

# Discovering Play Store Reviews Related to Specific Android App Issues

by

Angshuman Ghosh

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Computer Science

Waterloo, Ontario, Canada, 2018

© Angshuman Ghosh 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Mobile App reviews may contain information relevant to developers. Developers can investigate these reviews to see what users of their apps are complaining about. However, the huge volume of incoming reviews is impractical to analyze manually. Existing research that attempts to extract this information suffers from two major issues: supervised machine learning methods are usually pre-trained, and thus, does not provide the developers the freedom to define the app issue they are interested in, whereas unsupervised methods do not guarantee that a particular app issue topic will be discovered.

In this thesis, we attempt to devise a framework that would allow developers to define topics related to app issues at any time, and with minimal effort, discover as many reviews related to the issue as possible. Scalable Continuous Active Learning (S-CAL) is an algorithm that can be used to quickly train a model to retrieve documents with high recall. First, we investigate whether S-CAL can be used as a tool for training models to retrieve reviews about a specific app issue. We also investigate whether a model trained to retrieve reviews about a specific issue for one app can be used to do the same for a separate app facing the same issue. We further investigate transfer learning methods to improve retrieval performance for the separate apps.

Through a series of experiments, we show that S-CAL can be used to quickly train models that can to retrieve reviews about a particular issue. We show that developers can discover relevant information during the process of training the model and that the information discovered is more than the information that can be discovered using keyword search under similar time restrictions. Then, we show that models trained using S-CAL can indeed be reused for retrieving reviews for a separate app and that performing additional training using transfer learning protocols can improve performance for models that performed below expectation.

Finally, we compare the performance of the models trained by S-CAL at retrieving reviews for a separate app against that of two state-of-the-art app review analysis methods one of which uses supervised learning, while the other uses unsupervised learning. We show that at the task of retrieving relevant reviews about a particular topic, models trained by S-CAL consistently outperform existing state-of-the-art methods.

## Acknowledgements

First and foremost, I would like to thank my supervisors Professor Mei Nagappan and Professor Maura R. Grossman for their supervision during my graduate studies. I was allowed the opportunity to work on a problem that I found intriguing, and I am grateful for their support. I am also grateful for all the small pieces of advice and support offered to me during these two years, be it regarding my studies, or my future career choices.

Furthermore, I would like to thank Professor Gordon Cormack and Professor Mike Godfrey for agreeing to be my readers. However, Gordon acted as more than just a reader. It was his and Maura's class on High Stakes Information Retrieval that made this thesis possible. Additionally, Gordon went above and beyond by offering guidance regarding the technical aspects of my work even after I completed his course, and for that I am grateful. Both Mike and Gordon's inputs to my thesis are invaluable.

I would also like to thank Professor Haroon Malik (Marshall University), under whose guidance I interned at the University of Waterloo as a MITACS intern in the summer of 2015. He introduced me to the world of applied Software Engineering research which inspired me to return to the University of Waterloo as a Graduate Student. I would also like to thank MITACS for their financial support, both during my internship and during my graduate studies through their Graduate Fellowship Program.

I have found many friends during my time here at Waterloo who has enriched my life in many ways. My roommates through the two years Nimesh Ghelani, Kshitij Jain, Manan Dosi, and Shankha Shubhra Chatterjee were always there when I had a question. My peers at Waterloo, Eswar Yaganti, Aaron Sarson, Siddhart Sahu, Lakshmanan Arumugam, Chathura Kankanamge, Royal Sequeira and Rahul Iyer and Krishna Vaidyanathan were always there to grab dinner after a weary day at the university. Without all of you, my two years would have been a lot less colorful.

And last but not least, I want to thank my mother Sucharita Ghosh, my father, Malay Ghosh and my grandmother Gita Rani Ghosh. My dad, also an Engineer, inspired me to take up studies in a technical field. My mother taught me that adverse circumstances are never a reason to give up and that there is always light at the end of a dark tunnel and my grandmother taught me what unconditional love is. Despite them being an ocean and two continents away, their love and support enabled me to take this huge step in my journey, and it is their love and support that keeps me going forward. Thank you, Mom and Dad.

## **Dedication**

This is dedicated to my mom, my dad and my grandma, who nurtured me through my formative years and molded me into the person I am today.

# Table of Contents

List of Tables	ix
List of Figures	x
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem . . . . .	2
1.3 Our Approach . . . . .	3
<b>2 Background</b>	<b>4</b>
2.1 Preprocessing . . . . .	4
2.1.1 Tokenization . . . . .	5
2.1.2 Stemming . . . . .	5
2.1.3 Stopword Removal . . . . .	6
2.1.4 General Text Cleaning . . . . .	6
2.2 Indexing . . . . .	7
2.2.1 Inverted Index . . . . .	7
2.2.2 Term Document Matrix . . . . .	7
2.3 Retrieval and Ranking . . . . .	9
2.3.1 Boolean Retrieval . . . . .	9
2.3.2 Ranking . . . . .	9

2.4	Active Learning . . . . .	10
2.4.1	Active Learning Protocols . . . . .	11
2.4.2	Stopping Criteria . . . . .	12
2.4.3	Scalable Continuous Active Learning . . . . .	13
2.5	Transfer Learning . . . . .	14
2.6	Evaluation Metrics . . . . .	16
<b>3</b>	<b>Proposal and Study Design</b>	<b>18</b>
3.1	Experiments . . . . .	19
3.1.1	User Experiment . . . . .	19
3.1.2	Transfer Experiment . . . . .	19
<b>4</b>	<b>User Experiment</b>	<b>20</b>
4.1	User Study Design . . . . .	21
4.1.1	Order of Assignment . . . . .	21
4.1.2	Instructions for Participants . . . . .	22
4.1.3	Recruitment of Participants . . . . .	23
4.2	Data Set . . . . .	23
4.2.1	User Study Topics . . . . .	25
4.3	System Description . . . . .	25
4.3.1	S-CAL Interface . . . . .	25
4.3.2	Search Interface . . . . .	27
<b>5</b>	<b>Transfer Experiment</b>	<b>30</b>
5.1	Data Set . . . . .	30
5.2	Experiments . . . . .	32
5.2.1	Direct Transfer . . . . .	32
5.2.2	Additional Training . . . . .	32
5.2.3	Additional Sampled Training . . . . .	33
5.2.4	Comparison Study . . . . .	33

<b>6</b>	<b>Results</b>	<b>38</b>
6.1	User Experiment . . . . .	38
6.1.1	S-CAL Precision . . . . .	38
6.1.2	Unique Documents . . . . .	40
6.1.3	Inter-Participant Agreement . . . . .	43
6.2	Transfer Experiment . . . . .	45
6.2.1	Direct Transfer . . . . .	45
6.2.2	Additional Training . . . . .	48
6.2.3	Additional Sampled Training . . . . .	51
6.2.4	Comparison Experiment . . . . .	54
6.2.5	Discussion . . . . .	59
<b>7</b>	<b>Related Works</b>	<b>60</b>
7.1	Mobile App Review Analysis . . . . .	60
7.1.1	Classification . . . . .	61
7.1.2	Automated Information Extraction . . . . .	62
7.1.3	Applied Analysis . . . . .	64
7.2	Active Learning for High-Recall Search . . . . .	67
<b>8</b>	<b>Conclusion</b>	<b>69</b>
8.1	Contributions . . . . .	69
	<b>References</b>	<b>71</b>
	<b>APPENDICES</b>	<b>79</b>
<b>A</b>	<b>Participant Instructions</b>	<b>80</b>
A.1	General . . . . .	80
A.2	Search Interface . . . . .	80
A.3	S-CAL Interface . . . . .	81



# List of Tables

2.1	Illustration of Inverted Index with Positional Meta-data . . . . .	8
2.2	Tf-Idf Matrix According to Formula 2.1, Normalized to a Unit Vector . . . . .	9
4.1	Assignment of Each App to Participants, Ensuring All Possible Permutations were Assigned Exactly Once . . . . .	21
4.2	Title and Descriptions of Topics Used in the User Experiment . . . . .	24
5.1	Test Set Apps . . . . .	31
6.1	R-Precision of Models Trained Using S-CAL Used Directly for Ranking on Test Set . . . . .	47
6.2	R-Precision of Models with Additional Training Using Target App Reviews on Test Set . . . . .	51
6.3	R-Precision of Models with Additional Training Using Randomly Sampled Target App Reviews on Test Set. $2R$ and $5R$ indicates the number of randomly sampled negative examples in the training set . . . . .	54
6.4	R-Precision of Ranked Lists Returned by ALPACA. * The values for ‘Auto-Trader’ are values of recall, since ALPACA returned fewer than $R$ reviews in total. . . . .	55
6.5	Statistics from CLAP - Number of Clusters, Precision and Recall . . . . .	59

# List of Figures

2.1	General Document Processing Pipeline . . . . .	4
2.2	Active Learning Workflow . . . . .	11
2.3	Expected Advantages of Transfer Learning. Image Taken from Transfer Learning [61] . . . . .	15
4.1	Example of an User Review Containing a Star Rating, Title and Comment	23
4.2	S-CAL Interface Showing a MapMyWalk Review . . . . .	26
4.3	Interface for Keyword Search Showing Results for JustWink App . . . . .	28
5.1	ALPACA Final Output for the Autotrader App, Showing the Expanded Keyword Set and Retrieved Reviews . . . . .	35
5.2	CLAP Tool Interface, Showing Output for Stupid Zombies 2 app, Focused on Feature Requests . . . . .	37
6.1	Trends for P@K for $1 \leq K \leq 100$ . . . . .	39
6.2	Box Plot Showing the Number of Unique reviews Found by S-CAL( $S_1$ ), Keyword Search( $S_2$ ), and Found by Both( $S_3$ ) . . . . .	42
6.3	Box Plot Showing Agreement Between Participants Using S-CAL, Keyword Search, and Between S-CAL and Keyword Search Interfaces . . . . .	44
6.4	Plot Showing Recall at $R$ , $2R$ , and $3R$ of Models Trained Using S-CAL Used Directly for Ranking on Test Set . . . . .	46
6.5	Plot Showing Recall at $R$ , $2R$ , and $3R$ of Models Trained Using S-CAL with Additional Training on 100 Target App Reviews on Test Set . . . . .	49

6.6	Plot Showing Recall at $R$ , $2R$ , and $3R$ of Models Trained Using S-CAL with Additional Training on 200 Target App Reviews on Test Set . . . . .	50
6.7	Plot Showing Recall at Thresholds of $R$ , $2R$ , and $3R$ on Test Set with Additional Training on $R$ Relevant and $2R$ Non-Relevant Target App Reviews	52
6.8	Plot Showing Recall at Thresholds of $R$ , $2R$ , and $3R$ on Test Set with Additional Training on $R$ Relevant and $5R$ Non-Relevant Target App Reviews	53
6.9	Plot showing Recall at Thresholds of $R$ , $2R$ , and $3R$ for ALPACA. Here, expanded topic keywords were used to generate ranked list. ALPACA returned only 31 reviews for Autotrader, thus, only one point is shown. . . . .	56
6.10	Plot showing Recall at Thresholds of $R$ , $2R$ , and $3R$ for ALPACA. Here, topic keywords without expansion were used to generate ranked list. ALPACA returned only 27 reviews for Autotrader, so we only plot one point.	57

# Chapter 1

## Introduction

### 1.1 Motivation

The mobile application (“app”) market has experienced explosive growth over the last 10 years. Apps are distributed via app stores, which are central repositories for apps. The Google Play Store is the primary source of apps for phones that operate on Google’s Android operating system. Google announced the Android Market in 2008, and now, there are over 3.6 millions apps in the Google Play Store.<sup>1</sup> The Google Play Store provides two mechanisms for users to provide feedback about apps: ratings and reviews. Users can provide a rating for each app on a scale of one to five stars, and the average of these ratings is displayed for each app in the Play Store. Ratings are an overall measure of users’ perception of a particular app. Mobile app reviews, on the other hand, are the primary way for users to provide feedback to the developer, and often, they communicate complaints and feature requests through this medium. Mobile app reviews have been shown to contain information useful for developers [45]. Consider this review for the app “SavingStar - Grocery Coupons”.

“Disappointed....wish they would add these. So I noticed that a lot of grocery stores are listed but there is no Costco nor Safeway nor Trader Joe’s listed. We buy food from all three but yet none are listed. I have receipts I want to add but I can’t since it’s not listed. Please add these stores and make it fair for us that live in SF. \*\*\*\*\*will add more stars when these are listed...and I mean all of them are listed. Thanks.”

---

<sup>1</sup>The Android App Market became the Google Play Store in 2012.

This particular review identifies a specific issue that the user discovered while using the app, and provides a solution that they believe will improve their user experience. This information can be used by developers to improve their apps, or prioritize issues for the next release.

## 1.2 Problem

Extracting this information is a non-trivial task for two primary reasons.

1. Apps receive a considerable number of new reviews each day. A study conducted in 2013 [45] showed that the average number of reviews received by an app in one day is approximately 24, while popular apps like the Facebook mobile app received upwards of 4000 reviews in one day.
2. Additionally, these reviews are comprised of unstructured text. Research on extracting actionable information from these reviews uses advanced natural language processing techniques [8, 50, 29, 65], and is non-trivial task.

Existing research focuses on two major avenues.

1. Research using supervised learning aims to create a classifier to categorize app reviews into broad classes (e.g. ‘bug report’, ‘app praise’, ‘feature request’, etc). However, these methods can suffer from lack of granularity: reviews that complain about ‘app crashes’, ‘battery drainage’, and ‘slow interface’ get bundled into the same broad category: ‘bug report’.
2. Unsupervised learning methods have been used to cluster reviews and discover topics automatically, but these methods do not provide any guarantees. A particular topic may or may not get discovered, specially if the prevalence of this topic is low compared to other topics. It is also infeasible for a developer to go through a large number of reviews to discover information about a specific app issue that users describe in their reviews.

Thus, the challenges are to:

1. Provide app developers with information about a particular issue described in the reviews;

2. Discover as many reviews as possible that contain information about that issue; and
3. Ensure that minimal developer effort is spent in performing this task.

## 1.3 Our Approach

We investigate the use of Active Learning and Transfer Learning to create a framework for training and sharing models to rank reviews. Scalable Continuous Active Learning (S-CAL) [13] is an active learning protocol that can be used to train models to classify or rank documents for high recall search. In [13], the authors showed that S-CAL can be used to achieve high recall, optimize  $F_1$  scores, or simply train a model for classification or ranking. Our proposal investigates the use of S-CAL to allow developers to train models to rank reviews to identify app issues. This allows topics to be defined at any time, and a model can be trained to retrieve reviews for any topic. Additionally, a model trained for a specific issue present in a particular app can be used to rank reviews for a separate app facing the same issue. Our work attempts to investigate methods that would allow developers to discover as many reviews that contain information relevant to the issue being investigated, with minimal effort.

# Chapter 2

## Background

This chapter describes the various preprocessing, storage mechanisms and algorithms used in our experiments. Evaluation metrics are also discussed.

### 2.1 Preprocessing

Documents are generally stored as individual files. While this format is required for display purposes, it does not support the tasks of learning or retrieval. The raw text file is processed using a series of processes or filters where the output of each filter is passed to the next one. The output of the final filter is then stored using in a format suitable for the final task.

For illustration, we will demonstrate the preprocessing steps using the following sentences.

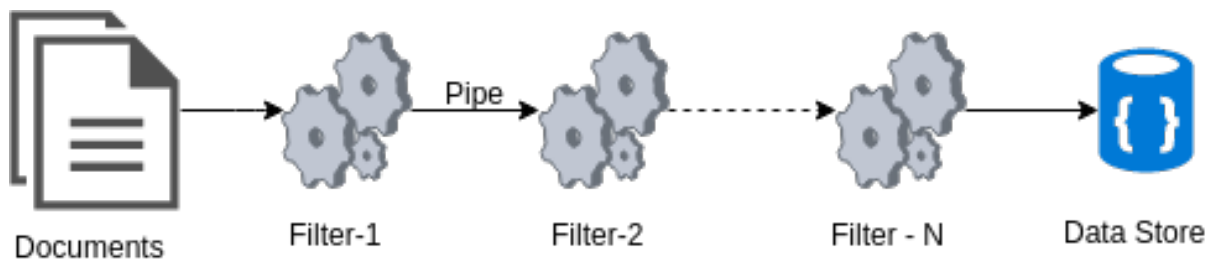


Figure 2.1: General Document Processing Pipeline

S1: These foxes can't be over 1.5m tall!

S2: The quick brown fox just jumped over the lazy cow.

### 2.1.1 Tokenization

Tokenization is the process of breaking a corpus of text into small fragments or words. For text in English, a white space or a punctuation mark is generally a good indicator of a word boundary. This tokenizer can be represented using the regular expression:  $[a - zA - Z0 - 9]^+$ . However, this creates two tokens for 'can't': 'can' and 't'. Now, if this was stored in an index, the query 'can' would match this snippet, even though it's semantic meaning is exactly the opposite. The Unicode text segmentation algorithm<sup>1</sup> is a rule-based tokenization algorithm that defines an extensive set of rules that covers multiple possible scenarios including words with apostrophes (e.g. 'can't'), numbers with decimal points or physical measurements (e.g. '1.5m'). The example sentence would thus tokenize as follows:

S1: These,foxes,can't,be,over,1.5m,tall

S2: The,quick,brown,fox,just,jumped,over,the,lazy,cow

One drawback to the two methods discussed here is that neither can tokenize emoticons.

### 2.1.2 Stemming

Stemming is the process of transforming words to their root form. The goal of this procedure is not to reduce words to their morphological roots, but to reduce words that have similar meanings to the same root (e.g. conjugations of a word should reduce to the same word stem). Algorithmic stemming is done by transforming the suffix of a word according to predefined rules. In information retrieval, the effect of stemming is similar to that of query expansion: by searching for one word, the user searches for all words that share the same root. This is the same as expanding the query to include more terms that have the same meaning. It also helps reduce the number of features for classification or clustering.

The Porter stemmer [68], an algorithmic stemmer, is popular choice for stemming. The effects of the Porter stemmer on the token stream produced by the unicode text segmentation algorithm follows:

---

<sup>1</sup>[Unicode Text Segmentation](#)



S1: These,fox,can't,be,over,1.5m,tall

S2: The,quick,brown,fox,just,jump,over,the,lazi,cow

There is a one important observation here: ‘lazy’ was stemmed to ‘lazi’, which is not the correct morphological root. However, for learning or ad-hoc retrieval, this is not necessary: as long as

### 2.1.3 Stopword Removal

Stopwords are words that are present in most documents. These words add negligible additional information to a document. Some examples are pronouns (e.g., he, she, it), prepositions (e.g., for, in, on, etc), conjunctions (e.g., and, but, etc) and articles (e.g., a, an, the). These words are usually removed from the token stream. The token stream after stopword <sup>2</sup> removal becomes:

S1: fox,1.5m,tall

S2: quick,brown,fox,jump,lazi,cow

One observation here is that standard stopword lists generally include words like ‘can’t’ or ‘doesn’t’, which might be of interest while mining app reviews. This is a potential drawback of using standard stopword lists for user review mining.

