspond to the essential item fields: title, description, image link, and universal identifier. There is also a CSS edit form that allows users with programming experience to write their own CSS to style their website.
3.3 Widget Libraries

We have created a suite of libraries for embedding Kibitz functionalities into arbitrary websites. There are three libraries: rating, list, and login.

We have a rating library that allows users to embed rating capability with recommendations into their rating widgets. The rating library adds event listeners to the user’s rating widget. These listeners, when triggered by a rating action, will make an API call to the chosen backend and create a rating. We also allow more advanced developers to write their own event listeners and API calls by triggering events on a load or rate.

The library can be used with the Kibitz recommender backend or an arbitrary backend. When using Kibitz for the recommender backend, the creator specifies an item ID field in the HTML of the rating widget. This item ID identifies the rating in the Kibitz system and is used for the purposes of recommendation. The recommender system ID must be specified as well.

If a non-Kibitz backend is used, get and post API’s must be specified for getting a rating value and making a rating. Using data attributes in the widget HTML, the user must also specify necessary headers for making the request (such as CSRF token) and either query parameters (for GET requests) or a POST payload for the purposes of identifying the item in the backend. The library will count the number of GET requests in a page and batch them into a single request to set the initial rating values.

We also have a list library, which is used to display ratable items as well as getting recommended items. This library is designed to offer the widget form of the standalone Kibitz website. Therefore, it only uses the Kibitz backend. The only requirement is to specify the recommender system ID. The user can specify a rating template as well as an item template. The library provides a default template in a raw, unstyled form for both the rating component as well as the item display. The default rating template is a `<select>` tag with `<option>` as rating values. The default item template is list of `<div>` tags for item fields. These default templates are designed to be styled by the user.
For both rating and list libraries, the user must be authenticated, either with Kibitz or another backend. We provide this function with a login button. The creator can embed this into their website. When the end user clicks on it, they are shown a popup that allows them to login into the Kibitz website. They will be able to use their Kibitz credentials in the third party website owned by the creator.
3.4 Kibitz Version One

We now describe the first version of Kibitz which preceded the one discussed above. The first version of Kibitz was created in 2015. It introduced the first set of core features. Some of those features include being able to deploy a stand alone website automatically without programming experience, allowing the user to customize the style of the website, and rating and recommendation. However, there were some fundamental design changes we made in the second version to address some usability issues.

Firstly, there were many core functionalities that would occur on the landing page before the user had reached the home page. This included creating the recommender system as well as creating a repository. Additionally, a single functionality could be accessible in many parts of the interface, particularly, both the landing page and the home page. For example, the original creation process could be carried out on the landing page as well as after the user logged in and arrived on the home page. This can be seen in Figure 3-17 and Figure 3-18. We decided to reduce the entire creation process to a single modal accessible in one location after the user signs in. This simplifies the interface and is a more natural organization of functionalities.

There were many screen overlays that combined different user flows. Creation of a repository and creation of a recommender system occurred in the same process, as seen in Figure 3-19. We found that the simultaneity was confusing and even overwhelming for some users. The use of specialized database terms such as "repository" was also unnatural for some users with no programming experience. In the new version, we remove manual repository creation and the use of the terminology "repository."
Figure 3-17: Creation Flow on Landing Page

DATAHUB ACCOUNT

To use Kibitz, please sign-in to your Datahub Account and allow Kibitz access to your repository containing your list of items. If you do not have a Datahub Account, you may create one via the Datahub registration page.

Repository Name

Allow Kibitz Access
Figure 3-18: Creation Flow on Home Page

Figure 3-19: Landing Page with Overlays.
Another feature change we made was editing the deployed standalone website with GUI widgets. Originally, the only way to change style was by raw CSS edit in the form of a textarea. This was useful for programmers with CSS knowledge but was not usable for non-programmers.

We also made some design changes to the default style for the item display. The original item template design utilized for the most part a horizontal display across the page with items being listed top down, as seen in Figure 3-20. We decided to introduce instead a 2D grid design to display items, which is more space efficient.

