AppGrouper: Knowledge-graph-based Interactive Clustering Tool for Mobile App Search Results

Shuo Chang^{*1}, Peng Dai², Lichan Hong², Cheng Sheng², Tianjiao Zhang², Ed H. Chi²

¹GroupLens Research
University of Minnesota
schang@cs.umn.edu

²Google Research
Google Inc.
daipeng, lichan, csheng, cathyzhang, edchi@google.com

ABSTRACT

A relatively new feature in Google Play Store presents mobile app search results grouped by topic, helping users to quickly navigate and explore. The underlying Search Results Clustering (SRC) system faces several challenges, including grouping search results in topical coherent clusters as well as finding the appropriate level of granularity for clustering. We present *AppGrouper*, an alternative approach to algorithmic-only solutions, incorporating human input in a knowledge-graph-based clustering process. AppGrouper provides an interactive interface that lets domain experts steer the clustering process in early, mid, and late stages. We deployed and evaluated AppGrouper with internal experts. We found that AppGrouper improved quality of algorithm-generated app clusters on 56 out of 82 search queries. We also found that the internal experts made more changes in early and mid stages for lower-quality algorithmic results, focusing more on narrow queries. Our result suggests, in some contexts, machine learning systems can greatly benefit from steering from human experts, creating a symbiotic working relationship.

Author Keywords

Interactive machine learning; Clustering; Search Result Clustering

ACM Classification Keywords

H.1.2. User/Machine Systems

INTRODUCTION

Presenting search results grouped by topic (Figure 1b) is a popular alternative to displaying results in plain ranked lists (Figure 1a), enabling users to explore and navigate topic space of search results. This technique is now used in many well-known websites, such as Google Play Store,

^{*}Work was done during internship at Google.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IUI 2016, March 7–10, 2016, Sonoma, CA, USA.
Copyright © 2016 ACM 978-1-4503-4137-0/16/03 ...\$15.00.
<http://dx.doi.org/10.1145/2856767.2856783>

Figure 1. Example of search results for the query “games” in Google Play Store: (a) Left side shows search results in grid; (b) right side shows search results grouped by topic.

Pinterest, Google News, etc., and has attracted attention from the research community, where the technique is referred to as Search Results Clustering (SRC for short).

Despite past efforts [6, 12, 17], purely algorithmic SRC techniques do not generally create production-quality document clusters due to the complexity of the unsupervised clustering problem. Search results change all the time due to new documents, changes in user interests, and discovery of new ranking signals. Therefore, the dynamic nature of search results requires an efficient on-the-fly clustering process. Furthermore, many properties of the clusters affect the final user experience, for example, whether there are descriptive labels for clusters; whether documents in clusters are semantically coherent; whether clusters cover all relevant topics under a search query.

Figure 2. Interface of AppGrouper — an interactive app clustering tool for Google Play Store — consists of three parts: A) side bar with a list of topic labels used in clustering, B) control panels for loading and running clustering, and C) editable app clusters.

We propose an interactive machine learning approach that combines human intelligence with machine intelligence for app search clustering in Google Play Store. Inspired by prior work [17], we design a knowledge-graph-based clustering process for a query and its associated ranked apps: extract topic labels from apps, cluster topic labels, and assign apps to topic label clusters. This process not only provides topic labels that describe the app clusters, but also opens up opportunities for expert input. For example, once major topic clusters are associated with a query, experts can create or edit the name for a cluster based on its content. For instance, even though the search results for “games” change over time, its topical clusters such as “action games”, “arcade games”, etc., remain relatively stable, and we can create cluster names for these clusters.

We hypothesize that expert input into clustering process can substantially improve app clusters from Search Results Clustering algorithms. Apart from incremental improvements such as picking appropriate names for clusters, experts can provide useful feedback in the process of generating topic clusters.

We built AppGrouper, an interactive clustering system (Figure 2) that generates app clusters while incorporating expert decisions in the process. AppGrouper provides an interface that depicts algorithm-generated, topic-label clusters and accepts expert input in multiple stages of clustering process: (1) *Early stage*: refining input to clustering algorithm; (2) *Mid stage*: steering the algorithm to generate more or less fine-grained clusters;

and (3) *Late stage*: editing topic label clusters and topic labels.

This paper offers the following contributions: (1) We presented a novel interactive machine learning approach, including a knowledge-graph-based clustering process and a visual interface, to solve SRC problems. (2) We designed and implemented AppGrouper, an end-to-end system that injects human judgements into the clustering process during its various stages. (3) Through evaluation with experts, we showed significant quality improvements of the clustering results generated through AppGrouper, and offer some implications for designing interactive machine learning systems in general.