### 2.1.4 General Text Cleaning

Even after these three preprocessing steps, there are other general filter that can be applied to these streams of tokens. One example is reducing all characters to the lower case. This ensures that capitalized and non-capitalized versions of the words are not recorded or indexed as separate words. However, doing this indiscriminately has an unfortunate side effect for phrases such ‘United States of America’: ‘US’ present in text becomes the same as the pronoun ‘us’ which is a stopword in a high proportion of stopword lists available for use.

Synonym expansion is a filter that can save the word US from being mistaken. The functioning of the synonym expansion filter is similar to that of stemmers. This filter simply maps tokens defined in a dictionary and transforms them into one single token. Such a filter could replace all occurrences of the ‘US’, ‘United States of America’ or ‘U.S.A.’ to ‘United\_States’.

---

<sup>2</sup> Based on the list of stopwords at Ranks NL: <https://www.ranks.nl/stopwords>

## 2.2 Indexing

Indexing is the process of storing the stream of preprocessed tokens on persistent storage. While preprocessing text is similar for both machine learning and information retrieval, requirements for storage can vary. Machine learning requires a matrix with numerical or categorical values. Retrieval requires Boolean query matching, and ranking requires relevance scoring. Boolean query matching requires indexes which can be searched easily given a query. Relevance scoring requires a matrix similar to machine learning. Thus, multiple formats are required for data storage and are discussed below

### 2.2.1 Inverted Index

An inverted index is a data structure that provides for efficient operations for boolean keyword queries. An index built on documents can be inefficient: searching for a word would require searching for the word in each of the documents. An inverted index is built on words that act as keys. Thus, each unique word in the corpus becomes a key. These keys are sorted lexicographically and point to a list of identifiers of all documents in which the word is present. Such a list allows the storage of other meta-data as well. For example, positions of the word can be stored in a sorted order to allow proximity queries, whereas frequencies can be stored to calculate relevance scores for ranking.

The inverted index for the two illustrative sentences is shown in table 2.1. In this form, it is very easy to calculate term frequency of a word in a document, as well as the document frequency<sup>3</sup> of the word. Term frequency is simply the length of the position list, whereas the length of the document list is the document frequency.

### 2.2.2 Term Document Matrix

A term document matrix can be used both for ad-hoc retrieval and learning. The vector space model [6] uses a term document matrix for retrieval and ranking. For learning, bag-of-words representation is a popular method for storing and representing a document corpus.

Let  $V$  represent the vocabulary (list of all unique words) of the document corpus. Let  $D$  be the number of documents. The term document matrix is a matrix in  $\mathbb{R}^{V \times D}$ . The value in the  $i_{th}$  row in the  $j_{th}$  column depends on the presence of the  $j_{th}$  word in the

---

<sup>3</sup>Document frequency ( $df_w$ ) is the number of unique documents which the word  $w$  is present in

word	doc-list
1.5m	s1:2
brown	s2:2
cow	s2:6
fox	s1:1, s2:3
jump	s2:4
lazi	s2:5
quick	s2:1
tall	s1:3

Table 2.1: Illustration of Inverted Index with Positional Meta-data

vocabulary in the  $i_{th}$  document. This value can be Boolean indicator of presence (True if present), or a count of the number of times it was present in the document. Generally, the values are a function of multiple aspects, including the term frequency, and document frequency of the word. Term frequency  $tf_{w|d}$  refers to the number of times a word  $w$  occur in document  $d$ . Document frequency ( $df_w$ ) is the number of unique documents which the word  $w$  is present in. Intuitively, if a word  $w_i$  is present in the document  $d_j$  multiple times, it should have a higher score in the matrix. A function of term frequency deals can be used to expressed this in the score. On the other hand, if a word occurs only in a few specific documents, it is likely that it is an important word. This can be expressed in the score using a mathematical transform of the term frequency. Inverse Document Frequency ( $idf$ ) is simply the ratio of the number of documents in the corpus to the number of documents which contain the word  $w$ . This implies that the rarer a word is, the higher the idf is. The combination of tf and idf is often the choice of value for a term document matrix.

$$tf-idf_{w|d} = (1 + \log(tf_{w|d})) \times \log\left(\frac{D}{df_w}\right) \quad (2.1)$$

Formula 2.1 is a popular choice of function used to transform  $tf_{w|d}$  and  $df_w$  into a score for each term present in a document in the term document matrix. The term document matrix for the illustrative sentences is shown in table 2.2. The intuition for the log transform on the term frequency score is that the importance of a word does not necessarily linearly increase with the frequency of the word. A similar argument can be made for the inverse document frequency as well. These term document matrices are usually sparse, and are usually stored on disc using a sparse matrix. The *SVM<sup>light</sup>*<sup>4</sup> file format is a popular file format which is used for this task.

<sup>4</sup>*SVM<sup>light</sup>* file format

	15m	brown	cow	fox	jump	lazi	quick	tall
s1	0.63	0	0	0.44	0	0	0	0.63
s2	0	0.42	0.42	0.30	0.42	0.42	0.42	0

Table 2.2: Tf-Idf Matrix According to Formula 2.1, Normalized to a Unit Vector

## 2.3 Retrieval and Ranking

The primary goal of information retrieval is to fetch documents that satisfy an information need. For ad-hoc search, this is usually provided as a keyword query. Given a query and an index, the expected output is a list of documents ranked according to decreasing levels of relevance. Here, relevance is simply defined as the extent to which a particular document satisfies a given information need and documents satisfying this information need are considered relevant. The task of returning the ranked list can be broadly separated into two parts; candidate set identification, and ranking the documents in the candidate set.

### 2.3.1 Boolean Retrieval

The first step in forming a ranked list is retrieving a candidate set of documents to rank. This is necessary as ranking all documents in a corpus is an unnecessary drain on computational resources, specially if the query is only satisfied by a small percentage of the whole corpus. Let  $Q = q_1, q_2, ..q_n$  be a query with  $n$  different terms. A candidate set is created by examining an inverted index and appending all documents containing the query terms to a candidate list. Since the document identifiers are sorted in the inverted index, this candidate set can be created using the ‘merge’ step of the mergesort algorithm. For ‘AND’, only documents in present in the lists for all query terms are retained (intersection of sets), whereas for ‘OR’, all the documents are retained (union of sets). For example, for a query ‘fox AND tall’, only  $s1$  would be retained, while both sentences would be included in an ‘OR’ query

### 2.3.2 Ranking

Once a candidate set of documents is retrieved, they are scored using a function that calculates the ‘relevance’ score of each document  $d$  with respect to the query  $Q$ . Considerable

research has been devoted into designing various different types of ranking functions [5, 59]. A simple ranking function can be created using Formula 2.1.

$$Score(Q, d) = \sum_{q_i \in Q} tf \cdot idf_{q_i|d}$$

This score can be calculated by multiplying the document vector with a vector formed by preprocessing the query using the same set of filters as the document corpus. However, one issue with this method is that it tends to favor longer documents. BM25 [6] is a state of the art relevance ranking model which takes into account the length of the document.

$$Score(Q, d) = \sum_{i=1}^n \log \frac{N - df_{q_i} + 0.5}{df_{q_i} + 0.5} \cdot \frac{tf_{q_i|d} \cdot (k + 1)}{tf_{q_i|d} + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})} \quad (2.2)$$

Here, *avgdl* is the average document length, and  $|d|$  is the length of document  $d$ .  $k$  and  $b$  are hyper-parameters that can be tuned for accuracy. The second part of Formula 2.2 is restricted to between approximately 1 and  $k + 1$ . The score becomes  $k + 1$  if the  $tf_{q_i|d}$  tends to  $\infty$ , whereas the value is very close to 1 if the  $tf_{q_i|d} = 1$  and  $avgdl \approx |d|$ . This range of possible values ensures that extremely long documents with are not rewarded disproportionately compared to shorter documents.

## 2.4 Active Learning

Machine learning is usually classified into two broad categories: supervised learning and unsupervised learning. Supervised learning deals with the task of training a classifier when labeled data is available. Unsupervised learning deals with scenarios where labels are not available, and the goal is to identify patterns or cluster the data into meaningful groups. Active learning falls into the semi-supervised category and addresses situations where there is a lack of labeled data. If labels are necessary but unavailable in a supervised machine learning scenario, a sample of the entire dataset can manually labeled and then a classifier trained on it. However, in case of unbalanced data (where the prior probabilities of each class vary by a considerable amount), this becomes difficult. Random samples will have few, if any, examples of the class with lower prevalence. Active learning can be used to efficiently train models in such scenarios.

Active learning follows a cyclic procedure as shown in Figure 2.2. The algorithm queries the annotator for labels for specific examples and then trains a model on these labels. This

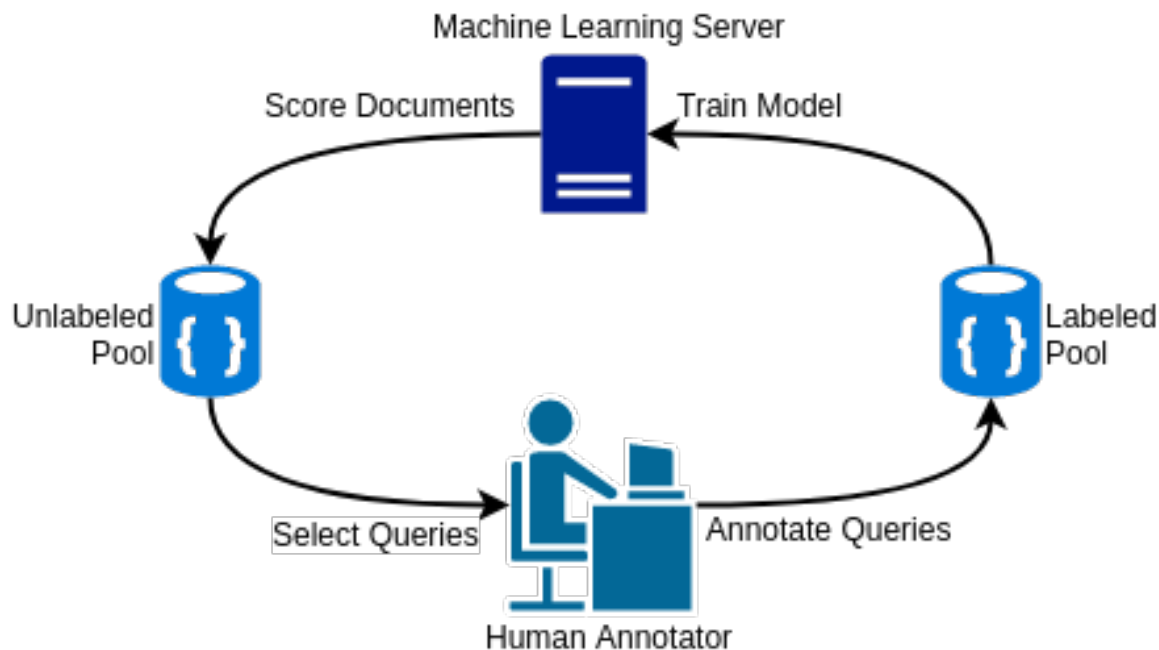


Figure 2.2: Active Learning Workflow

new model is used to select the next set of queries from the unlabeled pool, and the cycle continues until some stopping criterion is met (e.g., when there are no more informative examples left to label [64]), and a classifier is then produced. In our experiments we use active learning to model with highly unbalanced classes, and the minor class is the class we are interested in. In the rest of the thesis, we refer to examples of this class as ‘relevant’ or ‘positive examples’, whereas the other class is referred to as non-relevant, or negative examples.

### 2.4.1 Active Learning Protocols

In active learning, after each iteration, a model is trained using all examples for which labels are available. All unlabeled examples are scored using this model, and query selection for the next round of labeling is done using these scores. We discuss two methods of query selection. The nomenclature introduced in [12] has been followed here. In Simple Active Learning (SAL) or uncertainty sampling [37], the examples that are closest to the decision boundary of the classifier is chosen for annotation. These are the training examples that the algorithm is most uncertain about, and are simply the examples with scores closest

to the threshold. Continuous Active Learning (CAL), introduced in [14] takes a different approach, and selects those examples with the highest score (or most certainty) of belonging to the positive class. It can also be interpreted in the following way: CAL selects those examples which are most similar to the set of labeled examples. These training examples are thus less confusing than those selected by SAL. Therefore, it can be argued that annotating queries for CAL is easier than that for SAL. It can also be argued that an annotator can quickly discover more information about the topic using CAL. Since CAL requests label for the documents that are most likely to be relevant. Thus, important information is seen more quickly with CAL than SAL, since SAL focuses on the documents which are marginally relevant, or non-relevant (and thus, closer to the decision boundary).

## Baseline Model Implementation

The Baseline Model Implementation (BMI) is a version of CAL available publicly through a GNU license. It was released for use during the TREC Total Recall Tracks [54, 28]. The process is initialized with a user provided query. The negative class is created by randomly sampling 100 documents, which are temporarily deemed non-relevant for training the classifier. The classifier used is logistic regression. In each iteration, the number of labels requested is increased. All labeled documents are included in the training set and it is then augmented with 100 randomly selected documents. This process continues till all documents are labeled, or a maximum of 100 iterations are completed.

### 2.4.2 Stopping Criteria

Due to the lack of labels, determining when to stop active learning algorithms is not trivial. One method would be to look at accuracy on the training set (i.e., the set of examples labeled over multiple iterations of active learning). Stagnation in the change or steady decrease in accuracy over multiple consecutive iterations is a good indicator that the algorithm should be terminated. If the accuracy stabilizes, that suggests no new information can be gained by adding more training examples. In [15], two methods were proposed specifically for CAL. The knee method measures the slope of recall plotted against rank. If the ratio of the current slope compared to the slope at the start is higher than a threshold, the process is stopped. Another method discussed is the target method. In this method, relevant documents are identified using an independent information retrieval method, and a target set is created by sampling these relevant documents. CAL runs until all the documents identified in the random sample of relevant documents are found.

---

**Algorithm 1** Scalable Continuous Active Learning (reproduced) [13]

---

- 1: Find a relevant seed document using ad-hoc search, or construct a synthetic relevant document from the topic description.
  - 2: The initial training set consists of the seed document identified in step 1, labeled “relevant.”
  - 3: Draw a large uniform random sample  $U$  of size  $N$  from the document population.
  - 4: Select a sub-sample size  $n$ .
  - 5: Set the initial batch size  $B$  to 1.
  - 6: Set  $\hat{R}$  to 0.
  - 7: Temporarily augment the training set by adding 100 random documents from the  $U$ , temporarily labeled “not relevant.”
  - 8: Construct a classifier from the training set.
  - 9: Remove the random documents added in step 7.
  - 10: Select the highest-scoring  $B$  documents from  $U$ .
  - 11: If  $\hat{R} = 1$  or  $B \leq n$ , let  $b = B$ ; otherwise let  $b = n$ .
  - 12: Draw a random sub-sample of size  $b$  from the  $B$  documents.
  - 13: Review the sub-sample, labeling each as “relevant” or “not relevant.”
  - 14: Add the labeled sub-sample to the training set.
  - 15: Remove the  $B$  documents from  $U$ .
  - 16: Add  $\frac{r \cdot B}{b}$  to  $\hat{R}$ , where  $r$  is the number of relevant documents in the sub-sample.
  - 17: Increase  $B$  by  $\lceil \frac{B}{10} \rceil$ .
  - 18: Repeat steps 7 through 16 until  $U$  is exhausted.
  - 19: Train the final classifier on all labeled documents.
- 

### 2.4.3 Scalable Continuous Active Learning

One drawback with CAL is that it requires annotations for all documents. Scalable Continuous Active Learning (S-CAL) is an attempt to remedy this drawbacks. It differs from CAL in the use of sampling: only a random sample of the top-ranked training examples is selected for manual labeling. The model scores very similar (or duplicate) documents approximately equally. These training examples are unlikely to be diverse, and having the annotator label extremely similar documents is not the best use of time; more information can be gained by labeling a diverse set of positive examples. S-CAL helps reduce the human effort required to create an accurate ranking model. The S-CAL algorithm is reproduced in Algorithm 1 from the original paper [13]. Here, the trade-off is between accuracy and human effort. This trade-off is controlled using the parameter  $b$  (line 11).  $b$  training examples are randomly selected and shown to the annotator out of the top  $B$  (line 5) ranks



in every iteration.  $b$  is directly proportional to accuracy: low values can be used to form a decent classifier quickly, whereas by setting  $b$  higher than  $N$  causes it to act more like CAL.

## 2.5 Transfer Learning

Transfer learning is a machine learning task that deals with the use of models trained for a source task to perform prediction on a target task, with or without additional training. A popular use of transfer learning for text has been as feature extractors, for example, using pretrained distributed word embeddings (such as Word2Vec or GloVe word vectors) is a form of transfer learning. There are three major expected advantages of using transfer learning as shown in Figure 2.3.

- Higher start: A transferred model is likely to have better performance at the start;
- Higher slope: A transferred model is likely to train faster;
- Higher asymptote: A transferred model is likely to have a higher final performance.

However, transfer learning does not guarantee these advantages. The outcome of transfer learning can be categorized based on the performance of the baseline model compared to the transferred model. When the performance on the target task increases after transfer, it is referred to as positive transfer, whereas if the performance decreases, it is called negative transfer.

### Transfer Methodology

Typically, transfer learning involves three stages: source selection, model mapping, and training and prediction. In the source selection phase, the task is to find a source task that shares some similarity with the target task. Improper source selection can be a cause of negative transfer. Source selection can be done based on evidence, or by manual search. In evidence based selection, multiple source tasks are used to initiate transfer, and the task with the best performance after a few iterations is used as the source task. Alternatively, a source task can be selected manually using domain knowledge. For example, for sentiment analysis of tweets, word2vec models trained on Twitter corpora are likely to perform better than word2vec models trained on Wikipedia.