Figure 3-20: Original Kibitz Item Display
Chapter 4

Implementation and Architecture

Here we discuss Datahub, the platform used to host and manage user data. We then describe the major implementation details of the Kibitz server, the web applications, as well as the recommendation engine.

4.1 Datahub

This project utilized Datahub, a data management tool and platform [1]. It is managed by MIT CSAIL’s Living Lab. Datahub gives users control over their own data and allows them to manage it by providing a set of tools for uploading, querying, and visualizing datasets. We used Datahub because we foresee users in the future wanting to be able to manage and have ownership over their data outside of the Kibitz application. Datahub is a natural choice for this purpose.

Datahub also serves as a data hosting and storage service. Given that each recommender system can have millions of item records, it would be difficult to manage all user data on the application server. Datahub helps offload some of this data storage.

Datahub also provides a REST API that applications can use to manage user data. During the course of this project, we helped add several new API endpoints that were necessary for the Kibitz application. We added API endpoints that allow users to upload and read files as well as endpoints to create tables from files and export files from a given table.
4.2 Kibitz Server

4.2.1 Technologies Used

The Kibitz backend application is implemented in Python and Django. The server side database uses PostgreSQL. The web server used is the Apache 2 web server. The application is hosted on an OpenStack machine running Ubuntu Linux [13]. Apache Solr is used for fuzzy, full-text search [14]. It also allows the indexing of millions of items and facilitates rapid item search. For some creators with large collections of items, in the case of store owners, this was vital. For the recommender engine, Apache Spark is used to do collaborative filtering using the Alternating Least Squares (ALS) method.

4.2.2 Application Architecture

There are three main components in the Kibitz system: the server, the front end applications, and Datahub. Nearly all computation occurs on the server side. Datahub is used as a data hosting tool to store creator item data. The Datahub API is used to upload CSV files of items and transform those files into Datahub relational tables. The Datahub API is also used to make SQL queries to fetch item data from the tables. The rows returned may be sorted and filtered through constructed queries sent to Datahub.

We now discuss the issue of user data storage, in particular, user rating data. There are three natural choices for storing rating data. One design is to have the data stored in the creator Datahub account, giving ownership of the data to the creators. A second design is to have the data stored in the end user accounts. End users would then need to have their own Datahub accounts. This would give end users full control of their own rating data, which would be in line with the goals of the Datahub project. A third design would be to have all user rating data stored centrally on the Kibitz server.

The first design, where ratings are stored in the creator account, is advantageous
because the creator has more control over the data in their recommender. This also takes control away from the Kibitz system which may in turn build trust with the creators. However, creator access control is still an issue. We still may not want creators to be able to see end user ratings or have complete control over it. Another disadvantage is that to build a recommendation model with items across recommender systems, an HTTP request would have to be made to every creator account. Given the potential number of creators in the Kibitz system, this could significantly hinder performance.

The benefit of the second design, where ratings are stored in end user accounts, is that end users would have control over their own data and would not need to expose it to any third party until the recommendation model is built. The downside is that Kibitz would need to make an HTTP request to every single user who has rated an item associated with the given recommender. Given the potential number of users across all Kibitz recommender systems, this could significantly affect performance.

We choose the third design, where all user records, rating, and "not interested" data are stored on the Kibitz database as opposed to Datahub databases. We chose to do this to minimize the amount of calls needed to aggregate data across recommender systems. If rating data were stored across multiple Datahub accounts with their respective owners, the Kibitz recommender server would need to make calls to each database in order to make a recommendation using collaborative filtering. This could be hundreds of API calls to build the recommendation model. Given that item rating events occur frequently and that the model needs to be updated after such events, we decided that centralizing rating data was the more tenable design. We also centralize the user records. We chose this design so that we could identify users across different recommender systems. This allows ratings of an item in one recommender to carry over to an identical item in another recommender, eliminating the need for a user to make the same item rating twice.
4.3 Front End Applications