This paper is structured as follows. We first survey related work and situate our research. Next, we introduce the problem of search results clustering in app search. Then, we describe the interactive clustering system, elaborating on the knowledge-graph-based clustering algorithm and how we incorporate expert decisions. We follow this by presenting an evaluation of AppGrouper. Finally, we conclude with a discussion of the findings and design implications.

RELATED WORK

AppGrouper is an interactive clustering system that enables human intervention in Search Results Clustering process. Our work is broadly related to the general domain of interactive machine learning. For the closely related area of Search Results Clustering, most research attention has gone into algorithm solutions.

Interactive Machine Learning

Interactive machine learning combines human judgments with machine learning. Prior work has experimented with interactive machine learning in various applications, including image segmentation [11], document clustering [5, 10], image search [13] and specialized tasks such as grouping social network contacts [1].

Expert input to machine learning system comes in different forms depending on the underlying machine learning algorithms and application domains. Regardless of the types of input, machine learning systems need to be transparent in depicting its results to enable efficient human input [14]. For image segmentation, Fails et al. [11] asked users to provide training samples to machine learning algorithm. Vig et al. [18] enabled end users to critique the results of recommender algorithm and tuned hyper-parameters of the underlying algorithm accordingly.

A subtle and important difference in our work is that we ask for expert input on an internal intermediate model of the clustering process. In previous works [3, 4], users gave feedback of documents that should or should not be together (must-link and cannot-link constraints), guiding algorithm to generate clusters that satisfy these constraints. However, we must design a system that clusters a changing set of apps, because the search results are updated all the time. To be precise, if expert-provided constraints are simply on what apps do or do not belong to a cluster, these constraints may become irrelevant over time, because search results are dynamic. Instead, AppGrouper asks for expert input on an internal model — topic clusters, which specify what topics belong together. Dynamic search results are then assigned to these topic clusters during serving.

Search Results Clustering

Much research attention has been drawn into improving algorithms for clustering search results, as surveyed by Carpineto et al. [6]. Early works focus on approaches that derive features from texts of search result documents and cluster documents based on these features on the fly. One early clustering system called “Scatter/Gather” [16] interactively clusters text document using a bag-of-words model. Later, Ferragina et al. [12] proposed clustering on extracted phrases from search results. Osinski et al. [15] projected text documents into a latent topic space and then performed clustering. More recently, Carpineto et al. [7] presented a meta clustering system that combines outputs from several clustering systems and showed improvements.

Apart from topically-coherent search result clusters, descriptive labels for clusters are critical for good user experience [6]. Practitioners have realized this and spent considerable effort into description-centric clustering algorithms, as evident by commercial search result clustering systems such as *Lingo3G* and *CarrotSearch*. Scaiella et al. [17] contributed to description-centric approach

by presenting a topical clustering system. The system annotates text documents with topic entities from Wikipedia and clusters these topic entities together with documents. AppGrouper uses similar knowledge-graph-based approach to cluster apps. However, algorithm-generated clustering results are not perfect, resulting in sub-optimal user experiences when browsing these clusters. We are interested in augmenting the algorithmic solutions and build an interface that enables human interactions with algorithm to fix these problems.

BACKGROUND: APP SEARCH RESULTS CLUSTERING

Why Clustering Search Results

Grouping search results by topic helps users to better discover new areas for exploration. Generally speaking, for navigational search queries, presenting search results in lists seems reasonably effective, which directs users to the particular apps they want to install, e.g., “angry bird”. The same approach offers sub-optimal user experience for broad categorical search queries, e.g., “games”. Clustering search results for categorical search queries assists users to better achieve the following goals:

- *Explore the topic space related to search queries.* For example, users are interested in discovering what games are available when they search for “games”. Users can easily see all genres of games through app clusters shown in figure 1b.
- *Navigate to interesting subtopics.* If users are interested in certain subtopic, they can quickly navigate to the groups of search results related to that subtopic. For example, if a user is interested in “casual game” apps, she can easily navigate to these apps as shown in figure 1b.
- *Discover less popular results.* Search engines typically rank less popular (e.g., less clicked) results toward the bottom of result lists, attracting less attention from users. Grouping results by topics makes less popular results from a diverse set of topics more visible to users: assuming the first 4-5 listed results for query “games” are all action game apps, clustered search results might make arcade game apps and casual game apps more prominent than organic ranked results.