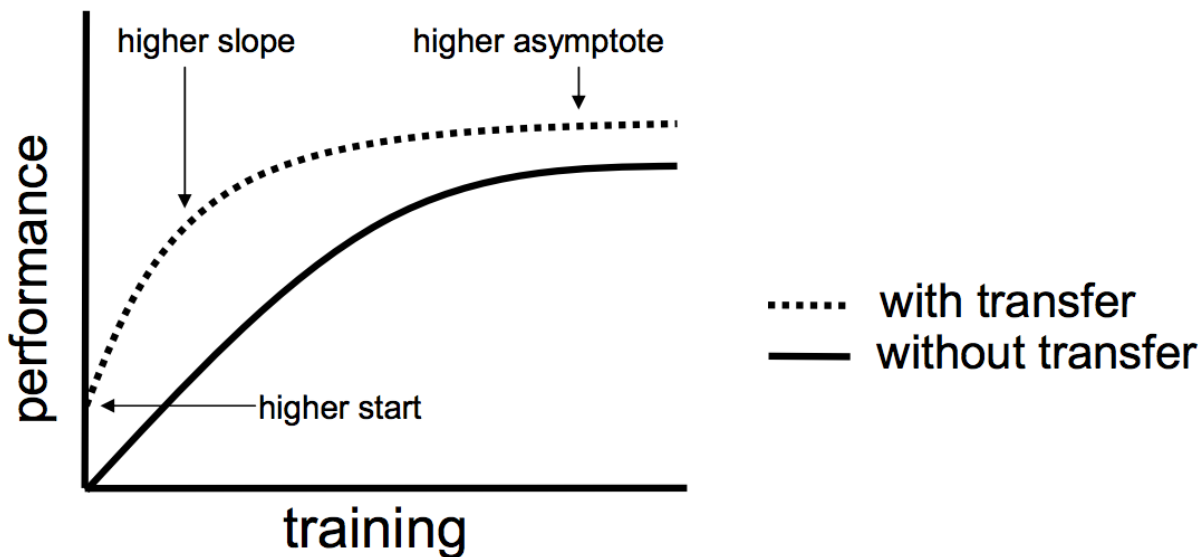


Figure 2.3: Expected Advantages of Transfer Learning. Image Taken from Transfer Learning [61]

Once a source model is selected, the next task is selecting a mapping from the feature set used in the source task to the feature set for the target task. One method is manually matching each feature in the source domain to a feature in the target domain. This matching may not be one-to-one or complete. Additional features in the source task can be simply discarded, whereas additional features present in the target dataset can be either discarded, or have randomly initialized parameters that are trained in the last phase. For tasks involving text, it is fairly straightforward for words that are present in the vocabulary of the source task. Out-of-vocabulary words in the target data can be addressed using smoothing methods [6], or more simply, can just be ignored.

Once a source model is selected and the features are mapped, the rest of the learning process can continue as just any other machine learning process. For an optimization algorithm, the transferred weight parameters can be considered the same as randomly initialized parameters, but are likely to be closer to the optimal solution. Training could also be performed to just modify weights initialized randomly for features only present in the target data. Such approaches are quite popular in transfer learning for deep learning [69]. Testing for negative transfer is usually done in this phase, and if detected, the whole process needs to be repeated.

## 2.6 Evaluation Metrics

Considerable research has gone into designing metrics for evaluating information retrieval methods. Precision and recall are two most widely reported metrics. Recall is the proportion of relevant documents that have been retrieved, while precision is the proportion of the documents that have been retrieved that are relevant. Let  $R$  be the set of relevant documents and  $U$  be the set of documents retrieved. Let  $\cap$  represent set intersection.

$$Precision = \frac{R \cap U}{U}$$

$$Recall = \frac{R \cap U}{R}$$

The  $F_1$  score is a harmonic mean of these two metrics. Let  $P$  and  $R$  represent the precision and recall respectively. Then,

$$F_1 = \frac{2 \times P \times R}{P + R}$$

### Precision@K and R-Precision

There are a couple of drawbacks with recall and precision. Calculating these metrics require prior knowledge of the set of all relevant documents. It also does not provide any information about the quality of the ranked list. Precision at  $K$  ( $P@K$ ) is a metric that can be calculated with relevance judgments available for only the top  $K$  ranked documents. Plots of  $P@k$  for all  $k \leq K$  capture the quality of the ranked list as well.

We calculate

$$P@K = \frac{|d \in Rel : rank(d) \leq K|}{K}$$

where  $Rel$  is the set of documents marked relevant.

R-Precision is the precision of the ranked list at rank  $R$ . If  $r$  is the number of relevant documents in the top  $R$  ranks, R-Precision [18] can be calculated as  $\frac{r}{R}$ . One interesting property of R-Precision is that it is equal to the recall of the ranked list, where rank  $R$  is considered to be a threshold for membership in the relevant class. Thus, at a threshold of  $R$ , the precision is equal to the recall, which is also equal to the  $F_1$  score.

## Measuring Effort

Gain curves, introduced by the authors of [54] aim to measure the annotation effort required to achieve a certain level of recall. In these curves, recall is plotted on the y-axis, and the effort in terms of number of documents annotated is plotted on the x-axis. Good performance is indicated by a steep increase in recall up to a point (80-90% recall) followed by plateau in the rate of increase of recall, till it finally reaches 100%. The knee method of stopping described in Section 2.4.2 is inspired by this expected shape of the curve. Once the rate of increase of recall drops significantly, that is an indicator that not many more relevant documents are likely to be found.

One issue with considering effort to be the number of documents annotated is that  $R$  (the number of relevant documents) can vary a lot. Thus, for a large  $R$ , the slope of the line can be low, even with precision close to 100%. To compare multiple gain curves using one single plot, we need to normalize these curves. We do this by plotting effort at multiples of  $R$  (e.g.  $R/2$ ,  $R$ ,  $3R/2$ , etc). This ensures comparability between the curves from two topics with high variation in  $R$

# Chapter 3

## Proposal and Study Design

The work presented in this thesis focuses on studying whether S-CAL can be used to train models to rank mobile reviews describing an app issue. We propose a framework to help app developers discover information about an issue their app is facing while at the same time, train a model to rank or classify reviews about the issue in the future. The end goal is to show developers reviews relevant to the issue they are interested in. Hence, we investigate methods to reduce developer effort required to do so.

In Section 1.2, we described three challenges developers face trying to extract information from reviews. We hypothesize S-CAL can be used efficiently to address these challenges. S-CAL can be used to train models for ranking reviews. This process is fast and accurate, and thus helps reduce the amount of developer time spent. While developers train these models, they also get exposed to a considerable number of relevant reviews. Thus, they discover information about the issue they are interested in, thereby increasing productivity.

The models trained by S-CAL focus on one issue faced by a particular app. However, these issues are not unique to one app and are often shared by multiple apps which provide similar features and functionality. For example, we look at two apps, ‘Trulia Rentals’ and ‘Auto Trader’. The former is an app where users can search for house rentals, whereas the latter focuses on sale of cars. Both these apps provide a search functionality, where users can filter results based on location, price, etc. Both apps face complaints and feature requests about these filters. User requests are to increase the number of available filters, or they complain about the performance of the existing ones. We investigate the reuse of a model trained on one app across multiple apps that face the same issue, directly or with additional training on the target app reviews.

## 3.1 Experiments

We designed two major experiments to investigate the two major aspects of our proposal. Our first experiment aimed to evaluate S-CAL as a tool for training models. Our second experiment aimed to evaluate the re-usability of models built on one app to rank reviews for another app, and compared the performance against the state of the art in mobile app review analysis.

### 3.1.1 User Experiment

In our first experiment, we sought to evaluate S-CAL for two tasks: training ranking models, and information discovery. To achieve these goals, we performed a controlled user experiment comparing the performance of ad-hoc keyword search and S-CAL. This experiment was the first time S-CAL has been evaluated through a user study. In our user experiment, we aimed to show that S-CAL could identify more unique relevant reviews than ad-hoc keyword search, and thus is a better tool for discovery than the latter. Precision of S-CAL was also measured. High precision indicates that the model being built is accurate. Additionally, it implies that the amount of developer time devoted to filtering out non-relevant reviews is reduced. The accuracy of the models built was measured via the transfer learning experiment.

### 3.1.2 Transfer Experiment

While the user experiment was designed to show that S-CAL is efficient at discovery and model building, the transfer experiment aimed to investigate the quality and re-usability of models built using S-CAL. The transfer experiment served two purposes. We evaluated the accuracy of the models built on a labeled test set of reviews from a target set of apps. We evaluated whether these models could be reused to rank reviews for a different app facing the same issue as the source app. We did the performance evaluation using the model directly to rank reviews and by using it as a base model which learned using a subset of the target set reviews. Finally, we compared the performance of the pre-trained models with the performance of two state of the art review analysis tools, ALPACA [66] and CLAP [63, 56].

# Chapter 4

## User Experiment

We evaluated S-CAL through a controlled user study. Our user study was approved by the University of Waterloo Office of Research Ethics (ORE #22288). The main goal we set for our participants was simple: retrieve as many reviews relevant to the topic as possible. Our study followed a controlled experimental design. We measured S-CAL against keyword search implemented using a state of the art relevance model: BM25 [53]. We implemented a web app with two interfaces, one for S-CAL and one for keyword search. We measured the performance of S-CAL against keyword search for two main reasons:

- An app developer cannot be expected to have the advanced text analysis skills required to process and extract insights from a corpus of reviews. However, it is reasonable to assume that they will be able to create a text file containing all reviews and use `grep` to perform keyword searches to try to find reviews containing information about a particular issue. In our view, keyword search using BM25, supporting features like phrase search, keyword negation, and ‘must’ clauses reflects an improvement over ‘`grep`’.
- Part of our goal was to evaluate S-CAL in a user evaluation scenario. Interactive Search and Judging (ISJ) using ad hoc keyword search has been shown to be able to form test collections of adequate quality when compared with depth-k-pooling [17, 55]. Depth-k pooling requires ranked lists from multiple diverse systems, and thus is not feasible for our study. Hence, comparing the performance of S-CAL against ad-hoc search using ISJ was one of the best available options.

		<b>Interface</b>	
		<b>Keyword Search</b>	<b>S-CAL</b>
<b>Session</b>	<b>First</b>	Participant 1	Participant 2
	<b>Second</b>	Participant 3	Participant 4

Table 4.1: Assignment of Each App to Participants, Ensuring All Possible Permutations were Assigned Exactly Once

## 4.1 User Study Design

In the study, each participant was asked to participate in two sessions, one using the keyword search interface and one using the S-CAL interface. Each session lasted one hour but the participant was allowed to stop early if they felt that they had annotated sufficient reviews to satisfy the information need. Each participant was paid \$15 per session, for a total remuneration of \$30. To motivate the participants' performance, we added a minor deception to our remuneration model. The participants were told that they would receive \$20 for participation, with another \$10 bonus based on accuracy, but everyone was paid the full amount irrespective of performance.

### 4.1.1 Order of Assignment

The order in which the sessions were assigned was significant because allowing all participants to use the same interface first could bias the study. For example, if S-CAL was used first, participants might have had a better idea of what kind of queries to use for keyword search. Similarly, if an app was always assigned in the second session, performance of participants on this app could differ from another app which was always assigned in the



first session. There were a total of four possible permutations for assigning an app to a participant, as shown in Table 4.1. The labels on the left indicate assignment order (was this the first app that this user worked on, or the second). The labels on top indicate assignment of interface - S-CAL or keyword search. Thus, for 10 apps, there were a total of 40 permutations. Each app was assigned to four participants, and each participant was assigned to two apps (for the two different interfaces).

Since participants were assigned different apps for the two sessions, they could not transfer their knowledge about the app from one interface to the other. This design reduced potential bias introduced by assignment. This way, we collected four sets of relevance judgements for each app, with two sets generated using S-CAL, and the remainder coming from keyword search. Thus, the total number of sessions required was 40 (four for each app, for 10 apps). We recruited 20 participants, each performing two sessions - one session using S-CAL and the other, using keyword search.

### 4.1.2 Instructions for Participants

We explicitly instructed our study participants to try to retrieve as many reviews as they possibly could. A summary of the instructions is seen below:

1. Each participant is assigned a particular interface (S-CAL or Keyword Search) and the reviews from a particular app (based on the study design described in Section 4.1). The assigned interface is to be used to find/classify as many reviews as possible in one hour. The participant may stop early if they feel that they can not find more relevant reviews, or if they have found enough.
2. The interface that the participant did not use in the previous, session along with a different app, is assigned to the participant in the second session. Using the newly assigned interface, the participant is asked to find/classify as many reviews as possible for one hour. As before, they may decide to stop early if they believe they can not find more relevant reviews, or if they have found enough.

The notion of ‘enough reviews retrieved’ was not defined and left at the discretion of the participants. Additionally, the exact instructions provided to participants can be seen in Appendix A



Figure 4.1: Example of an User Review Containing a Star Rating, Title and Comment

### 4.1.3 Recruitment of Participants

We recruited participants via an android developer mailing list, the CS graduate student mailing lists at our university, and posters posted in the Computer Science and Engineering departments. We restricted our participants to those with software development experience, specifically, mobile app developers, or university students with software development experience as part of their research or internships. This was to ensure some familiarity with user-generated bug reports/feature requests. We posited that people belonging to this demographic were more likely to be able to answer the question “Is this review related to the issue mentioned in the topic description?”

## 4.2 Data Set

Our dataset was curated from a corpus of 51 million reviews of 10369 mobile apps in the Google Play Store, collected between March 2014 and March 2017 [42]. The top 500 apps for each category in the Play Store at the time were selected for crawling. An open source crawler was used to collect the data.<sup>1</sup> The crawler collected meta-data about apps (e.g., developer name/website, version code, date of release, etc), the reviews received and the .apk application package. Of interest to us were the reviews (figure 4.1). These reviews have three primary components: the title (blue text in figure 4.1), the review text (black text) and the star rating, which has a value between 0 and 5. The Play Store also allows the developer to respond to a review left by a user, but the prevalence of these responses was low. Nonetheless, in this thesis, the term ‘review’ is used to refer the combination of these four fields, or a subset containing the star rating, and at least one of the title and comment fields.

---

<sup>1</sup>Play with Google Play API :) <https://github.com/Akdeniz/google-play-crawler>

App Name	Topic	Topic Description
Walk with Map My Walk	Issues signing up	People complained about having issues while signing up. Find all such complaints that speak of issues during log in/registration
JustWink Greeting Cards	Integration with apps	App users complained about issues related to sharing cards on facebook. Find complainits related to signup/sharing/syncing with facebook
Voxer Walkie Talkie Messenger	No Notifications?	App users were concerned they were not being notified for some messages while others felt they got the same notification over and over again. Investigate this issue and find complaints that deal with notifications
Teen Vogue Me Girl	Costly in-app purchases	Users complained that some of the downloadable content cost too much(in game currency, or in-app purchases). Find as many related comments as you can
Xbox 360 SmartGlass	Can't Connect to XBox	Some people seem to have problems syncing and connecting to the xbox. Find Complaints that talk about connection/sync issues
SavingStar - Grocery Coupons	Not enough offers?	Users seem to feel that there are not a lot of offers, and often these offers don't work because they got maxed out. Find complaints that deal with unavailability/low number of offers
Smart AppLock (App Protect)	Fingerprint authentication	Users are requesting for fingerprint based authentication. Find all such feature requests relating to fingerprinting.
Amazing Spider-Man 2 Live WP	Not enough options in free version	A lot of users felt that the app provided very few options in the free version
iBeer FREE - Drink beer now!	Too many ads	People Kept complaining that the ap shows too many ads, even for a free app. Find complaints that relate to this
Trulia Rent Apartments & Homes	Filtering results	Customers complained about not being able to filter search results properly (for example, by price range, or location, due to lack of options/speed/accuracy/etc). Find complaints related to this

Table 4.2: Title and Descriptions of Topics Used in the User Experiment

### 4.2.1 User Study Topics

For the user study, we selected 10 apps with an average of 9988 reviews per app. The author created the candidate topics for each app by looking at low-rated reviews for the app on the Play Store website. A keyword query was then formed and the corpus was searched using `grep` for confirmation. Each topic had a title and a short description. This was to help the participant understand what information was required for a particular topic. Information about selected apps and the associated topic is summarized in Table 4.2.

## 4.3 System Description

We created a web app for our user study. It provided two interfaces, one for S-CAL, and one for keyword search. We built a backend server hosting services for keyword search and S-CAL, as well as a service to keep track of users responses.

### 4.3.1 S-CAL Interface

The interface for S-CAL was simple. We showed the participant the title, rating (between one to five stars), and comment text for a review, and provided two buttons to indicate their relevance judgement. We provided the participant a synthetic review constructed from the topic description (as per Algorithm 1, Step 1) to initialize the learner.

To further explain what a participant was expected to do, we imagine Bob, using the S-CAL interface to retrieve reviews for MapMyWalk (Figure 4.2). He starts off with a synthetic review, which is shown just above the review (e.g., query shown in figure 4.2: “can’t signup sign up”). He clicks on ‘start marking’, and reviews begin to appear. To navigate to the next review, Bob must mark the current review either as relevant, or non-relevant. He continues this process for an hour, or until he is satisfied he has retrieved enough reviews.

The preprocessing steps for S-CAL were as follows:

1. Tokenize words according to the regex: `[A-Za-z0-9]+`
2. Filter out tokens that appear only once
3. Stem tokens using Porter stemmer

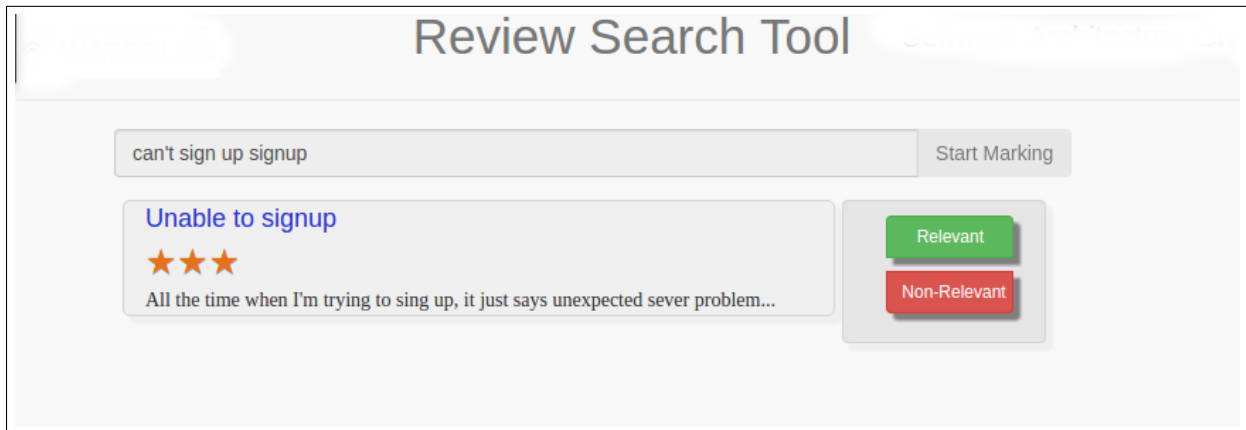


Figure 4.2: S-CAL Interface Showing a MapMyWalk Review

#### 4. Apply TF-IDF weighting according to Formula 2.1

On the backend, we used a package provided by Cormack and Grossman [13] wrapped around using a REST API. The Machine Learning algorithm used in the S-CAL package is logistic regression. The implementation available in Sofia-ml was used in the package.<sup>2</sup> The implementation of S-CAL in the package had a small number of differences from Algorithm 1.