Both the creator website and the end user website are implemented in Javascript and AngularJS. AngularJS is a large framework that allows rapid creation of single page applications [15]. It has a vibrant ecosystem of developers that create useful libraries that other AngularJS developers can use to quickly incorporate common features into their web application. This was the optimal choice for Kibitz as we were prototyping many new designs and needed shorter development cycles.
4.4 Recommendation Engines

In the first version of Kibitz, the recommender engine was based off of the Apache Mahout project’s collaborative filtering algorithm. The algorithm computes user-to-user similarity, or the neighborhood approach, utilizing the Pearson correlation coefficient. We decided to use Apache Spark with the alternating least squares (ALS) method [18]. Our users, bookstore owners in this case, had the problem of data sparseness, where their item collections were far greater in size than the number of users they had. ALS deals with data sparseness much more efficiently than the standard Pearson correlation coefficient method. The new algorithm was therefore a better choice. In the future, may add the option to use different recommendation engines depending on the data provided by the user.
Chapter 5

User Studies

5.1 Bookstore Owners

In this study, we make a first attempt to collaborate with real world users in building recommender systems with Kibitz. We conducted user studies with three local bookstore owners and had them create their own recommender systems with data from their own store. Each store had at least 100,000 titles. We chose the subjects because they were users with a real need for recommendation systems. Another reason was the sizable volume of items and number of users.

5.1.1 Interviews

We begin the study by asking the bookstore owners a series of questions intended to better understand their needs and what they want from a recommender system. We also try to understand their past experience with recommender systems.

We first ask about their prior programming experience as well as how long they believe it would take to build a recommender using Kibitz. Most bookstore owners had little to no programming experience, including HTML and CSS. One store said none of his staff had any programming experience at all. One said that he personally had some HTML experience from decades ago but was outdated already. One staff member of a store said he had some HTML and CSS experience but not much more.
Overall, it seemed like the bookstores did not have the resources or expertise to create a recommender system on their own. Most bookstore owners said they thought it would not take much time to build and customize their recommender system using Kibitz. However, the time to gather enough user rating data to have the system producing quality recommendations was a concern for them.

We then asked them why they wanted a custom recommender system. Most bookstore owners felt that recommendations from large recommender systems were not as relevant to their users. They said that their own staff recommendations were more specific and personal for their customers. Oftentimes their own recommendations were of better quality as well. They felt that staff had more expertise about certain genres or were better able to understand a customer to make a personalized recommendation. They also felt that customers were more trusting of human recommendations. However, they face two problems with this approach. One is scale. It isn’t feasible for staff to make recommendations for every single customer they have. Store owners say staff often cannot get to every customer. If they had a recommendation engine that could assist them, they might be able to get to more customers. The other issue is lack of knowledge about a certain topic. Bookstore staff usually have expertise in a few genres or categories, outside of which they have trouble giving recommendations. A user may need a recommendation that the staff does not have expertise in. A recommendation engine could be of use in this case.

We also asked bookstore owners how they would go about getting a recommender system developed. Most said they had few options and would probably try to find a professional developer. One store owner said they had tried a content-based filtering eCommerce solution before but the results were poor and they discontinued using it.

We talked to the owners about embedding a recommender system into their existing website. Most were open to the idea and some thought it would be important for their business. They said that they wanted as much traffic going to their website as possible and thought an embedded widget could help. Some even hoped that the standalone website could link back to their own website. However, one owner was concerned about the need for an end user to be logged into the Kibitz system for
recommendations. A single login system seems to be an important feature for some owners.

5.1.2 User sketches

After the interview, we asked the bookstore owners to make sketches of how they would like their recommender system to look and function.