Challenges of Clustering

Search Results Clustering, unlike the popular usage of clustering as a data exploration method, generates clusters that are shown to end users of search engine. This user-facing application creates the following requirements for clustering:

- **P1: Semantically meaningful topic labels for clusters.** Descriptive labels are important for users to understand *what are included in clusters*. Furthermore, topic labels should be sub-concepts under a search query, so that users understand *why clusters are presented*. For example, topic label “tool” would be inappropriate for the query [weather app], because “tool” is a parent concept encompassing all “weather

Figure 3. The knowledge-graph-based clustering process, which has three major steps, to generate app clusters for search query [games].

apps”. This is a great challenge for topic-model-based clustering algorithms, which typically do not capture hierarchical semantic relationships perfectly.

- **P2: Topical coherence within clusters and separation between clusters.** This is a common objective but also a long standing challenge for all document clustering algorithms. For example, query [game] should not cluster “blackjack” and “angry bird” together when there are other arcade games and card games available.
- **P3: Appropriate granularity of clusters.** High granularity results in small number of cluster, and low granularity results in large number of clusters. Granularity of clustering is query dependent, therefore number of clusters is also query dependent. For example, a “Blackjack” cluster is suitable for search query [card games] but might not be for search query [games], because it is too fine-grained. In recent work, Scaiella et al. asked Amazon Mechanical Turkers to evaluate topicality of search result clusters from their proposed system and two commercial systems, and found that around 20% of clusters should be merged [17].
- **P4: Balanced cluster sizes.** The sizes of clusters, i.e., number of apps in clusters, should be balanced. If one or two clusters contain most search results while the other ones contain only few results, the interface essentially degenerates to plain lists of search results.

- **P5: Clusters should cover most search results.** It is a common practice to include an “other” cluster that contains results not belonging to any cluster. However, when clusters do not cover many results, users have to find targeted result in a very big “other” cluster, defeating the goal of presenting clusters.

Generating clusters with these properties (especially *P1*, *P2* and *P3*) poses a great challenge for the algorithm, but where artificial intelligence struggles human intelligence may excel. For example, experts can judge whether a label describes apps in a cluster, understand concept hierarchy between cluster labels and search query (e.g., “tool” is an inappropriate cluster for search query “weather app”), and evaluate whether apps in one cluster are topically coherent, while algorithms face challenges for many of these tasks.

THE APPGROUPER SYSTEM

We build an interactive clustering system, AppGrouper, that incorporates expert decisions into multiple stages of a knowledge-graph-based clustering algorithm. In this section, we describe several key aspects of AppGrouper, including the clustering algorithm, the user interface, and ways of eliciting and using expert input.

Knowledge-Graph-based Clustering Algorithm

Instead of directly clustering apps, the algorithm extracts topic labels (mapped to concepts in knowledge bases such as DBpedia¹) from search results, runs clus-

¹<http://wiki.dbpedia.org/>

tering on a semantic graph of topic labels, and assigns search results to topic clusters. Knowledge-graph-based clustering algorithm have been somewhat explored in prior work [17].

This approach is made possible with increasingly accurate topic annotation systems, capable of extracting topics from text [8], image [19] and even video [2]. In our system, the topic extraction algorithm is a state-of-the-art entity recognition algorithm that runs over the app meta-data, including its title, descriptions, reviews, etc. The result of the entity recognizer is then fed into a regression-based ensemble learner that generates the most likely topic labels.

In describing the clustering algorithm, we introduce the following notations:

$A = \{a_1, \dots, a_n\}$ denotes the set of apps returned by search engine for a query;

$T = \{t_1, \dots, t_m\}$ denotes the set of topic labels extracted from A using a topic annotator system for apps;

$C_t = \{T_1, \dots, T_k\}$ denotes k clusters of topic labels; $C_a = \{A_1, \dots, A_k\}$ denotes k app clusters.

The clustering algorithm optimizes for generating app clusters with previously introduced properties, which we formally define as follows:

- **P2:** Apps within the same cluster are coherent and apps in different clusters are distinct:

$$coherence(C_a) = \sum_{A_x \in C_a} \sum_{a_i, a_j \in A_x} sim(a_i, a_j) \quad (1)$$

$$separation(C_a) = \sum_{a_i \in A_x, a_j \in A_y} [1 - sim(a_i, a_j)], \quad A_x, A_y \in C_a \quad (2)$$

- **P3:** Granularity of clusters is query-dependent, and choosing the right numbers of clusters for different queries is difficult. We penalize clustering results that have too many or too few clusters with the following equation (σ_{lower} and σ_{upper} are lower bound and upper bounds).

$$num_clusters(C_a) = \begin{cases} \sigma_{lower} - |C_a|, & |C_a| < \sigma_{lower} \\ |C_a| - \sigma_{upper}, & |C_a| > \sigma_{upper} \\ 0, & otherwise \end{cases} \quad (3)$$