1. The cycle was repeated for a maximum of 100 iterations, or till  $U$ , the uniform sample of documents was exhausted.
2. Normally, this would mean that at most  $b \times 100$  documents would get shown. However, in the implementation used, if the number of relevant reviews returned in a particular iteration is 0, then the next iteration is not sampled: all  $B$  highest scoring reviews are shown to the participant.

The regularization parameter lambda was set to .0001 and dimensionality was set to 1100000. These parameters were the default parameters in the S-CAL package provided. We set the  $b$  parameter to 5, i.e., at most five reviews were shown in each iteration for annotation as long as at least one review was marked relevant in the previous iteration (as mentioned in item 2 of the list above).

<sup>2</sup>Sofia-ML: a fast machine learning library for regression, classification and ranking, available at <https://code.google.com/archive/p/sofia-ml/>

### 4.3.2 Search Interface

Our keyword search interface was slightly more complicated and provided more options. The features are summarized below:

1. Users could input a query using a text-box, and results were displayed on the page as shown in Figure 4.3.
2. Users could choose to annotate or ignore a review. This is different from S-CAL, where participants had to respond to all reviews S-CAL selected for annotation. As in S-CAL, participants could mark a review relevant, or non-relevant.
3. We also provided the option of bulk-coding all reviews above a certain review. This is similar to thresholding, where the participant selects a value for score, above which all documents are considered relevant.
4. Keyword Search supported complex keyword queries. ‘+’ could be used to indicate that a word must be present, ‘-’ was used to indicate a word must not be present, and quotes were used to indicate exact-phase search. The plus and minus operators could be used in combination with phrase search.
5. Once the participant reached the end of the search results, they could load more reviews using a button.
6. The interface provided an option to hide reviews that were already annotated. This situation arose if a particular review appeared in the results of multiple queries. If a review marked before appeared in a new search, it remained highlighted as shown in Figure 4.3.

To illustrate a participant’s experience using the keyword search interface, we imagine Alice searching for notification issues regarding the JustWink Greetings Card app as shown in Figure 4.3. In the beginning, Alice sees only a text box to enter a query. Upon entering a query, 10 results are loaded onto our page. Alice now has the option of marking these relevant or non-relevant according to her judgement. However, providing judgements for each document is not compulsory. Alice could mark only relevant reviews, completely ignoring the non-relevant ones. Once Alice has marked a document, she can choose to hide it using the filtering options provided: she can filter out all documents she has already marked, only those she marked relevant, or only those she marked non-relevant. These options were provided to help Alice focus on the task at hand, without being distracted

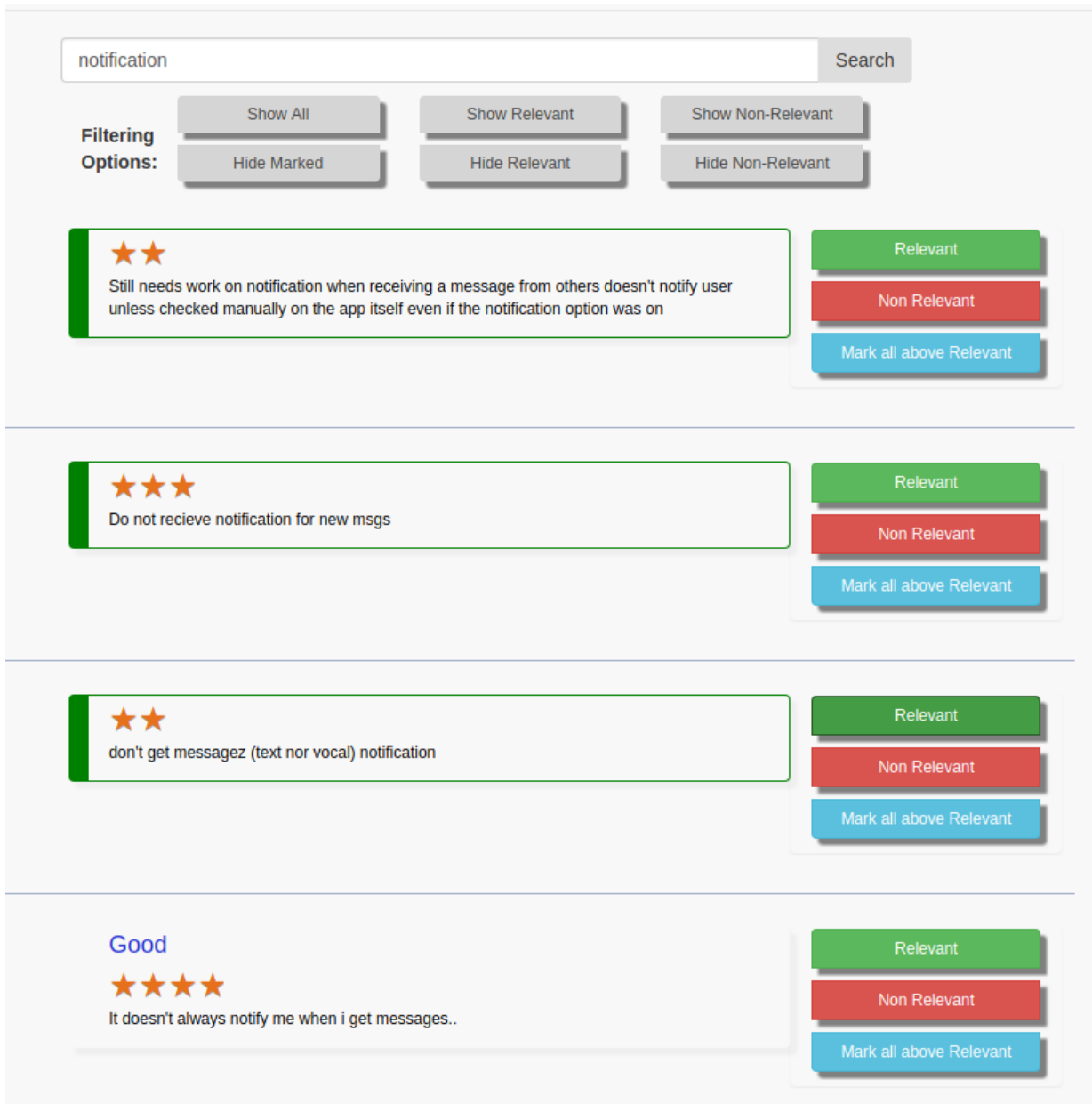


Figure 4.3: Interface for Keyword Search Showing Results for JustWink App

by documents she had previously marked. At any point, Alice could enter a completely new query and start a new search, or load more results once she has seen the top 10. She continues doing this for an hour, or until she is satisfied with her work.

On the backend, we indexed our dataset using ElasticSearch.<sup>3</sup> The preprocessing steps were as follows:

1. Reviews were tokenized using the standard Unicode Text Segmentation algorithm.
2. Tokens were stemmed using the Lovin's stemmer.
3. Standard English stopwords were removed and HTML characters were stripped.
4. Given a query  $q$ , we scored each document using BM25 similarity (formula 2.2)
5. Each part of the review (Title, Comment, and Reply Text) was analysed and given equal weights.

---

<sup>3</sup>ElasticSearch: <https://www.elastic.co/>



# Chapter 5

## Transfer Experiment

We used transfer learning to simulate scenarios where app developers reuse models trained for different (source) apps to rank reviews from their own (target) app. There are two scenarios possible. If the model trained on the source app can rank the reviews for the target with high precision and recall, then no further training is necessary, and the model can be used as is. However, if the developers are not satisfied with the output, they may decide to perform additional training on the target set. We simulated both scenarios in our transfer experiments. We also compared the performance of two state of the art tools that are used for review analysis against models trained using S-CAL.

### 5.1 Data Set

We performed the transfer learning experiments using a data set curated from the corpus of 51 million reviews described in Section 4.2. We loaded the entire corpus into ElasticSearch and performed searches using the candidate queries we used to create topics. We aggregated the top results by app to get a count of the reviews, and selected five target apps where estimated prevalence suggested class imbalance. These five target apps belonged to similar categories as the source apps. We elected to proceed with just five apps for the test set as we were unable to pair the remaining five apps with reasonable effort. Candidate queries for some topics were very generic. For example, for ‘JustWink Greeting Cards’ the reviews returned by the query did not complain about the topic associated with the app, and we were not able to reformulate this query to get a reasonably accurate estimate. At the other extreme was ‘Microsoft Smartglass’ which was the only app in our database which is capable of connecting to an X-Box, and thus the only one with issues syncing

Target App Name	Source App Name	Relevant Reviews	
		Test Set (R)	Train Set
AppLock	Smart AppLock	7	2
AutoTrader -Cars for Sale	Trulia Rentals	35	17
Stupid Zombies 2	iBeer Free	11	3
Gmail	Voxer Messenger	8	4
Earn Money	SavingStar	5	4

Table 5.1: Test Set Apps

with one. While some of these pairs of apps have different goals, they provide similar features. For example, ‘Trulia Rentals’ is used for buying, selling and renting homes, while ‘AutoTrader - Cars for Sale’ does the same for cars. However, they both provide functionality to search for a particular type of car or home, and options to filter the result lists returned. Similarly, ‘Voxer Messenger’ is an instant messaging platform whereas ‘Gmail’ is a mail client. However, both these apps provide push notifications whenever there is a new message or email, which is an issue related to both apps.

For the test set, we randomly sampled 400 reviews from each of the five selected apps and the author then labeled this dataset in random order. Thus, we had a total of 2000 reviews in the test set, of which 66 were relevant. For the experiments (see Sections 5.2.2 and 5.2.3) with additional training on target data, we created a training set by randomly sampling 200 reviews for each app from the set of reviews that remained unselected. This set was labeled by the author in random order. Out of the 1000 reviews in the training set, 30 reviews were marked relevant. Information about the data set is available in Table 5.1.

For the model mapping phase, we preprocessed the test set reviews following the method described in Section 4.3.1. To get the vector representations, we fit the reviews from the target app to the tf-idf model calculated on the source app reviews. All out-of-vocabulary words were ignored.

## 5.2 Experiments

### 5.2.1 Direct Transfer

In the direct transfer experiment, we took the models trained on the source app reviews, and directly used them to rank the target app reviews. This simulated the scenario when an app developer directly uses a model pre-trained for reviews of a different app to rank reviews for their app. In this scenario, the two apps provided similar functionality and were likely to face similar issues. For each target app, we had two models trained on the source app by two different participants during the user experiment. We ranked 400 reviews from each of the five target apps using both models. For evaluation, we measured r-precision of the ranked list, and also report recall at integer multiples of  $R$ .

### 5.2.2 Additional Training

Even if multiple apps receive reviews complaining about the same feature, models trained on reviews from one app might not necessarily perform well on those from another app. In such a situation, app developers would have two choices.

1. They could start training a model from scratch using S-CAL. This scenario was simulated in our User Experiment.
2. They could use an existing model (source model), and use labeled reviews from their app (i.e., the target app) to further tune the the model. This scenario corresponds to a transfer learning task with additional training on target app reviews.

We simulated the second scenario in this experiment. We performed additional training in two stages. For the first stage, we used only 100 of the 200 reviews in the training set. We repeated this experiment with the remaining 100. In the second stage, we used all 200 target set training reviews to perform additional training on the models. Here, we wanted to show that model accuracy can be improved by training our models on target set reviews. Increasing the amount of target data the source model trains on would increase performance. We expected to see the models trained with all 200 reviews to outperform the models trained with only 100.

We took the two models trained by participants in our experiment for each source app and we perform additional training. Then, we use the models trained to rank the

400 target app test set reviews. We report R-Precision, and plot curves showing recall at integer thresholds of  $R$ . The results from the first stage was conducted for both folds of 100 target set reviews, and the results averaged.

### 5.2.3 Additional Sampled Training

Using a random sample of reviews for additional training added a large number of non-relevant reviews to the dataset, but only a few relevant ones. One method CAL and S-CAL use to deal with the class imbalance is to under-sample the negative class. We simulated this scenario as well. This scenario corresponded to a developer providing a small sample of relevant reviews identified using an arbitrary method, along with a comparable number of randomly selected non-relevant reviews. The positive reviews could be identified by just looking at reviews with low ratings, whereas a small sample of reviews could be labeled to identify the negative dataset. Another approach developers can take to create the training set to generate synthetic reviews.

For this experiment, we added all reviews marked relevant by the author to the set used for additional training. We performed two sets of experiments. In the first one, we randomly selected  $2R_{train}$  non relevant reviews, where  $R_{train}$  was the number of relevant reviews present in the 200 reviews selected for additional training for each app. We then performed additional training with these  $3R_{train}$  reviews on the base models trained by participants and used it to rank the reviews in the test set. We then repeated the experiment with  $5R_{train}$  randomly sampled non-relevant reviews as opposed to  $2R_{train}$ . Finally, we report R-Precision and plot the values of recall at integer multiples of  $R$ . Additionally, since random selection was used to create the training sets, we repeated each of the two experiments 100 times and report the mean.

### 5.2.4 Comparison Study

We performed a study comparing the performance of the transferred models with state-of-the-art methods used for review analysis. When selecting tools for comparison, we focused on tools that either provided a ranked list of app reviews, or clustered reviews into specific topics. We chose to ignore studies that focused simply on classifying reviews into broad categories (for example, ARDOC [50]) since the problem we are investigating is discovery of specific issues.

We reached out to the authors of papers describing three tools: MARK [65], CLAP [63, 56], and AR-Miner [8]. These tools were not an exhaustive list of tools that satisfied our

criteria, but the authors of the respective papers have showed that the performance of their tools are state of the art, and at least as good as the other options. Among these, we did not hear back from the authors of AR-Miner, and the authors of MARK recommended using ALPACA [66], which is a tool built on top of their previous research, with more features than MARK. Thus, we perform our experiments using ALPACA and CLAP.

## MARK and ALPACA

MARK was built as a framework for semi-supervised analysis of user reviews in mobile app stores. Users can describe a topic they are interested in by providing a set of keywords. MARK uses word2vec [44] word embeddings to find keywords similar to the keywords entered by the user. The user can also use the set of keywords, original or expanded, to retrieve reviews related to the topic. ALPACA is an expansion of the capabilities of MARK. ALPACA uses linguistic pattern mining to identify meaningful patterns in reviews, and then uses these patterns to identify intentions. A pattern is defined as a sequence of functional words and POS-tag placeholders such that different words can be inserted to form different sentences which convey the same intent. Reviews are then matched with patterns and filtered the same way as MARK, using expanded keywords. The output returned by ALPACA is a list of reviews that match extracted patterns and filtered using supplied topic keywords as shown in Figure 5.2.4.

The authors of ALPACA provided a Java GUI application of their tool. We used their tool without modification for our experiments. The experiment set-up was quite simple. We transformed the test set reviews into the format used by ALPACA, and loaded it into their GUI tool. We ran pattern extraction, keyword expansion and opinion extraction, and ALPACA returned a list of reviews that matched the extracted patterns and contained the keywords present. The seed keywords used for extraction were the same as the candidate queries used to generate topics as described in Section 4.2. We extracted opinions using both original and expanded keyword sets. Since the output is a list of reviews, we could use ranked list evaluation metrics to evaluate the results. We report R-Precision and recall at thresholds that were integer multiples of  $R$ .

## CLAP: Crowd Listener for Release Planning

CLAP is a framework that detects and prioritizes app issues to help developers plan future app releases. It achieves this goal through three main steps. First, a classifier classifies the reviews into the categories ‘Bug Report’, ‘Feature Suggestion’ or ‘Non-Functional Requirement’. The last class is further subdivided into performance, energy security and

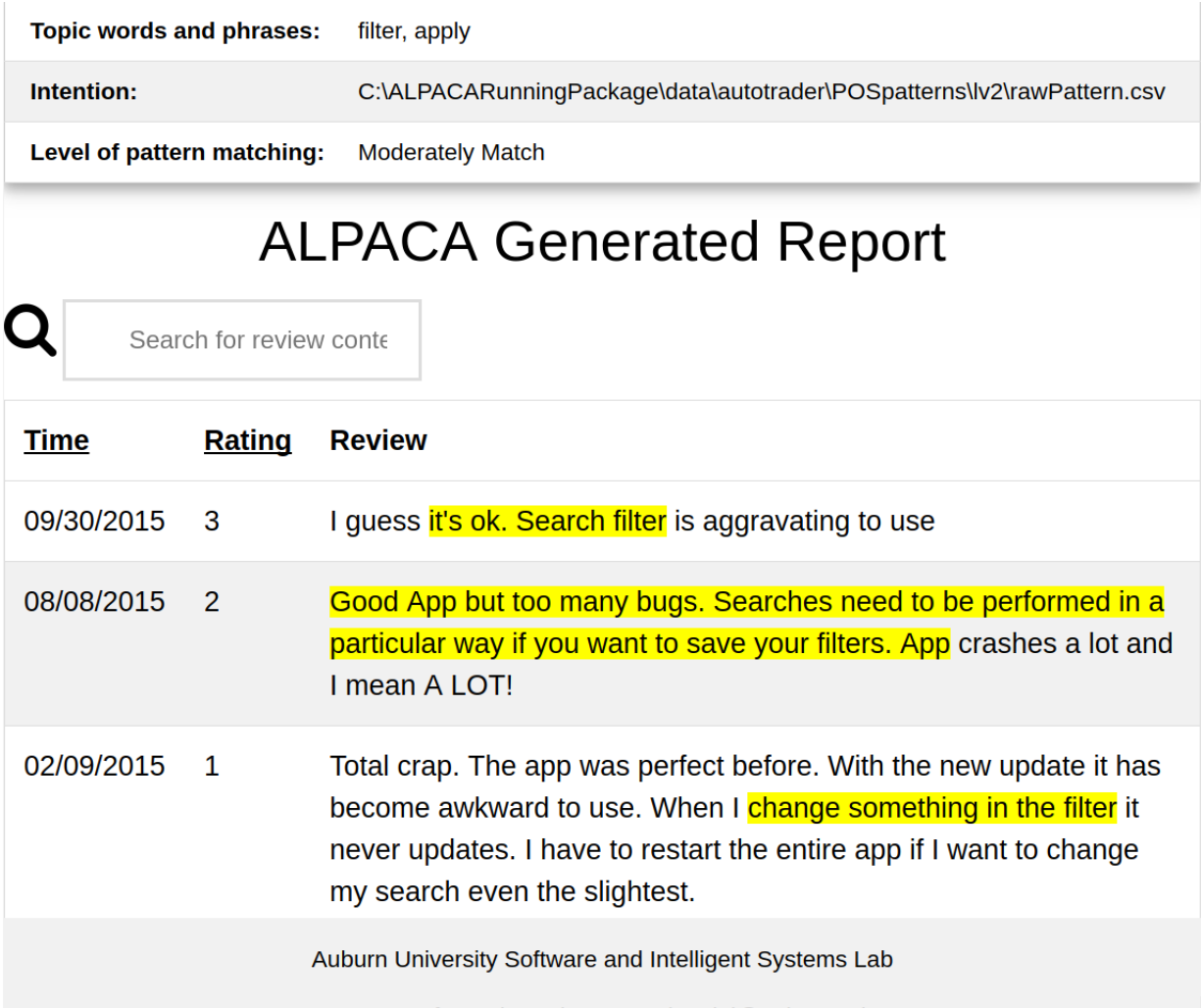


Figure 5.1: ALPACA Final Output for the Autotrader App, Showing the Expanded Keyword Set and Retrieved Reviews

usability issues. This classification is done using a Random Forest Classifier trained on a labeled dataset. The reviews classified as ‘Bug Reports’ or ‘Feature Suggestion’ are then clustered to form topics using the DBSCAN clustering algorithm. Finally, these clusters are prioritized using features generated using ratings of reviews in a cluster, the size and the variation in metadata of the reviews (e.g., the number of unique devices from which the reviews originated). In Figure 5.2.4, we show the output for the ‘Stupid Zombies 2’. The final output is a set of clusters, identified by keywords, containing one or more reviews that report the same issue (bug report or feature suggestion). All reviews that are not classified into any of the three main categories are present in the ‘other’ category.