The first owner’s design emphasized staff item recommendations as well as user questions. This owner’s original website included a form of questions that customers could fill out. They would then get hand-picked staff recommendations. The owner was therefore interested in combining human recommendations and algorithmic recommendations. The owner emphasized the desire to have a recommender engine to assist rather than replace staff recommendations. The design includes a section where the user can interact with the recommender system by chat or dialogue. The owner suggested that this "chat bot" could be based off of staff preferences or knowledge. In turn, it could be a kind of pseudo-staff member that helps make recommendations for customers. After talking to the customer, the recommender would then understand the user’s preferences and display suggestions as show in *Figure 5-1*.  

![Design 1](image.png)  

*Figure 5-1: Design 1*
The second owner’s design emphasized an item attribute taxonomy, or an item tagging system. This taxonomy would have a collection of attributes, or "tags," that describes or characterizes each item. These tags would be generated by customers and staff. The website design would be as follows. The user would begin with a single product along with a list of tags displayed. A list of products most similar to the initial one would be displayed to the right. The connections between the similar products would be mapped out by the tagging system. Each similar product would be linked to the initial one by the tags that they have in common. This can be seen in Figure 5-2. This owner was interested in exploring different visual tools for finding items relevant to a customer.
Figure 5-2: Design 2. Interactive Item Similarity Map
The third owner’s design emphasized embedded rating and list widgets as well as a similar items widget. The owner had a design where items were displayed individually on an entire page. It would show the item attributes and also provide a rating function. Below the item would be similar items that a user might find relevant.

Figure 5-3: Design 3
5.1.3 Tasks

After the user sketches, we asked the bookstore owners to create their owner recommender systems. Each had a CSV file of their book data. Some bookstore owners had more trouble getting book data into a usable format. We assisted them in preparing their CSV file before the start of the user study. For example, one bookstore owner was unable to get book titles into human readable format. To cope with this, we used ISBN codes to fetch book information, including book title.

Most bookstore owners used a standard laptop computer. One used a desktop computer in their office. The time to create a recommender system took on average of 6 minutes with the longest time being less than 10 minutes.

After creating the recommender system with a CSV file, bookstore owners were asked to edit the visual style of their recommender as they saw fit. They edited the visual style of the individual item templates as well as the rating widget. They also edited the sorting and filtering options for the end user. They then activated the recommender and visited the created website. They were also asked to update their item collection by reuploading a CSV file.

5.1.4 Post-Task Interviews

After the owners created their recommender systems, they each completed a survey which included measuring workload according to the NASA Task Load Index [12]. All the owners found using the tool to be easy and required little to no effort or assistance. The tasks required little to no demand or effort. The owners were not frustrated with the tasks and were overall satisfied with their performance.

We then interview them about their experience using Kibitz to build a recommender system. Overall, there were no major design changes suggested for the website for building a recommender system. One owner suggested allowing creators to change the background color of the standalone website. However, overall they thought Kibitz was easy to use and would be able to make another recommender system on their own.
5.2 End User Studies

We are currently conducting ongoing surveys and interviews with bookstore staff as well as customers. The purpose is to get feedback on the user experience of the Kibitz created websites. At the time of writing, we have feedback from some staff members of the bookstores. Overall, staff members were able to use the website without much difficulty. They were able to browse, search, sort, and filter items without major problems. One staff member suggested having purchase links. Another reemphasized having a widget embedded in the original store website.

We have also conducted interviews with some customers. During the interviews, we walked customers through the process of using the created website. We instructed them to perform the core functions of browsing and rating items as well as searching and filtering. Overall, there were no major problems in performing the core functions. They found the interface easy to use and required little effort.

One customer noticed that filter parameters were all or nothing. As in each filter condition needed to be met for an item to show up in the filter results. Essentially, we are doing a logical AND operation over the filter parameters. What might be interesting is doing a logical OR operation over filter parameters. In other words, if an item meets any filter parameter condition, it would show up in the results. For example, if a bookstore customer wanted to see books on artificial intelligence as well as 18th century English literature, he could select both categories in the filter settings and see both types of books in the results.