- **P4:** App clusters have balanced sizes:

$$balance(C_a) = max_{A_x \in C_a} |A_x| - min_{A_y \in C_a} |A_y| \quad (4)$$

- **P5:** App clusters cover most search results:

$$coverage(C_a) = \sum_{A_x \in C_a} |A_x| \quad (5)$$

Since we want to generate a clustering result that simultaneously optimizes all the objectives above, we formulate the problem as a multiple objectives optimization, defining the objective function as weighted sum of the above properties:

$$\alpha_1 \cdot coherence(C_a) + \alpha_2 \cdot separation(C_a) + \alpha_3 \cdot num_clusters(C_a) + \alpha_4 \cdot balance(C_a) + \alpha_5 \cdot coverage(C_a) \quad (6)$$

We set the weights $\alpha_1 \dots \alpha_5$ and the range of number of clusters $[\sigma_{lower}, \sigma_{upper}]$ through human evaluations, aligning the objective function with expert-judged quality of clustering results.

To generate app clusters optimized for the objective function (6) defined above, we design a multi-stage clustering process, which we will describe with an example shown in figure 3:

1. In Step 1, we obtain topic labels for the search result apps by using a topic annotation system. The topic semantic relation graph is used to construct the topic label graph (as shown in the figure for the search query [games]). The weights of edges in the graph reflect the topical relevance between nodes, computed as the similarity between sets of apps covered by two topic labels.
2. In Step 2, after obtaining the topic label graph, we run a standard clustering algorithm, Hierarchical Agglomerate Clustering (HAC for short), on the graph. Like most other clustering algorithms, HAC requires a hyper-parameter, which is the minimum similarity threshold that two nodes can be merged into same cluster in the process of building a hierarchical tree. We exhaustively search for the optimal value for this parameter that maximizes the objective function (6) by running HAC with varying values of the threshold. In the example, this optimal result has four clusters.
3. In Step 3, to obtain balanced clusters, we filter out clusters that cover too few apps (the “simulation” cluster) and then we rank order the topic clusters. We rank the topic clusters by assigning apps to those clusters and computing the quality of the clusters based on the app assignments. The rank order of the topic clusters affects numbers of apps assigned to clusters, because we assign apps to the ranked clusters in a greedy way. In ranked order, apps are assigned to the first cluster that matched its topic label. For example, in the figure, we assign all apps that are associated with “arcade games” to the first cluster, then remaining apps that are associated with either “casual games” or “prank” to the second cluster and finally remaining apps that are associated with “action games”. To obtain the ordering that maximizes balance, we exhaustively search all orderings. Finally, we make a “more” cluster that includes all apps not included in app clusters.

Figure 4. A flow chart with expert decisions in the clustering process. Parallelograms represent data, while rectangles represent process. The three orange boxes on the bottom show expert input in the clustering process in early, mid and late stages.

This clustering process has a couple of advantages over past search result clustering search results approaches. First, the process generates topic labels that describe app clusters; human can easily understand and give critique on clusters of topic labels. Second, knowledge-graph-based clustering is computationally more efficient than computing the clustering on the fly, because clustering systems can store the relative stable topic clusters and assign dynamic search results on the fly to these clusters. For example, users would expect to see the subtopics “action games” and “arcade games” for the query [games], even if the pool of search result changes over time. If humans make edits on topic label cluster, these edits have a long lasting effect.

User Interface

AppGrouper, shown in Figure 2, has a visual interface that lets users load pre-computed clusters and interactively generates new clusters using the algorithm previously described. The interface has three main components: a sidebar on the left (A in figure 2), panels in the middle (B in figure 2) and clusters on the right (C in figure 2).

The panels (B in figure 2) in the middle affect the clusters shown on the right. From top to bottom, the first panel lets users run clustering on a user-specified search query and indicate a preference on the number of clusters; the second panel lets users configure the language of apps shown in clusters; the third panel presents users all clustering results previously computed and lets users search for and select clusters to show.

The right side (C in figure 2) of the interface shows clusters with topic labels for a search query (either newly computed or loaded from previously computed results). Figure 2 shows the clustering result for the query [English]. There are two topic label clusters - one having topic labels “Vocabulary” and “Grammar” and another having topic labels “Phrase” and “Sentence”. Each topic cluster has an algorithmically-chosen cluster label (e.g., “Vocabulary”) — the topic label that covers most apps in each cluster. The green round boxes contain measurements of clusters’ various properties, e.g., coverage of apps and coherence of apps. There is an app cluster with label “More” on the bottom, containing all apps returned from search engine. By clicking on the radio buttons beneath the title of clustering result, users can

change the number of apps to show in each cluster. We will describe how users can modify clustering results with other buttons on this panel in the following section.