The authors of CLAP provide a web-app for users who wish to use their framework. The process is completely automated, and once reviews are uploaded, their framework automatically categorizes and clusters reviews and prioritizes these clusters. However, since no ranked lists are returned, rank-based performance metrics could not be used to compare performance. We used a two-fold evaluation strategy to evaluate the results obtained from CLAP. If there was a cluster which accurately describes the topic we were interested in, then we reported precision and recall for that cluster. However, if there were no clusters representing our topic of interest, we considered the reviews reported as a bug report, feature suggestion or non-functional requirement as the positive class, and report precision and recall. Our intuition behind treating all these classes as the same positive class was simple: if developers were to discover the issue of interest using CLAP, they would have to have read through at least all the reviews classified by CLAP into one of the major categories. Thus, it seemed reasonable to report the precision and recall of this set.

Bug reporting




Suggestion for new feature

Non-functional requirement




Other

C4: adds add graphics great [3 reviews] 




... ★★★★★  
Add more levels needs still better graphics.  
— Device: Date: 05 lug 2014


Confirm category   
Change cluster   
Delete review 

... ★★★★★  
Nice game but many adds.  
— Device: Date: 28 feb 2017

Confirm category   
Change cluster   
Delete review 

Great game would give 5 stars but too many adds... ★★★★★  
.  
— Device: Date: 18 mar 2014

Confirm category   
Change cluster   
Delete review 

C3: ad ads start add [1 reviews] 


C2: want bullets fun love [1 reviews] 

Figure 5.2: CLAP Tool Interface, Showing Output for Stupid Zombies 2 app, Focused on Feature Requests



# Chapter 6

## Results

### 6.1 User Experiment

We recruited 20 participants for our study. During these sessions, participants took an average of 12 minutes and 19 seconds to mark 100 reviews (min 5:57, max 28:48) while using the S-CAL interface and provided 9740 judgments in total. For the User Evaluation, we simply assume that the user’s judgment is correct. This assumption simply meant we were evaluating S-CAL from the participant’s perspective; we measured how accurate the participant found S-CAL to be. We also measured the effectiveness of S-CAL at the task of discovering unique relevant reviews, and we compared this performance against Keyword Search. We also report inter-participant agreement.

#### 6.1.1 S-CAL Precision

The goal of this evaluation is to measure the participant’s impression of accuracy. The rationale was that the developers could use S-CAL to find relevant reviews to label. This could be to create a model or to manually peruse these reviews and identify more information about the issue. We wanted S-CAL to show as many reviews to the participant as possible and the vast majority of these reviews to be relevant (i.e., the ranked list should have high precision at low ranks).

Precision at  $K$  is the fraction of reviews viewed ( $K$ ) that were judged relevant. Since it was compulsory for participants to mark every review shown to them by S-CAL, we can

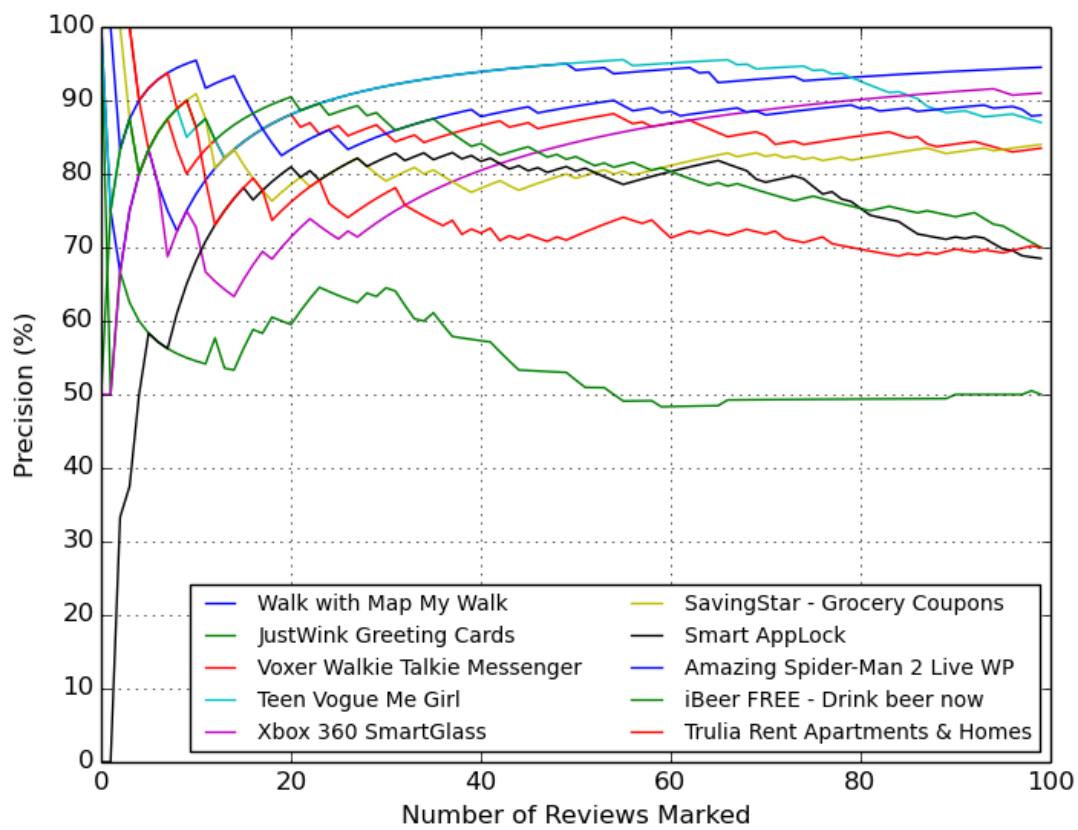


Figure 6.1: Trends for  $P@K$  for  $1 \leq K \leq 100$

calculate Precision at K by using the formula below.

$$P@K = \frac{|r \in Rel : rank(r) \leq K|}{K}$$

where *Rel* is the set of reviews marked relevant, and the ranking is the order in which reviews were shown to the participant. Since we had two sets of relevance judgments for each app, we averaged the Precision at K for each K between 1 and 100. In Figure 6.1, we show the trend of Precision at K for the 10 apps from 0 to 100. The average precision after 50 results is 81.69%, and after 100 results it drops to 78.65%. We can see in general that precision after 100 results varies between 70% and 95% with one outlier, ‘JustWink Greeting Cards’, on which precision was only 50% after rank 100.

Further investigation showed that precision on Keyword Search for ‘JustWink’ was extremely low as well, with only 38 reviews being marked relevant out of 760 viewed by one participant, whereas the other participant viewed a total of 3150 reviews, again marking only 38 of them relevant. It is possible that the difficulty for this topic was higher than for the others. Nevertheless, precision for S-CAL was high for all other apps.

We chose not to perform precision analysis in a similar way for Keyword Search since participants had a learning curve while figuring out which queries to use for a topic, and thus, it did not seem to be a fair comparison. We highlight this with one example from a participant on for the app ‘Xbox 360 SmartGlass’ searching for ‘synchronization issues with an X-Box’. This participant started off with combinations of the words ‘xbox’ and ‘sync’. The first 10 reviews were all negative. After going through 130 reviews with little or no success, he changed his query to ‘connect’ and got a precision of 96% for his first 50 results. With the correct query, the participant attained a high precision, but it took him quite a bit of time to reach the correct query. This can cause very low precision at the start. This is expected behavior for search and might be one reason pointing to the superiority of S-CAL in terms of efficiency, usability, and learnability.<sup>1</sup>

## 6.1.2 Unique Documents

We wanted users to be able to discover the most information possible in a fixed period of time, and seeing more relevant reviews was one indication of this. We evaluated the performance of S-CAL against keyword search. We compare the number of unique relevant reviews retrieved by each method. Here, uniqueness was defined as being retrieved by one

---

<sup>1</sup>The property of software that allows users to learn how to use it: <https://en.wikipedia.org/wiki/Learnability>

method only: if a review was found using S-CAL but not using Keyword Search, it counted as a unique review found using S-CAL. Since instructions given to participants were the same, if participants using S-CAL found more unique relevant reviews, this would be indicative of the superiority of S-CAL.

We considered the set of all reviews judged for a particular app. There were three categories: reviews that were only retrieved using S-CAL, documents that were only retrieved using keyword search, and reviews retrieved by both. Let set  $S_{scal}$  denote reviews found by S-CAL,  $S_{search}$  denote those found by keyword search and ‘\’ be the set difference operation. Then, we can find the quantities above according to the following formulae:

$$S_1 = S_{scal} \setminus S_{search}$$

$$S_2 = S_{search} \setminus S_{scal}$$

$$S_3 = S_{search} \cap S_{scal}$$

We considered a review to be in  $S_{s-cal}$  if at least one participant judged it to be relevant. We constructed  $S_{search}$  in a similar manner. In cases where the two users disagreed (one marked a review relevant, while the other marked it non-relevant), we considered it to be relevant. To test for significance, we conducted the Wilcoxon Rank Sum Test <sup>2</sup> on the cardinalities of each set.

In Figure 6.2, we report the cardinalities of these three sets as a box plot. From the figure, one can deduce that S-CAL generally found more relevant reviews than keyword search. The average value of  $|S_1|$  was 220.7 (min 12, max 898, median 184) whereas for keyword search, the average was 75.7 (min 8, max 241, median 112.5). This difference is statistically significant; the difference between the two distributions was significant with a P-Value of 0.00694 and a Z value of -2.711 on the Wilcoxon test.

The outlier in this result was Microsoft SmartGlass. One participant marked 863 documents relevant for S-CAL. The most relevant documents found using Keyword Search was also found for X-Box SmartGlass. Thus, it would be reasonable to surmise that this was a relatively easy topic, with easy indicator words such as ‘connect’ as mentioned in the previous section. Again, the hardest topic was the complaint for JustWink, which had  $S_1 = 231$ , and  $S_2 = 26$ . This demonstrates that even for hard topics, S-CAL outperformed keyword search.

Additionally, we found that participants were able to learn and use the advanced query syntax provided by our interface. We noticed that the median number of distinct words

---

<sup>2</sup>Wilcoxon Rank Sum Test <http://www.socscistatistics.com/tests/signedranks/Default2.aspx>

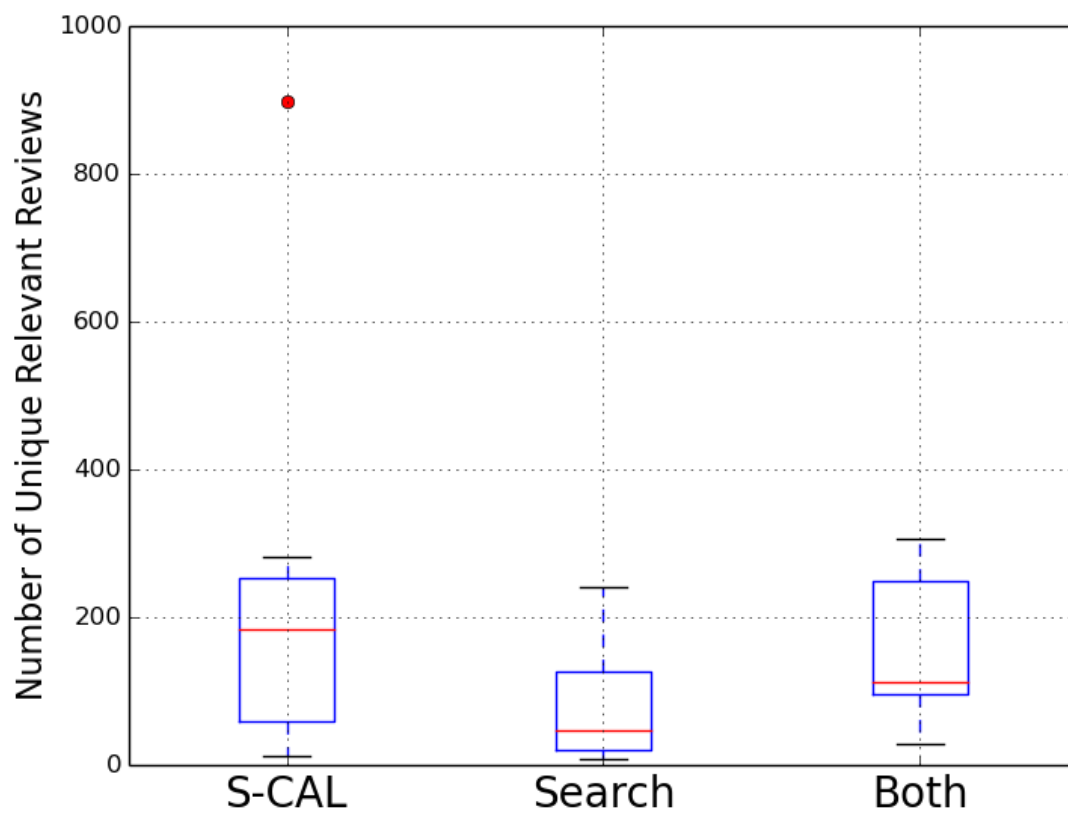


Figure 6.2: Box Plot Showing the Number of Unique reviews Found by S-CAL( $S_1$ ), Keyword Search( $S_2$ ), and Found by Both( $S_3$ )

used in queries by a participant was 7 (min 1, max 91). The participants were able to use advanced search features as well with a median of 10 (min 1, max 115) distinct queries performed by the participants using the search interface. However, keyword search had a learning curve and took some getting used to. Even if users of keyword search were experts at using the search interface, it is likely that they would still not know which query would yield high precision. However, when using S-CAL, only one relevant review or a simple search query is required to initialize the learner. The learner learns about additional words and terms to apply from the responses of the user.

### 6.1.3 Inter-Participant Agreement

High inter-participant agreement has been considered in the past to be an indicator of the quality of judgements [2, 58]. While there will be differences in opinion for a small fraction of reviews, participants should have agreed about a vast majority of the reviews that they annotate. We calculated inter-participant agreement as described in the TREC 2006 Legal Track overview [2]. We calculated Cohen’s Kappa according to the following formulae:

$$Kappa = \frac{p_o - p_e}{1 - p_e}$$

where

$$p_o = \frac{n_{00} + n_{11}}{n}$$

,

$$p_e = \frac{(n_{00} + n_{01})(n_{00} + n_{10})}{n^2} + \frac{(n_{11} + n_{01})(n_{11} + n_{10})}{n^2}$$

$n_{00}$  is the number of reviews participants agree are non-relevant,  $n_{01}$ ,  $n_{10}$  are cases of disagreement,  $n_{11}$  are reviews where both participants agree are relevant, and  $n$  is the sum of these quantities. Mean Kappa for S-CAL was 0.46, which is within limits of values expected for NIST assessors reviewing document collections for the Text Retrieval Conferences [58, 2]. In Figure 6.3, we show a box plot of  $p_o$  for various combinations, and these values are within ranges seen in previous TREC tasks. This can be seen as possible evidence of accuracy: If people agreed on their judgments more often than not, these judgments should have a high probability of being the correct judgment.

Qualitatively, we can also argue that the notion of true relevance is not important to us for the user evaluation, i.e., if a participant labeled a review is relevant, then they believed that the review contained some information related to the issue described in the topic. We

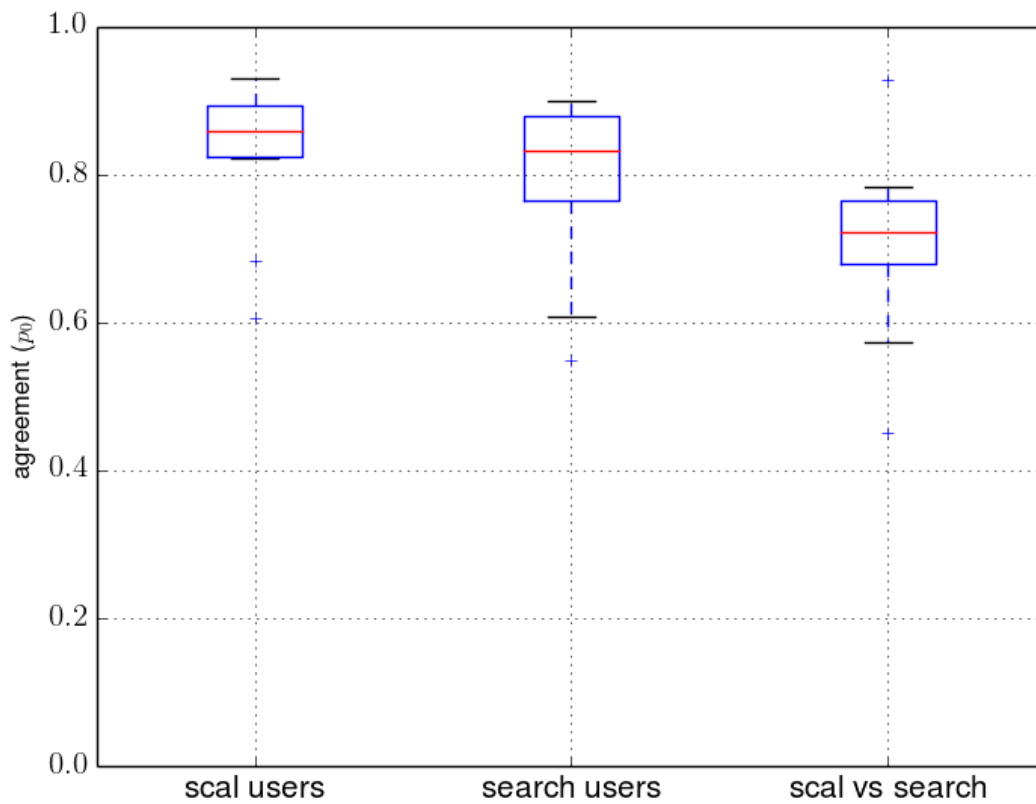


Figure 6.3: Box Plot Showing Agreement Between Participants Using S-CAL, Keyword Search, and Between S-CAL and Keyword Search Interfaces

provided clear instruction stating that a review should be marked relevant if and only if the participant felt that the review contained information about the issue. Thus, if the participant was to be a developer of that app, they would have obtained some information by seeing this review. Disagreements would arise due to the variation in strictness between participants’ notions of relevance. An example can be seen in this review for ‘Trulia Rent Apartments & Homes’: one participant using S-CAL marked the following review relevant, while the other marked it non-relevant.