One source of confusion for some customers was the meaning of the rating widget. One customer wondered whether the rating displayed in the home page was their own rating or the general rating of the item. Another customer questioned what the rating icons represented. Possibilities considered by the customer included book quality and the degree to which a book was recommended, among others. Some customers were also concerned about a rating widget with all empty rating icons (all empty stars). They thought that some people might confuse this to mean a rating of below one, the worst possible rating. Some also suggested having a "not rated yet" to mitigate this
confusion. Overall, some users felt that the semantics of the rating widget could be more clear. We attempt to improve this by adding more descriptive help text next to the widget as suggested above.

Some customers were concerned that the amount of information about an item might be insufficient. One customer instinctively clicked on a item, expecting it to go to an item page with more details. Another customer thought that items needed a description. We did not explore the process of retrieving item data in a systematic way. This was left to the creators. However, we created a program for one bookstore that retrieved book information from ISBN codes. In the future, we may try to offer this functionality to other bookstores or expand to UPC codes for products.

One feature suggested was wish lists or saved items. One customer said that he didn’t necessarily know enough about a book to rate it but would like to rate it in the future. A saved list could help users collect items that they interested in but not able to rate yet. Another feature suggested was having an interface where a new user specifies preferred item categories. This would help build an initial user profile that would assist the recommender model, similar to how Netflix asks for movie preferences for new users. Another suggestion was to somehow better utilize "not interested" information that users provide. One customer said that this function would not be very useful if it didn’t hide related items. In addition to hiding related items, this information might also be used to tune the recommender model.
Chapter 6

Future Work and Conclusion

6.1 Item Categories to Explore

Although we evaluated Kibitz with books and bookstore owners, we believe that there is a wide range of item categories that could be explored in recommender systems. Many of these categories do not necessarily have a commercial focus. One example is recommending university courses. There are many tools for evaluating courses at a university but few that recommend courses to its students. Another potential area to explore is academic grade prediction. With a database of classes and grades, where a letter grade is defined to be a rating, we could use collaborative filtering to predict a student’s grade in a class.

One area we are currently exploring is restaurant dishes. There are many solutions for recommending restaurants but little to no recommenders for individual restaurant dishes.

Public libraries are also of interest. We envision placing a kiosk running a Kibitz recommender system at a public library where users could rate and receive recommendations for books. The customer traffic in private bookstores is much lower in comparison to libraries. The inventory of books in a public library is also much larger than that of a private bookstore.
6.2 Conclusion

We have developed Kibitz, a framework for creating and customizing recommender systems. This framework requires no programming experience and works for any collection of items. The created recommender system is a standalone website where end users can browse and rate items as well as search, sort, and filter those items. We provide templates for organizing the items, either in a grid format with images or a vertical text based format. Creators can customize these standalone websites by selecting individual item templates, choosing what item attributes to display, and styling an item’s rating widget. We also created a suite of libraries that allows any creator to embed rating and recommendation functionalities into an existing website. We conducted user studies with bookstores to understand what real world users need in a recommender system and receive feedback about the recommender system builder. We also conducted surveys and interviews with end users to get feedback on the user experience of websites created with Kibitz.
Bibliography


1-35

Appendix A

User Interview Questions, Tasks, and Surveys

A.1 Creator Interview

Pre-task questions:

- How long (do you predict) would it take to have a recommender system up and running?

- How much web programming experience do you have? How familiar are you with HTML or CSS?

- Why would you want a custom recommender system (over a generic services like Amazon, Goodreads, etc.)? How would it be useful? How would you use it?

- What problems do you foresee with a custom recommender system? If you had to build a recommender system, how would you go about doing it or getting it made?

- Where and how is your data stored? How does it get updated? How would you integrate your data with the recommender system?
• Do you want a standalone site or something that integrates into your current site?

• How would you imagine the integration? Would you want a rating widget next to items?

• How about a widget with a list of items? What would that list look like?

• What kinds of sorting/filtering displays would you want? (Side nav, horizontal gallery, like google images, modal?)

• If you were building a recommender site or embedded widget, what functionalities or features would you have? Which are necessary, important, or just "nice to have."