The left sidebar (A in figure 2) shows all the topic labels in the topic label graph, excluding labels already included in clusters. We will elaborate in the following section how users use this list of topic labels.

Expert Decisions in the Clustering Process

Using AppGrouper, users can inject their judgments at various stages of the clustering process, as shown in Figure 4. As previously discussed, the algorithm is optimized for a set of weighted clustering properties. Expert decisions can help to ensure that *topic labels are semantically meaningful for app clusters*, *app clusters are topically coherent* and *the granularity of clusters is appropriate*, while making the tradeoff decisions between these requirements. We describe the way which we integrate expert judgments in the clustering process below.

Early stage. When the algorithm builds a topic label graph, users can “blacklist” semantically inappropriate topic labels (P1), which contribute to low-quality clustering results. For example, if the topic label “game” is included in topic label graph for the query [games for girls], a “game” cluster will most likely be generated. This is bad because “game” is a broader concept than the original query and this “game” cluster will likely include all search results.

Figure 5 shows the left sidebar, zoomed in, depicting a list of topic labels, which are sorted by the percentages of apps covered by that topic, shown to the right of labels. Users can click on topic labels to check their definitions, click on the cross buttons to add labels to blacklist and click on “blacklist and rerun” to re-execute the clustering process.

Mid stage. When the clustering algorithm generates clusters from the topic label graph, users can indicate preferences on a larger or smaller number of clusters, resulting in more or less fine-grained clusters (P3). The predetermined range for the number of clusters ([8, 12]) does not work for queries that have a very narrow or a very broad concept. For example, we find that three clusters (“weather service”, “temperature” and “digital data”) cover most apps related to the query [weather app] and have appropriate granularity. In other cases, we

Figure 5. The interface to blacklist topic labels.

see algorithm generated “mega clusters” (large clusters that include many incoherent topic labels), and we break them by increasing the number of clusters. By giving users control over the granularity of clusters, they can make case-by-case adjustments.

Figure 9 shows the panel for adjusting the number of clusters. Users can choose “Fewer” or “More” clusters and click on “run” to get a clustering result with coarser or finer granularity. Under the hood, when users make these selections, we change the range $[\sigma_{lower}, \sigma_{upper}]$ $[4, 8]$ for “Fewer” and $[12, 16]$ for “More”) in equation 3.

Figure 6. The interface to adjust the granularity of clusters.

Late stage. After the clustering algorithm runs, we present the topic label clusters together with the resulting app clusters and let users make direct edits, including adding/removing clusters, adding/removing topic labels of the clusters, and editing the user-facing names for the clusters. In contrast to early and mid stages’ human input, which trigger rerunning clustering algorithm on topic label graph, directly editing topic clusters results in instant and deterministic changes. Experts use this functionality to make minor adjustments to the clusters. After manual edits on the clustering, we pass the edited clusters to another algorithmic step to rank-order the clusters, which is a step that is too tedious for experts.

Figure 7 shows the zoomed-in app cluster panel in figure 2. Users can make edits as follows: (1) delete a topic label from a cluster or even an entire cluster by clicking on the deletion buttons highlighted in two smallest red boxes; (2) add a cluster by clicking on “add cluster”

button and add topic labels to a cluster by entering the ids of labels, which are listed in the sidebar (figure 5); (3) rewrite names for clusters by double-clicking on the box highlighted at the top. Whenever users add/remove a cluster or topic label, the system re-assigns the apps in all affected clusters instantly, keeping users informed of the effect of their actions.

Figure 7. The interface to make direct edits on topical cluster.

EVALUATION

We evaluated AppGrouper with domain experts. Apart from testing our hypothesis that expert input improves clustering quality, we are curious about how experts use AppGrouper and whether they show different patterns of usage for different queries. For example, perhaps broadness of the query might affect how the system is used to guide the clustering algorithm. We have the following research questions to guide our evaluation.

- **RQ1:** Does AppGrouper help a domain expert to improve algorithm generated clustering results?
- **RQ2:** How does the expert use AppGrouper?
- **RQ3:** Does the expert interact with the clustering process in the same way for all queries? Does the broadness of queries matter?

Setup

The participants in our evaluation are domain experts, the targeted users of AppGrouper. These domain experts are engineers who are tasked with making app clusters for search queries in Play Store. In their previous practice, internal experts generated app clusters with algorithm, filtered in only high quality results, made minor changes on cluster titles and shipped the clusters to production.