”Hate that you can’t change price to your liking

Also I hate that it spams you with other building out of your price range.”

A strict reviewer could mark this review as irrelevant as there is no mention of the word ‘filter’, but one could also argue that this issue could be fixed if there were better filtering options and hence should be marked as relevant. This decision could only be made by the actual developer of the app.

## 6.2 Transfer Experiment

### 6.2.1 Direct Transfer

We evaluate the test set reviews ranked using the models trained using S-CAL using R-Precision. Table 6.1 shows the the values of R-Precision for each app for two different models trained by two different participants during the user experiment. The highest R-Precision reported was for the app Stupid Zombies ( $R = 11$ ). 10 out of the 11 relevant reviews were ranked in the top 11 by both models. AutoTrader, the target app with the highest number of relevant reviews in the test set ( $R = 35$ ), has moderately high recall at  $R$ , with the two different retrieving 28 and 26 reviews respectively. Macro-average of R-Precision is 71.82%. This value for is similar to the values observed for manual and automatic runs at the TREC 2016 Total Recall Track [28], and competes with BMI, which is an implementation of CAL with full human intervention [28].

In Figure 6.4 we look at the recall past  $R$ . The value on the y-axis is the average value of recall from the two models. The macro-average for recall across all apps at  $2R$  is 76.92%, and rises to 77.5% at  $3R$ . The low scores are easily explained by looking at the results for the ‘Earn Money’, which appears to be an outlier with a constant value of recall (40%). However, if we look the other apps, the macro-averages for recall at  $2R$  and  $3R$  are 96.16% and 96.87% respectively, which are in the same range as reported in the TREC Total Recall tracks.



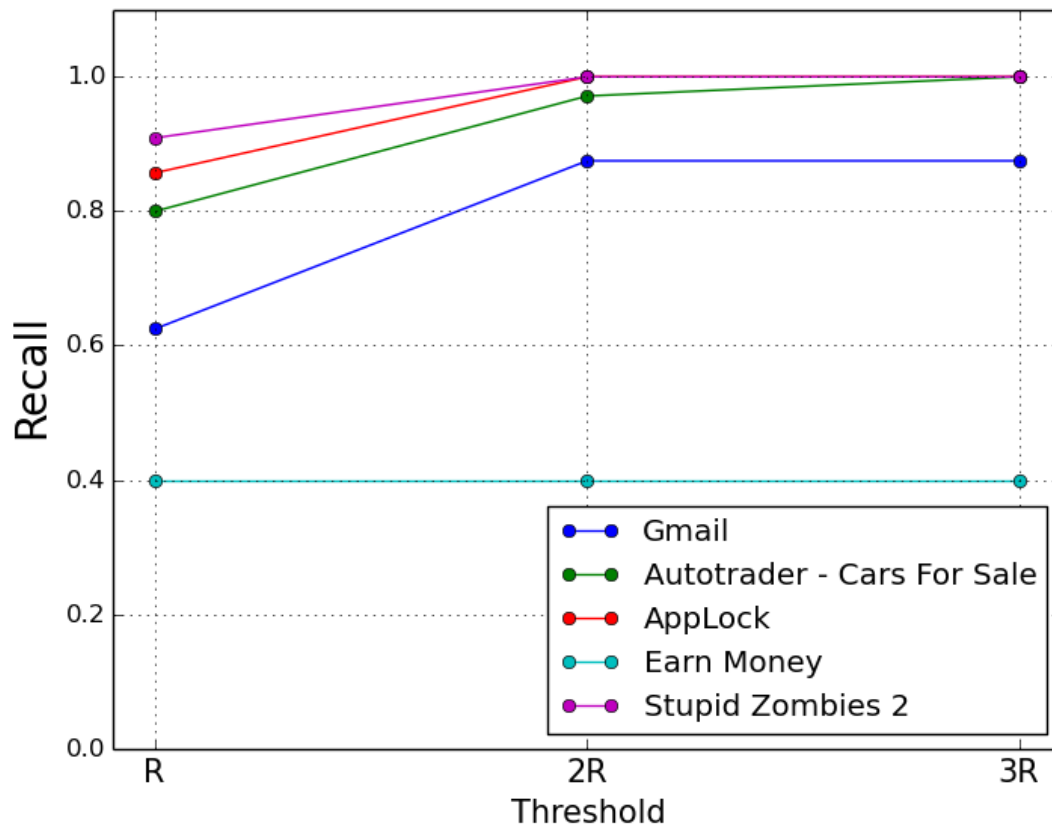


Figure 6.4: Plot Showing Recall at  $R$ ,  $2R$ , and  $3R$  of Models Trained Using S-CAL Used Directly for Ranking on Test Set

App Name	Model 1	Model 2
Autotrader - Cars For Sale	80.00	74.29
AppLock	85.71	85.71
Stupid Zombies 2	<b>90.91</b>	<b>90.91</b>
Gmail	62.50	62.50
Earn Money	<b>40.00</b>	<b>40.00</b>

Table 6.1: R-Precision of Models Trained Using S-CAL Used Directly for Ranking on Test Set

## Discussion

Though the topic was same, there were significant differences in the reviews marked relevant by the first author for the app ‘Earn Money’ and those marked relevant by participants for the paired ‘SavingStar App’. Reviews from the SavingStar app were similar to the example shown below; they were generally long, positive, and pointed out that they just wanted more options.

*It’s got the right idea.*

I wish the offers were more extensive. But none the less, the app has the right idea and is on the right track. Just keep adding offers! And I have had zero issues with app failure thus far!! Thank you for that!

However, for Earn Money, the reviews were shorter, often less than five words. Both models ranked the following review from Earn Money as 1:

Wish it had more options Still getting used to it

However, the other reviews were all short, had no clear sentiment and the vocabulary match was not strong as shown below.

No offers by now..why? Nice app

This mismatch can possibly explain the low value of recall for Earn Money. This could be due to suboptimal source model selection. A developer faced with such a scenario has different options. He could perform additional training using the base model. If the performance still does not improve, he can conclude that model selection was flawed and go back to the selection of a source model. Alternatively, he could choose to start fresh or restart from model selection at any point during his analysis.

## 6.2.2 Additional Training

There were two sets of performance evaluations performed for the additional training experiment. The first set is the evaluation of the models trained with 100 additional reviews and then averaged. The second set is the evaluation of the ranked lists returned by the models which were trained with 200 additional reviews from the target app. Since we had two models trained by participants during our user experiment for each source app, we have two sets of results for each. R-Precision of these models is presented in Table 6.2. Figures 6.5 and 6.6 show the values of recall at thresholds of integer multiples of  $R$ . The plots show the recall for each app; we aggregate the values by grouping them together using the target app, and plot the mean recall for each app. The macro-averaged R-Precision is 61.00% after training on 100 additional reviews, whereas after training on 200 additional reviews, it increases to 65.57%. There is a noticeable improvement in performance after training on more reviews at higher thresholds as well. This is shown by the plots in Figures 6.5 and 6.6. The values of recall for models trained on 200 additional target set reviews are higher than or equal to those trained on only 100 extra target app reviews. We also notice an improvement for the ‘Earn Money’ app, which reaches a maximum recall of 80% at a rank threshold of only  $2R$  after training with the additional 200 reviews from the target app.

## Discussion

The performance on the test set with additional training is lower than the performance of models without additional training for at least two of the five apps. ‘Gmail’ definitely sees negative transfer after additional training, whereas ‘Stupid Zombies 2’ has slightly lower precision up to a threshold of  $2R$ . A drop in performance is also seen for the ‘AutoTrader’ app, though after training on all the 200 reviews, it sees an improvement on the performance of the directly transferred models. However, in general, there is an increase in performance with increasing size of the training set of target app reviews.

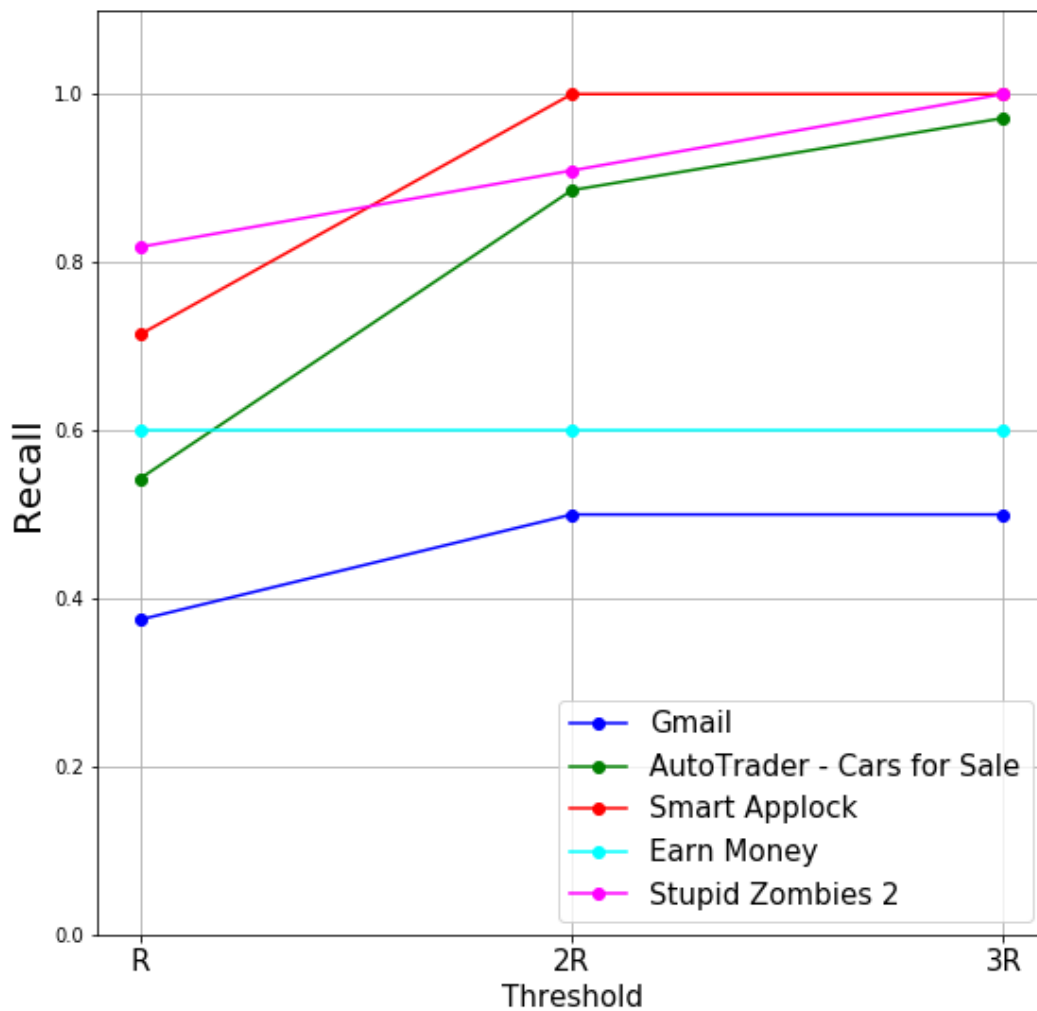


Figure 6.5: Plot Showing Recall at  $R$ ,  $2R$ , and  $3R$  of Models Trained Using S-CAL with Additional Training on 100 Target App Reviews on Test Set

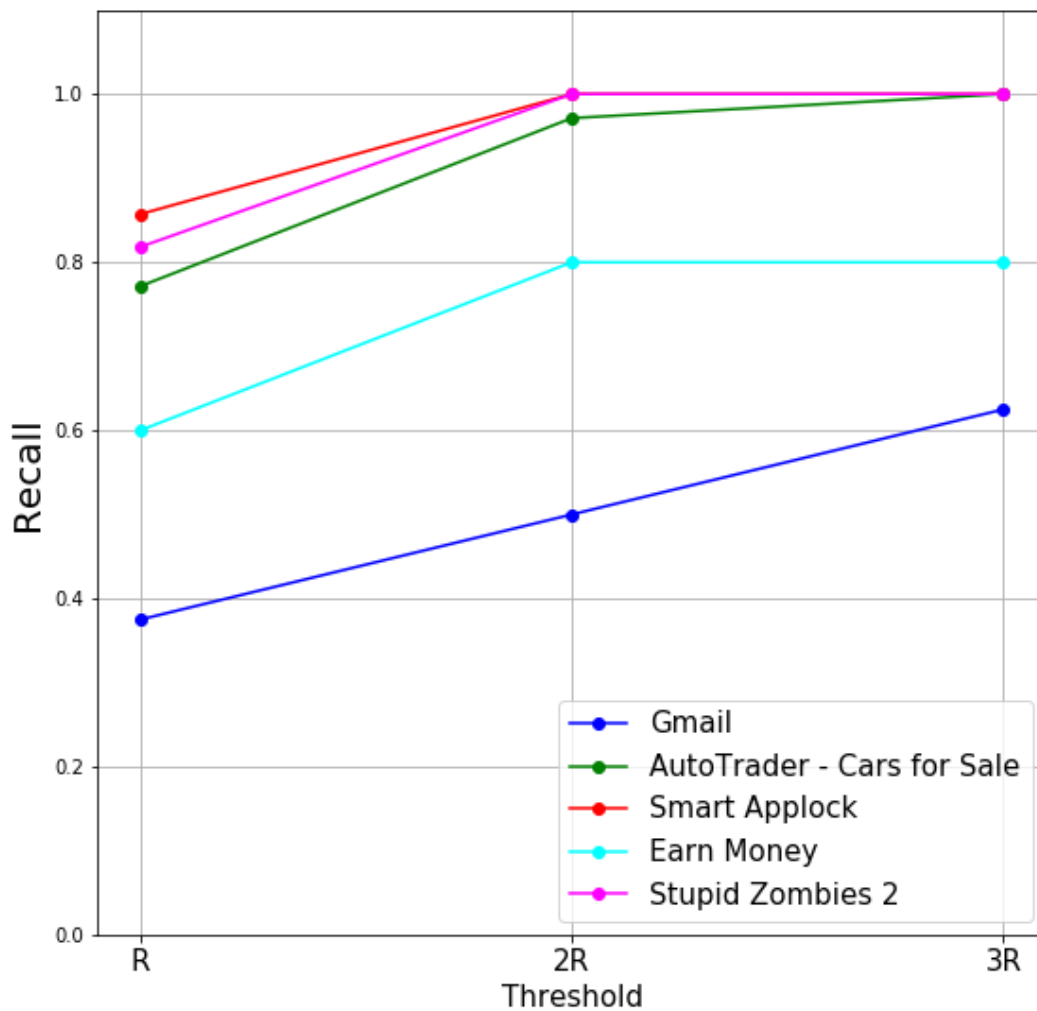


Figure 6.6: Plot Showing Recall at  $R$ ,  $2R$ , and  $3R$  of Models Trained Using S-CAL with Additional Training on 200 Target App Reviews on Test Set

App Name	Train Set: 100		Train Set: 200	
	Model 1	Model 2	Model 1	Model 2
Autotrader - Cars For Sale	54.28	54.28	77.14	77.14
AppLock	71.42	71.42	85.71	85.71
Stupid Zombies 2	81.82	81.82	81.82	81.82
Gmail	37.50	37.50	37.50	37.50
Earn Money	60.00	60.00	60.00	60.00

Table 6.2: R-Precision of Models with Additional Training Using Target App Reviews on Test Set

### 6.2.3 Additional Sampled Training

Table 6.3 shows the mean R-Precision of the experiment repeated 100 times. The macro-averaged R-Precision is 59.66% with  $2R$  non-relevant training examples and 61.41% with  $5R$ . In Figure 6.7, we plot the mean recall at  $R$ ,  $2R$  and  $3R$  with additional training using all relevant training examples and  $2R$  non-relevant ones. As with our previous experiment, there is negative transfer for Gmail, and Stupid Zombies 2 at lower thresholds. While ‘Stupid Zombies 2’ almost reaches 100% recall at  $3R$ , ‘Applock’ show losses in performance throughout the ranked list. Conversely, there is a consistent improvement in performance on reviews from ‘Autotrader’ and ‘Earn Money’ throughout the ranked list. Figure 6.8 shows the results obtained for the experiment with  $5R$  randomly selected non-relevant examples in the training set. While there is slight improvement in performance observed for all apps, there is a significant increase in the performance of ‘Applock’, which approaches 100% recall on average at a threshold of  $3R$ . Mysteriously however, recall for Gmail still remains much lower than what we observed in the direct transfer experiment.