**Pre-Interview Tasks:**

• Make a sketch of how you would want your standalone site or list widget to look.

**Pre-Task:**

• Prepare CSV file of item data

**Main Tasks:**

• Create a recommender with your csv data set

• Edit the visual style of the individual item template

• Select which item attributes to display to end users

• Edit item filtering/sorting options

• Activate recommender. Visit the page.

• Update your item collection (using CSV)
Post task questions:

- What do you like about this tool?
- What would you change?

Usability:

- How hard do you think building a recommender system was?
- How hard do you think building another recommender using this interface would be?
- Is this app easy to use overall? How usable would you say it is?
- Were you able to create what you want? What did you want to create that you were unable to using this app?
- Does this app allow you to accomplish what you want? What were you not able to accomplish?
- What redesigns or extra features would you like to have?
- What parts of the interface were difficult to understand or use? When and why?

Utility:

- Do you find an app like this valuable, with the current design?
- In what ways?
- How valuable overall do you find this app?
- What changes or new features would make it even more useful?
- What alternatives to this tool would you consider?
A.2 Creator Post-Task Survey

Post-Task Survey

Questions after building recommender

* Required

What's your name or organization? *

Your answer

Task Difficulty

How difficult was the task?

How hard do you think building the provided recommender with Kibitz is?

- 1 - Easy - I could build this on my own without much effort
- 2 - Moderate - I could build this on my own, but I'd have to search for lots of help
- 3 - Challenging - I'm not sure if I could build it on my own, but there's a good chance I could
- 4 - Hard - I could build this on my own, but would need to ask someone questions along the way
- 5 - Very hard - I don't think I could build this unless someone walked me through it step by step
Workload Assessment

The following assessment is used to measure your personal opinion on how much workload was required of you during the task you just completed.

**Mental Demand**
How mentally demanding was the task?

1  2  3  4  5  6  7

Very Low 〇 〇 〇 〇 〇 〇 〇 〇

Very High

**Physical Demand**
How physically demanding was the task?

1  2  3  4  5  6  7

Very Low 〇 〇 〇 〇 〇 〇 〇 〇

Very High

**Temporal Demand**
How hurried or rushed was the pace of the task?

1  2  3  4  5  6  7

Very Low 〇 〇 〇 〇 〇 〇 〇 〇

Very High

**Performance**
How successful were you in accomplishing what you were asked to do?

1  2  3  4  5  6  7

Very Low 〇 〇 〇 〇 〇 〇 〇 〇

Very High
Effort
How hard did you have to work to accomplish your level of performance?

1 2 3 4 5 6 7
Very Low       Very High

Frustration
How insecure, discouraged, irritated, stressed, and annoyed were you?

1 2 3 4 5 6 7
Very Low       Very High

Submit
A.3 End User Survey

End User Post-Task Questionnaire

Please input your username or email that you used to log in.

Your answer

Age

Your answer

Gender

☐ Female

☐ Male

☐ Other

☐ None

What is the name of the recommender website you are testing?

Your answer
How often do you use recommendations on Amazon, Netflix, Spotify, etc.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Very familiar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not at all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Answer the following questions about the recommender website

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the recommender to find what I like is easy.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It required too much effort to tell the system what I like.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found it easy to tell the system about my preferences.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I will tell my friends about this recommender.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The recommender system helps me discover new, interesting things.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The recommender's interface provides sufficient information.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Are there things you would change about the way the site works?

Your answer

Is there anything in the site that seems unnecessary?

Your answer

Are there any missing features that you think would make the site more useful to you?

Your answer

Do you have concerns, suggestions, or other feedback about the website interface and/or design?

Your answer
Could you see yourself using a recommender like this to get recommendations for other things (movies, music, restaurants, hotels, performances, recipes, etc)?

1 2 3 4 5
Not at all 〇 〇 〇 〇 〇 Absolutely

What other things would you like to get recommendations for?

Your answer

How could something like this be useful for you?

Your answer

SUBMIT