During the week of September 14, 2015, we deployed AppGrouper with our domain expert who used the system to generate clustering for a wide range of search queries. We randomly sampled 82 search queries, 41 broad queries and 41 narrow queries, from top 1000 search queries in Play Store. We define broad and narrow queries as follows: we sort queries based on number of topic labels associated and categorize the top 25% queries as broad and the bottom 25% as narrow.

Figure 8. Comparison of ratings on algorithmically-generated app clusters and ratings on expert-guided app clusters using AppGrouper. Clusters generated with expert guidance received higher rating than those without. The colored areas in boxes indicate the percentage of clustering results with certain rating.

The expert’s task was to use AppGrouper to make topic clusters for these 82 queries, each of which is expected to take 3 minutes on average. At the end of the task, we had a 30 minute interview with the domain expert. The interview was recorded for later analysis. Note that the expert was not allowed to rewrite the names for clusters, which would make the clustering easily recognizable as human-manipulated.

Another domain expert (referred to as “judge”) compared the quality of these 82 clusters with the quality of clusters generated from the clustering algorithm alone. For each search query, we asked the judge to blindly rate the quality of two randomly ordered clusters on a five-point Likert scale (1–5, where 5 means high quality).

Findings

Through analysis of quality evaluations on app clusters and usage log of AppGrouper, we summarize the findings as follows.

AppGrouper helped improve quality of clustering results.

Figure 8 shows that the second expert rated expert-guided app clusters more highly than algorithm-generated app clusters. A Wilcoxon Signed-Ranks Test indicates that the median ratings on app clusters using AppGrouper, $median = 4$, are significantly higher than the median ratings on algorithm generated initial app clusters, $median = 3$ ($Z = 403.5, n_1 = n_2 = 82, p < 1e - 6$).²

Another result reinforces the finding from a different perspective: 68.3% of the results are evaluated to be better; 14.5% of the results are evaluated to be unchanged; only 17.1% of the results are evaluated to be worse after edits made through AppGrouper. In a related manuscript under preparation, we show that having two experts reaching agreement on which two clustering results is better is actually quite difficult, with Cohen’s Kappa as low as

²Because ratings on five-point Likert scale is ordinal, Wilcoxon Signed-Ranks Test is more appropriate than a t-test.

0.33. In other words, when the expert made an improvement to algorithmically-generated clusters, the improvement appears to be significant.

The expert mainly guided in the late stage of clustering process.

Our analysis of the server log of AppGrouper reveals how the expert used AppGrouper to interactively make clusters. Table 1 shows that the expert made changes in the late stage, i.e., added/removed cluster or topic labels in clusters, for 98.8% of the queries, while only gave input to clustering process in early and mid stages for less than 20% of the queries.

A plausible explanation is that guidance operations are more costly in early and mid stages. Indeed, as shown in Table 1, both blacklisting topic labels in early stage and tuning the granularity of clusters in the mid stage have significantly higher system response time than directly editing topic label clusters in late stage, because these actions caused rerunning of the clustering algorithm.

Stage	Early	Mid	Late
Median response time (s)	12	9	0
% queries w/ human input	13.4%	19.5%	98.8%

Table 1. Response times of user actions and percentage of queries receiving human guidance in early, mid and late stages of clustering process. User actions in early and mid stage have significantly higher delay than that in late stage. The expert more frequently gave input in late stage than early and mid stages.

The expert used AppGrouper differently for broad and narrow queries.

As we expected, the algorithm generates more clusters for queries with broad concepts than for queries with narrow concepts (shown in figure 9). This difference is statistically significant using MannWhitney test ($U = 319, n_1 = n_2 = 41, p < 1e - 6$, one-sided). When judged by humans, the quality of app clusters for narrow queries, $median = 2$, is significantly lower than the quality of app clusters for broad queries, $median = 3$, (MannWhitney $U = 576.5, n_1 = n_2 = 41, p < 0.01$, one-sided), as shown in figure 10.

Figure 9. Comparison of number of clusters generated by algorithm for broad and narrow queries. Broad queries on average have more clusters than narrow queries.

Figure 10. Comparison of quality ratings on algorithmically-generated app clusters for broad and narrow queries.

These different properties result in different user behavior in modifying clusters. Though the expert spent slightly more time on broad queries (figure 11a), the expert made more edits on narrow queries (shown in figure 11b). This difference is marginally significant (Mann-Whitney $U = 691$, $n_1 = n_2 = 41$, $p = 0.08$, one-sided).

Our hypothesis is that the number of clusters and the quality of results both contribute to this behavior. On the one hand, broad queries on average have more clusters, requiring more cognitive effort and time to evaluate the clustering results. On the other hand, lower-quality narrow queries took more time to edit. When comparing the type of actions, we find that the expert is much more likely to blacklist topic labels and adjust the number of clusters on narrow queries than on broad queries (table 2): 70% cases when the expert gave input in early and mid stages were on narrow queries.