### Discussion

For both experiments on additional training, we notice a large variation in values of recall at all thresholds. The most likely reason is that performance depends on the set of negative examples provided for training. However, adding more training data had a positive impact on recall at all thresholds. This is consistent with the observation made in the previous experiment as well, where the recall was higher when training with 200 additional reviews,

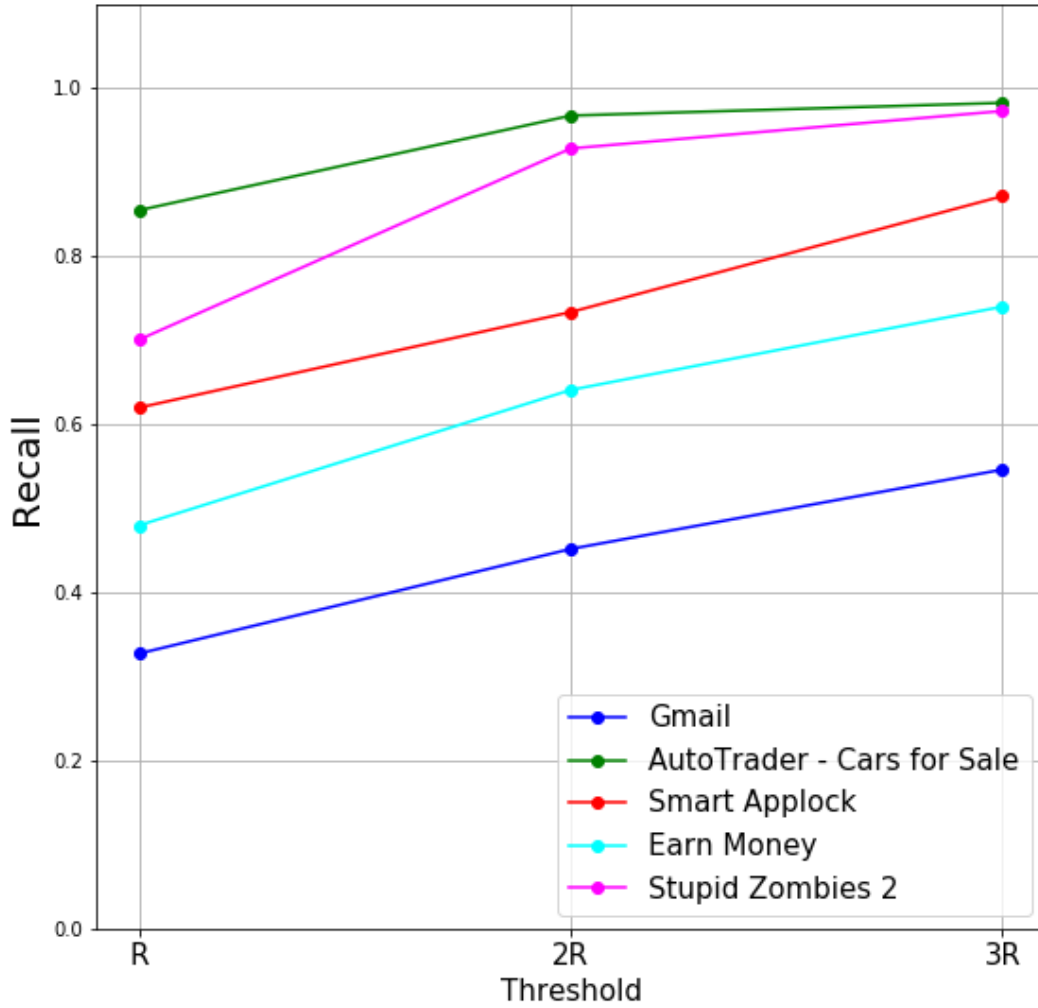


Figure 6.7: Plot Showing Recall at Thresholds of  $R$ ,  $2R$ , and  $3R$  on Test Set with Additional Training on  $R$  Relevant and  $2R$  Non-Relevant Target App Reviews

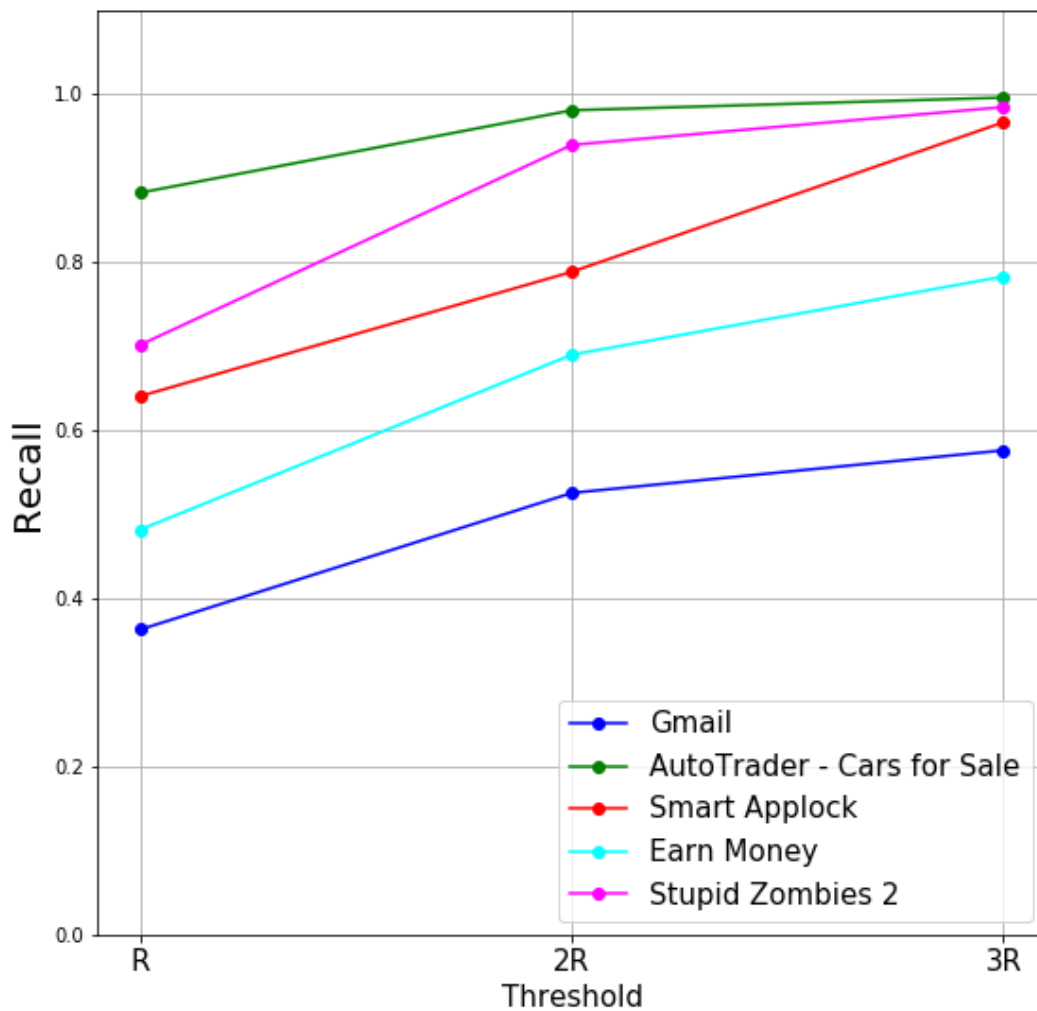


Figure 6.8: Plot Showing Recall at Thresholds of  $R$ ,  $2R$ , and  $3R$  on Test Set with Additional Training on  $R$  Relevant and  $5R$  Non-Relevant Target App Reviews



App Name	Mean R-Precision	
	$2R$	$5R$
Autotrader - Cars For Sale	85.47	88.27
AppLock	62.00	64.07
Stupid Zombies 2	70.09	70.18
Gmail	32.75	36.31
Earn Money	48.00	48.20

Table 6.3: R-Precision of Models with Additional Training Using Randomly Sampled Target App Reviews on Test Set.  $2R$  and  $5R$  indicates the number of randomly sampled negative examples in the training set

as compared to only 100. Thus, we can argue that even if there is a drop in accuracy at the start, when a developer decides to perform additional training, it is highly likely that with enough training, the performance will be better than that of the base model, and it is also likely to reach that stage faster than if the developer were to start from scratch.

One probable cause for the loss in performance when additional training is performed is the introduction of noise via the randomly sampled target app reviews. In case models already had good performance, adding randomly sampled data added noise to a model which is already stable. However, in cases where the initial performance was low, the additional data provided enough extra information that the overall performance managed to increase. While not tested, this hypothesis is consistent with all our results.

## 6.2.4 Comparison Experiment

### ALPACA

We evaluated ALPACA in the same manner as our other transfer learning experiments. Table 6.4 shows the R-Precision of ranked lists of reviews returned by ALPACA for the target set apps. For Autotrader, only 31 and 27 reviews were returned using and without using expansion respectively. Hence, we report the recall for this app. ALPACA performs the best for AutoTrader, which is also the app with the highest prevalence. Recall for ALPACA is only slightly lower than the Recall at a rank threshold of  $R$  for direct transfer.

App Name	R-Precision	R-Precision
	With Expansion	Without Expansion
Gmail	12.50	25.00
AutoTrader - Cars for Sale	71.42*	65.71*
Smart Applock	28.57	14.28
Earn Money	0.00	0.00
Stupid Zombies 2	0.00	0.00

Table 6.4: R-Precision of Ranked Lists Returned by ALPACA. \* The values for ‘AutoTrader’ are values of recall, since ALPACA returned fewer than  $R$  reviews in total.

ALPACA was unable to retrieve any relevant reviews for the app ‘Earn Money’, which is the app with the lowest scores for recall in the direct transfer experiment.

Figures 6.9 and 6.10 show the recall at thresholds of multiples of  $R$ . Since the number of reviews retrieved for ‘AutoTrader’ was less than  $R$ , we plotted the value of recall at the end of the ranked list, and show it as a green ‘ $\odot$ ’ in the plots. While there was no difference in the recall values obtained for ‘Earn Money’ and ‘Stupid Zombies 2’, ‘Smart AppLock’ showed improvement when using expansion and ‘Gmail’ showed slightly lower recall at lower rank thresholds. Performance of ALPACA using keyword expansion was better than the performance of ALPACA using only seed keywords for all apps but ‘Gmail’. We investigate this further by looking at the expanded keyword sets returned by ALPACA. The seed keyword input was ‘notification’, which was expanded to ‘filter, notification, reply’. We looked closer, and found that one review was requesting ways to filter notifications. This review is shown below:

What I’d really like to see is a way to filter notifications from this app so that maybe Important messages generate a notification, perhaps filtering from contacts as well. Any chance this can be a reality sometime

ALPACA caught on to this, and then retrieved a few more reviews that requested filters for various other aspects of the app (e.g., emails, contacts, etc). The keyword ‘reply’ had a similar effect and this caused the overall focus of ALPACA to slightly shift away from notification issues, causing a drop in R-Precision. However, this is a minor issue as it can

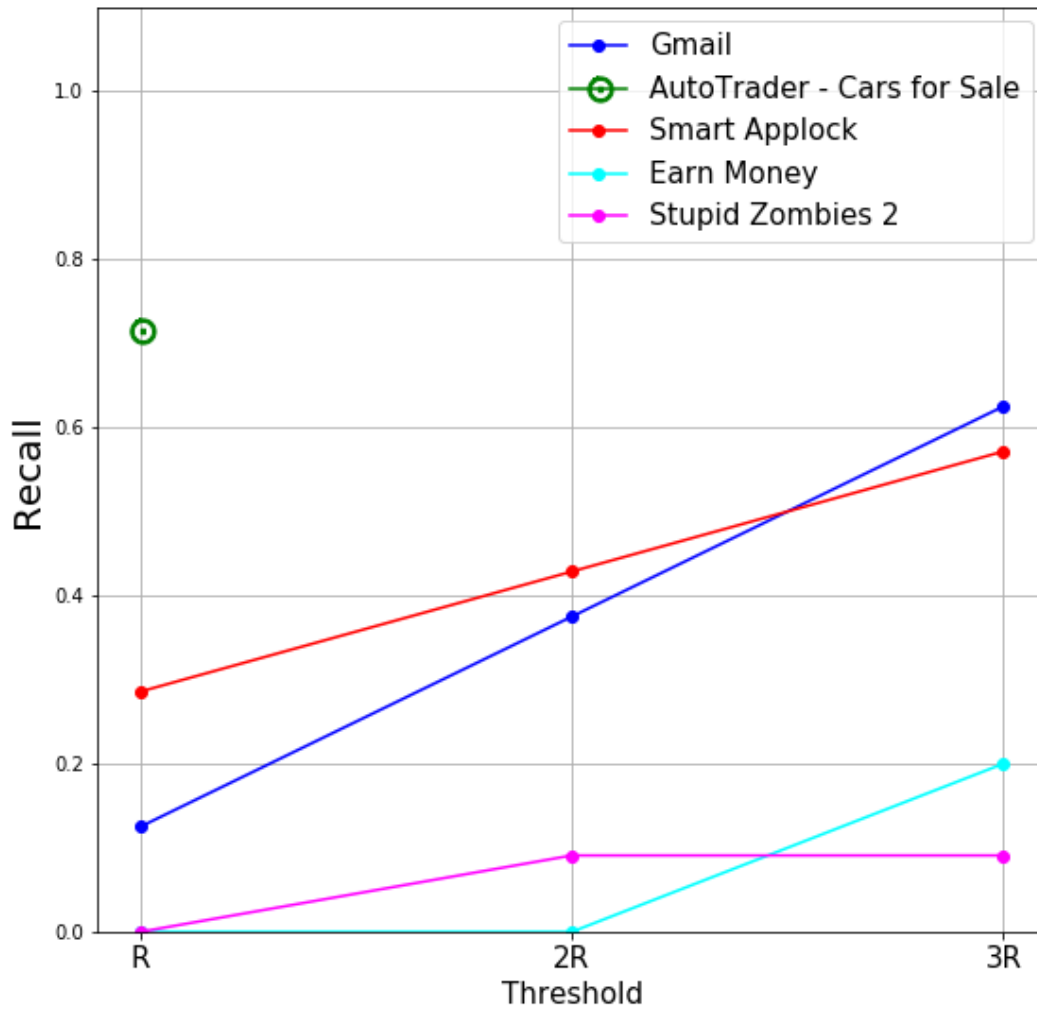


Figure 6.9: Plot showing Recall at Thresholds of  $R$ ,  $2R$ , and  $3R$  for ALPACA. Here, expanded topic keywords were used to generate ranked list. ALPACA returned only 31 reviews for Autotrader, thus, only one point is shown.

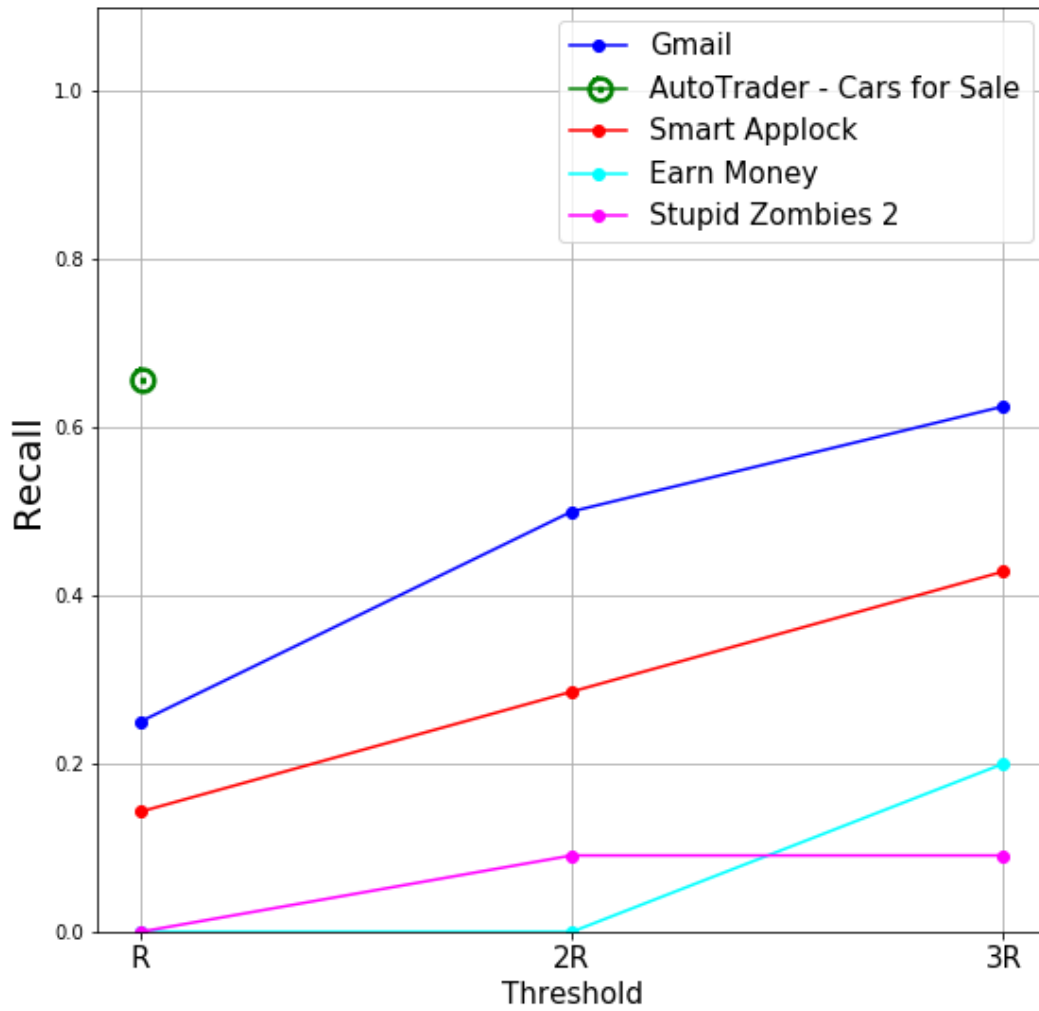


Figure 6.10: Plot showing Recall at Thresholds of  $R$ ,  $2R$ , and  $3R$  for ALPACA. Here, topic keywords without expansion were used to generate ranked list. ALPACA returned only 27 reviews for Autotrader, so we only plot one point.

be fixed by manually selecting keywords from the set expanded by ALPACA with minimal effort.

The evaluation metric scores of ALPACA in either setting is consistently lower than those in the direct transfer experiment. In the direct transfer experiment, four out of the five apps have a recall of greater than 85% at a rank threshold of  $2R$ , while three out of these five apps reached a recall of 100% before the  $3R$  threshold is reached. Transferred models with additional training also performed better than ALPACA in most cases. ‘Gmail’ had almost similar levels of recall, whereas for ‘AutoTrader’, performance by ALPACA beat some instances of our additional training experiments at a threshold of  $R$ . It is also worth noting that ALPACA had the best performance for ‘AutoTrader’, which is the app that had a much higher prevalence than all the other apps in our target set.