	Early stage	Mid stage	Late stage
broad	18.2%	25%	50%
narrow	81.8%	75%	50%

Table 2. Breakdown of actions in different stages on broad and narrow queries. The engineer more frequently interacted with clustering process in early and mid stages on narrow queries.

DISCUSSION

AppGrouper improves quality of app clusters.

The interview corroborates our quantitative finding that interactive clustering using AppGrouper improves quality of clusters generated from algorithm. We asked the expert what difficulties he had working with algorithm generated clusters and whether AppGrouper was helpful. The expert responded “the major pain point was that I had to throw away a lot of [algorithm generated] results. I find it [AppGrouper] super helpful, it helps me to achieve what was not possible.”. Using AppGrouper, these engineers can edit topic label clusters and “look at the app clusters being updated in real time”, allowing them to “be experimental” and “higher chance to save a result”.

Most results only require minor changes.

Why did the expert edit 98% of the queries in the late stage of the clustering process? In our log analysis, we

find that the expert mainly edited clustering result by adding or deleting clusters or topic labels. We suspect that the long response time of making changes in the early and mid stages contributes to this behavior, so we asked the expert about his strategy of choosing what features to use. The expert gave a somewhat surprising response — “Time is not a issue. Quality is more of a concern. I do not choose features [to use] based on time. At least half of the queries are good, only requiring remove a topic label or a cluster.”.

The expert also appreciated the feature that filters and reorders user-modified clusters when saving clusters: “The clusters are ordered; ranking is important. I trust the algorithm (on filtering and reordering).”

The expert gave input in the early and mid stages of the clustering process for hard queries.

Why was the expert 3 times more likely to blacklist topic labels and adjust the granularity of clusters when editing lower quality narrow queries than editing broad queries? The expert explained the situations to use these two features: “[clustering results for] queries that can not reach quality standard by my manual edit. These are usually hard queries.”. Narrow queries on average have worse initial quality (“hard queries”), therefore the expert was more likely to resort to changing clustering results in the early and mid stages. For blacklisting topic labels, “I blacklist topic labels that are synonyms of [the original] query, topic labels that have larger concept than query, and dominating topic label, say, topic label that cover over 60% apps.” For adjusting granularity of clusters, “I change number of clusters when I see mega cluster [cluster that has many incoherent topic labels] or too few clusters on a broad query.”

The expert optimizes his effort in improving the quality of app clusters.

Operations in the early and mid stages have uncertain outcome, while operations in the late stages have deterministic outcome. The expert commented “It sometimes does not work the way I want it, sometimes generates surprisingly good result.”. The setup of our evaluation experiment, using AppGrouper to edit app clusters in limited time, mimics real tasks the internal experts face in their work. The expert naturally tried to reduce his effort on each clustering result while improving quality. In this circumstance, we observed that the expert preferred edits that have a deterministic outcome, and was only willing to take actions that have an uncertain outcome for difficult cases.

DESIGN IMPLICATION

AppGrouper is a system that enables human guidance during an otherwise-automatic clustering process, leveraging human wisdom to tackle challenges inadequately addressed by the clustering algorithm. AppGrouper is designed to the specific application of making clusters for search results in Play Store, but can be extended for other Search Results Clustering problems and other

Figure 11. Comparison of time and number of actions on app clusters for broad and narrow queries. The expert spent more time (on log scale) on broad queries, but made more edits to narrow queries.

closely-related applications, such as topic modeling and knowledge extraction [9].

How to design systems that efficiently incorporate human judgment into machine learning is an open research area. By building AppGrouper and observing how it is used by domain experts, we summarize our key lessons as:

- **Make the machine learning process transparent.** Appropriate abstraction and visualization of the underlying machine learning process helps users understand both the strengths and the weaknesses of the process; helping users to give the relevant feedback. In our case, knowledge-graph-based clustering enables users to operate on clusters of topic labels, which are easier to understand and edit than clusters of discrete apps. Providing feedback at several stages while trusting the algorithm performing most tedious tasks on clustering, ranking, and filtering lets users focus on areas the machine learning process needs the most help.
- **Expose appropriate level of complexity.** Admittedly, not all machine learning processes can be easily abstracted and understood by average users. Thus, the system should be designed according to the expertise of users. In our case, we designed interactions at various stages of the algorithm, which enables users to engage at the appropriate control level with the clustering process.
- **Give users control.** This is perhaps the greatest challenge in designing an interactive machine learning system. Giving users control of and clear feedback about their actions are desirable features of good UI, but interactive machine learning system sometimes can not offer these, due to the computation delay and uncertainty of the underlying machine learning algorithm. We observed that an expert preferred making edits that have deterministic outcome over using features that might result in surprisingly good outcomes.

We also observed that the expert tended to interact in earlier stages when clustering tasks are hard.

CONCLUSION AND FUTURE WORK

This work makes three major contributions:

- We designed AppGrouper, an interactive system for guiding the topic clustering of app search results in Google Play Store.
- We identified key challenges in making high-quality clusters that can be served in a real production system. We proposed a knowledge-graph-based clustering process that groups topic labels from search results into topic clusters, and then assigns the apps to these topic label clusters. We designed features that enable human guidance at various stages of the clustering.
- We deployed AppGrouper and evaluated it with domain experts, finding that clustering results generated from the interactive clustering process are significantly higher quality than those from a pure algorithmic process. Our observations on the pattern of usage of AppGrouper offers several lessons learned for designing interactive machine learning systems.

In the future, we plan to explore ways to improve machine learning based on human guidance. We believe that the role of human experts in a mixed-initiative system can go far beyond correcting mistakes made by machines. Instead, together in a symbiotic working relationship, the sum is greater than its parts.

ACKNOWLEDGEMENT

I thank Google for providing me the internship opportunity. I appreciate help from Jilin Chen, Zhiyuan Cheng, Sagar Jain in this project and their support during my internship. I also want to thank Max Harper, Daniel Kluver and Vikas Kumar from Grouplens research at University of Minnesota for their feedback on this manuscript.

REFERENCES

1. Saleema Amershi, Bongshin Lee, Ashish Kapoor, Ratul Mahajan, and Blaine Christian. 2011. CueT: Human-guided Fast and Accurate Network Alarm Triage. In *CHI*.
2. Hrishikesh Aradhya, George Toderici, and Jay Yagnik. 2009. Video2Text: Learning to Annotate Video Content. In *IEEE International Conference on Data Mining Workshops*.
3. Pranjal Awasthi, Maria-Florina Balcan, and Konstantin Voevodski. 2014. Local algorithms for interactive clustering. In *ICML*.
4. Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. 2004. Active Semi-Supervision for Pairwise Constrained Clustering. In *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM-04)*.
5. Sumit Basu, Danyel Fisher, Steven M. Drucker, and Hao Lu. 2010. Assisting Users with Clustering Tasks by Combining Metric Learning and Classification.. In *AAAI*. AAAI Press.
6. Claudio Carpineto, Stanislaw Osiski, Giovanni Romano, and Dawid Weiss. 2009. A survey of Web clustering engines. *Comput. Surveys* 41 (2009).
7. Claudio Carpineto and Giovanni Romano. 2010. Optimal Meta Search Results Clustering. In *SIGIR*.
8. Shuo Chang, Peng Dai, Jilin Chen, and Ed H. Chi. 2015. Got Many Labels?: Deriving Topic Labels from Multiple Sources for Social Media Posts using Crowdsourcing and Ensemble Learning. (2015).
9. Jason Chuang, Sonal Gupta, Christopher D. Manning, and Jeffrey Heer. 2013. Topic Model Diagnostics: Assessing Domain Relevance via Topical Alignment. In *International Conference on Machine Learning (ICML)*.
10. Marie desJardins, James MacGlashan, and Julia Ferraioli. 2007. Interactive Visual Clustering. In *IUI*.
11. Jerry Alan Fails and Dan R. Olsen, Jr. Interactive Machine Learning. In *IUI*.
12. Paolo Ferragina and Antonio Gulli. 2005. A Personalized Search Engine Based on Web-snippet Hierarchical Clustering. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*.
13. James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. 2008. CueFlik: Interactive Concept Learning in Image Search. In *CHI*.
14. Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *IUI*.
15. Stanislaw Osinski and Dawid Weiss. 2005. A Concept-Driven Algorithm for Clustering Search Results. *IEEE Intelligent Systems* 20, 3 (May 2005), 48–54.
16. Peter Pirolli, Patricia Schank, Marti Hearst, and Christine Diehl. 1996. Scatter/Gather Browsing Communicates the Topic Structure of a Very Large Text Collection. In *CHI*.
17. Ugo Scaiella, Paolo Ferragina, Andrea Marino, and Massimiliano Ciaramita. 2012. Topical clustering of search results. In *WSDM*. ACM Press, New York, New York, USA.
18. Jesse Vig, Shilad Sen, and John Riedl. 2011. Navigating the Tag Genome. In *IUI*.
19. Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. WSABIE: scaling up to large vocabulary image annotation. (2011).