## CLAP

As described in Section 5.2.4, our approach for evaluating CLAP was two-fold. First, we investigated whether any of the issues we were looking for was reported as a priority task. However, to our surprise, none of them appeared as a reported issue. In total, 36 out of the 66 relevant reviews were identified and only 26 clusters out of 214 clusters had at least one relevant review in it. Since none of the issues were identified as a cluster by CLAP, we report the precision and recall of the sets reviews which were at least classified as a bug report, feature request, or non function requirement. This evaluation is reported in Table 6.5. The precision of S-CAL direct transfer is much better than that of CLAP, whereas the macro-averaged  $F_1$  score is 12.04% for CLAP compared to 71.82% at a rank threshold of  $R$  for direct transfer. Overall, we conclude that models trained using S-CAL and transferred to a different app show significantly better performance than CLAP. We ran into multiple challenges when we tried to use CLAP. They included at least one server crash. The web-app was also unable to process .csv files which had any quotation marks in them, even if the files were correctly formatted according to accepted standards. The error messages provided in case of any failure was extremely uninformative, and we were only able to correct errors in the formatting of the .csv files by contacting the authors of the paper since the documentation provided did not include any information regarding the file input format. Another major shortfall we noticed was the exclusiveness of the clusters. A review could belong to only one cluster. However, this need not be the case, as a user may have complaints about multiple issues in a single review. Neither ALPACA nor any of our methods suffer from this limitation.

App Name	Number of Clusters		Precision	Recall
	Bug Report	Feature Request		
Autotrader - Cars For Sale	56	19	10.36	57.14
AppLock	26	19	4.34	71.42
Stupid Zombies 2	18	5	12.82	45.45
Gmail	35	24	2.30	25.00
Earn Money	8	4	5.88	40.00

Table 6.5: Statistics from CLAP - Number of Clusters, Precision and Recall

### 6.2.5 Discussion

The overall performance on the test set seemed to indicate transferability of knowledge across apps facing the same issue. If app developers were willing to share the models they train, these could be used by other developers to monitor different apps. If developers were aware of the presence of an issue, they need only peruse the first few documents in the list ranked by a model to test the fit. In case there is not a good fit, they can try to train the model with additional reviews labeled from the ranked list, and try again. According to our experiments, increasing the amount of data for additional training always helps improve performance on the target set. However, developers could also repeat the entire process using a different source app to try and find a better fit. In case none of these approaches work, they always have the option of starting from scratch, which is the scenario that we studied in the user experiment.

Upon comparing the results of the direct transfer and additional transfer experiment with CLAP and ALPACA, we observed that it usually manages to perform as well as ALPACA and much better than CLAP. Another takeaway is that our approach incorporates the positives of both: similar to CLAP, we use supervised learning to improve performance, and like ALPACA, we allow topics to be defined at any time. While developers will initially spend time training models, once enough models have been trained, the time required to get a ranking model up and running for issues should decrease considerably.

# Chapter 7

## Related Works

The main focus of this thesis was to present a new framework for app review analysis. However, the work done was also the first evaluation of the performance of a high-recall search algorithm (S-CAL) through a controlled user study. Thus, in this chapter, we focus on presenting the state-of-the-art research conducted on app review analysis in detail, and in addition, provide a short summary of the latest research in high recall search methods using active learning.

### 7.1 Mobile App Review Analysis

One of the first analyses of mobile app reviews was performed in 2012, when Vasa et al. [62] published two papers studying common trends in reviews [62, 32]. Using a dataset of about 8.7 million reviews from 17,330 different apps from the Apple App Store, Vasa et al. [62] reported the lengths of app reviews, and investigated the relationship between the length of reviews, and the ratings [62]. They concluded that negative reviews are usually longer than positive ones. Hoon et al. [32] used the same dataset to mine the reviews the most popular words for different categories of apps [32]. They also studied sentiment and found that a majority of the reviews with negative polarity they identified had low ratings, whereas those with positive polarity had higher ratings.

Since 2012, there has been considerable research on analysis of mobile app reviews. In this chapter, we divide the research on app review analysis into three major categories. The first category focuses on the classification of such reviews into broad categories (e.g., bug report, feature request, etc.) [50, 48, 39, 31]. The second group of research focuses

on automatically extracting information regarding various topics present in reviews for specific apps [29, 56, 8, 33, 19]. The third category includes all other research where mobile app reviews were studied. This includes research which uses analysis of mobile app reviews to provide supporting evidence for studies focusing on ads [38, 23], security [7, 36] or permissions [25], and research done on linking app reviews to artifacts related to code and version control of code [46, 47, 26, 52].

### 7.1.1 Classification

Maalej and Nabil [39] performed an in-depth analysis of machine learning schemes to classify reviews [39]. They used multiple features sources (e.g., bag-of-words text representation, sentiment score, rating, and length of reviews) combined with multiple text preprocessing strategies (e.g., lemmatization, stopword removal) and tested these combinations using three different classifiers: Naive Bayes, logistic regression, and decision tree. An extension of their study [40] also used bigrams mined from text as a feature source. However, Maalej et al. [40] reported that there was no standout performer among classifiers. Augmenting text data with review metadata helped improve performance for some classes, whereas stopword removal using standard available lists often caused a loss in accuracy. Guzman et al. [31] also performed a similar experiment using a taxonomy of seven classes [31]. Their study, however, concluded that artificial neural networks had noticeably higher performance than SVMs, logistic regression, and Naive Bayes, whereas the performance was no worse than that of ensembles created using combinations of classifiers already trained. Guzman et al. [31] reported accuracy on a manually labeled dataset and showed neural networks had the best performance for non-ensemble methods, with a mean  $F_1$  score of 0.64, with only a slight increase to 0.65 using ensemble methods.

McIlroy et al. [43] investigated the performance of multiple classification techniques for the task of multi-label classification of reviews. They created a taxonomy of 14 categories of reviews, and compared various multi-label classification strategies (e.g., classifier chaining, binary relevance, pruned sets with threshold estimation, etc) using a variety of different classifiers (J48, SVM, Naive Bayes). They reported that SVM with pruned sets and classifier chains performed the best, with a macro-average  $F_1$  score of 0.65.

Panichella et al. [50] published a series of papers that focused on building a reusable classifier for mobile app reviews. In [49], the authors performed a comprehensive experiment designed to detect which combination of features and classifiers would perform the best at the task of classification. They focus on five classes: reviews seeking information, those giving information, bug reports, feature requests, and others. They calculated three



feature sources: term frequency matrix calculated directly from the reviews, NLP features extracted by parsing using 246 manually defined linguistic rules, and sentiment features extracted using Naive Bayes trained on a manually labeled sample set of reviews. They test J48, SVM, Naive Bayes, logistic regression, and decision tree classifiers. Training was performed on a sample set of reviews labeled by two separate evaluators and performance was reported on a held-out test set. The best performance was reported for J48 classifier using all three data sources, with an  $F_1$  score of 0.72. As a follow up, the authors released the tool AR-DOC [50]. This tool provides a graphical user interface where analysts can load their reviews and using a set of pre-trained classifiers, perform classification. Models trained using various combinations of the three features investigated in [49] were packaged as part of the tool. The tool also provides a JAVA API for programmatic usage. SURF [19] takes the output of AR-DOC and adds topic classification to group reviews into more fine-grained topics. The authors defined 10 separate topics and built a classifier based on keyword expansion (using WordNet) and filtering. The output of these two classifiers together combines to form a summary, which is then reported to the analyst. SURF was evaluated through a user evaluation study, where participants used SURF and responded to survey questions regarding performance. The authors reported that most participants found the tool to be complete in terms of category coverage, useful, and time-saving. An extension of their work is presented as AUREA [11]. The authors focused on fine-grained topics and multi-label classification. This addresses the observation that reviews often contain information about multiple issues, and that information available in reviews classified by AR-DOC are often too broad to be of help to app developers. Classification was done for 13 different categories. These categories include root causes like mobile device, Android version, usability, UI, performance, battery, etc. A web-app is provided to perform multi-label classification of upload reviews, and the reported  $F_1$  score was 0.793.

### 7.1.2 Automated Information Extraction

This section deals with research that uses unsupervised learning to present the user with a report. This report might be topics formed as a result of clustering reviews, or running topic extraction models. It might also be a set of reviews that describe the topic in question. We highlight some innovative and state-of-the-art research done on extracting topics and clusters in user reviews, summarizing these topics, or ranking reviews for automated information extraction.

In 2013, Iacob and Harrison [33] introduced MARA, a tool for automatic extraction and clustering of feature requests from app reviews. The authors created a set of keywords to heuristically filter reviews which were likely to contain feature requests. Then, the

authors manually inspected these reviews and identified common linguistic patterns that were observed in the reviews isolated using keyword filtering. MARA uses these linguistic rules to determine whether an app review contains a feature request. The authors report an accuracy of 85% for this step. Then, they used Latent Dirichlet Allocation (LDA) [4] to create topics and to group the reviews identified in the previous step, and presented these topics to app developers. These groups of feature requests are then reported to the analyst. In 2014, they released an updated version of MARA with additional linguistic rules for identification of bugs. The precision reported on a sample of the outputs was 91% , whereas for recall, they labeled all reviews of a randomly selected app and reported a recall of 89% for that app.

Aspect and Sentiment Unification Model (ASUM) [34] is a topic modeling tool that extends LDA to include sentiment information. Instead of selecting words to represent a topic as in LDA, ASUM selects pairs of (topic, sentiment). ASUM has been used to automatically extract topics by Galvis Carreño and Winbladh [20]. The authors directly applied ASUM for topic classification, and compared the results with a manually classified test set, and reported a maximum  $F_1$  score of 80%.

AR-Miner [8] was one of the first tools published that used a pipelined architecture with multiple components that employed machine learning tools for automated information extraction. At the heart of the tool is a Naive Bayes Classifier that decides which reviews are “informative” (defined as being of some use to the developer). The reviews marked informative are then fed into a topic modeling algorithm. The authors tested both LDA and ASUM methods, and their evaluation showed that LDA outperformed ASUM. These groups are then ranked using a combination of features including the number of reviews in each group, analysis of time series of incoming review volume, and ratings of reviews in each group. These groups are then visualized. Finally, the authors implemented ranking for reviews in each group using the rating, review length, and freshness, a function that returns lower values for older reviews, thus prioritizing new reviews. AR-Tracker [22] attempted to perform the same task as AR-Miner, but without the additional step of labeling to detect informative reviews. They compared various topic modeling schemes including LDA, non-negative matrix factorization, latent semantic indexing, and LDA with Gibb’s sampling and reported that LDA Gibb’s sampling performed the best.

Guzman and Maalej [30] attempted to identify app features and users’ perception of these features [30]. Collocation analysis is performed on the corpus of reviews to identify possible groups of words describing features. The reviews augmented with collocation information are provided as input to LDA to generate features. The sentiment associated with each sentence in a review is calculated separately using a dictionary-based algorithm. The analyst is then presented a report containing the frequencies of sentiment scores for

each feature. They evaluated the two parts of their framework separately. They reported an  $F_1$  score of .549 and a strong positive correlation between sentiment scores reported by their method and manual labeling.

PAID [21] is a tool by developed by Gao et al. [21] that used mutual information <sup>1</sup> to generate phrases, and then to cluster them into topics using a dynamic version of LDA [3]. A semantic score calculated using KL-divergence between topics and phrases, and a sentiment score based on rating and length of the review in which the phrase is present in was used to associate phrases representing a topic. Individual reviews within a topic were ranked based on length and rating, the assumption being that longer reviews with low ratings were likely to have more information. They reported mean precision for topics generated using PAID, and the values ranged between 0.56 and 0.77. IDEA [24] is an extension of this framework using a Adaptively Online LDA(AOLDA), novel variant of LDA. The input to AOLDA was reviews divided into time-slices, and topic generation for a particular time-slice depends on the topics present in the previous time-slices. The Jensen-Shannon divergence of a particular topic over previous time-slices was calculated, and a topic for the time-slice under investigation was reported if the divergence value was sufficiently different from the mean divergence for that topic. Phrase selection for topic representation and ranking of reviews followed the same methodology as PAID.

Finally, Gu and Kim [29] released SUR-Miner, a tool for visualization of users' perception of features. They first classified reviews into five classes: aspect evaluation, bug reports, feature requests, praise, and others. Then reviews from each category were fed into a natural language parser to create a semantic dependency graph. The authors created a set of predefined rules that extracted aspects (or features) from these dependency graphs. The sentiments for these features were calculated separately and aspects were added to create aspect-sentiment pairs. The authors then used a heuristic hierarchical clustering technique to aggregate these pairs into clusters. They reported  $F_1$  scores for individual components. The scores were 0.75, 0.85, and 0.80 for review classification, aspect-opinion extract, and sentiment analysis, respectively. They also reported that their method was superior to the method reported by Guzman and Maalej [30].

### 7.1.3 Applied Analysis

This avenue of research focuses on the application of the tools and techniques introduced by the research in the previous two categories. Here, the main focus of research was not to classify or cluster the reviews, but to extract information about specific classes of issues

---

<sup>1</sup>Mutual Information, formula available at: [https://en.wikipedia.org/wiki/Mutual\\_information](https://en.wikipedia.org/wiki/Mutual_information)

via the use of the aforementioned techniques. Below, we highlight research that use review analysis techniques to study user perception of ads, permissions, and security.

App review analysis has been used for research into user’s perception of ads in mobile apps [23, 38]. Lyu et al. [38] performed keyword filtering to identify reviews regarding mobile apps, and used those reviews to show users’ perceptions regarding mobile app reviews. Using the reviews, the authors then derived a set of guidelines for the placement of ads in apps that could potentially improve the user experience of . Research done by Gao et al. [23] focused on the user’s perception of the cost of ads in terms of data usage, battery consumption and computation. They leveraged the tool PAID [21] to extract phrases describing app issues, and then performed clustering on a word2vec embedding of these phrases to identify similar topics. They also developed a framework for automated calculation of the cost of an ad and then correlated the calculated cost to user perception. Their research suggested that users are more concerned about CPU and memory overhead than network usage.

Research from Gomez et al. [25] focused on detection of buggy apps [25]. This was achieved by creating checkers that leveraged patterns in permission requests noticed in buggy apps. Analysis of app reviews was used to identify apps with permission-related issues. They used Latent Dirichlet Allocation (LDA) to generate topic models from reviews, and manually inspected the topics formed to identify clusters that discussed errors. Error-suspicious apps were then identified by selecting apps with reviews that discussed at least one of the topics identified in the previous step. Using this method, they identified 10,658 error-suspicious apps out of 38,781 apps considered in their study. Checkers to detect buggy apps were created using patterns observed in permissions requested by the error suspicious apps, and the reported accuracy was approximately 62%.

Kong et al. [36] used app reviews to identify users’ perception of security issues faced by apps. They argued that while program analysis can reveal most security issues, it is also important to understand the users’ perceptions, and claim that for identifying financial issues, review analysis performed better than program analysis. Their app analysis pipeline uses keyword filtering to identify app reviews, semantic expansion of reviews using query expansion techniques from information retrieval, and sparse SVM on bag-of-words representation of expanded reviews for identification of security issues. They reported a mean accuracy of 94%. Reviews with information about the extracted issues is then fed to a crowd-sourcing algorithm to determine which of these issues actually exists in an app. This work was an extension of their previous work [7] which focused on the assignment of labels to reviews indicating presence of a particular type of security issue.

App reviews have also been used to supplement similarity detection and app search.

Chen et al. [9] released SimApp [9], a tool targeted at discovering apps which provide similar functionality. They defined the similarity of apps using multiple types of metadata (e.g., app description, size, permission overlap, etc.). For app reviews, they used LDA to generate topic embeddings, and calculate the similarity of the embeddings using a linear kernel. All sources of similarity were then aggregated via a linear function trained using stochastic sub-gradient method. Their method had a precision@10 of 76.9%. Park et al. [51] proposed using app reviews to help users search for apps using functionality [51]. The Play Store app search engine uses the description and metadata of an app to build a search engine. However, not all functionality of an app is listed in its description. Thus they decided to augment the app description using reviews, and performed experiments using multiple information retrieval models including BM25, query likelihood Language model and a novel topic model based approach, appLDA. AppLDA used LDA on both the app description and the app reviews and identified topics in reviews which were also mentioned in the app description. They reported a significant improvement in retrieval performance using appLDA over the traditional information retrieval methods they investigated, and also reported that augmenting app descriptions with review information helped improve performance.

Recently, there has been a focus on directly linking reviews directly to artifacts like stack traces or source code modules. Palomba et al. [47] presented research on linking app reviews to source code artifacts that needed to be changed in order to solve the issue reported in the app reviews. They used ARDOC to classify the reviews and then filtered reviews classified as bug reports or feature requests. Source Code was processed separately using methods that including extracting words from camel case method and variable names and removal of programming specific words (e.g., for, while, etc). These two data sources were linked using dice text similarity, and the authors report a precision of 81% in their experiments. This work was a follow-up to their previous work [46] where the authors leveraged AR-Miner to identify informative reviews, and then linked them to commits and issues reported in version control systems used by app developers. As in ChangeAdvisor, the authors used dice similarity to link issues and commits to user reviews as reported by AR-Miner. Further research was done by Grano et al. [26]. They attempted to link stack traces of app crashes directly to reviews. They first classified the reviews into a hierarchy of classes, with the top level dealing with high-level concepts like bug reports, feature requests, resources, etc. These were then further subdivided in a lower level taxonomy. For example, the bug reports class was further divided into two classes: crash, and features & UI. Separately stack traces extracted using SAPIENZ[41] were preprocessed, sanitized and augmented with information about methods. The authors then linked the reviews with the processed stack traces using multiple text similarity metrics including Dice, Jaccard,

and Vector Space Model. They reported an accuracy of nearly 80% using Dice, which was the best performer. They also released a tool based on their research [52].

## 7.2 Active Learning for High-Recall Search

Active learning has been used for multiple purposes including image retrieval [60], named-entity recognition [10], citation screening [67, 16], and technology-assisted review in electronic discovery [27]. Of these, electronic discovery and citation screening are high-recall tasks where the goal is to find as many relevant documents as possible. Cormack and Grossman [14] proposed Continuous Active Learning to tackle the high recall search problem. The significant difference between active learning and continuous active learning is in the selection of documents for labeling. Most active learning applications use uncertainty sampling [37, 57]. In uncertainty sampling, annotators annotate the training samples closest to the decision boundary. In CAL, annotators annotate the documents that are most likely to be relevant. In [12], the authors showed that CAL can outperform uncertainty sampling for the task of technology-assisted review. In the TREC Total Recall track [54, 28], the Baseline Model Implementation (BMI) provided to participants was an implementation of the CAL protocol. Participant submissions over two years were not able to significantly improve on its performance. Cormack and Grossman [15] have also introduced multiple criteria to decide when to stop the review process [15]. In [15], they showed that methods such as these can be used to achieve high levels of recall with a certain degree of confidence.

Active learning has also been used for citation screening as well. One example of the use of active learning for biomedical citation screening is [67]. The authors used a form of uncertainty sampling for selection of documents to label. They used domain knowledge to detect uncertainty and showed a significant improvement in results over straightforward uncertainty sampling. In the CLEF 2017 E-Health Lab [35], Cormack and Grossman [16] used BMI for the same task and their submission was the best performer.

Building on work done on CAL, Abualsaud et al. [1] released a tool for high recall search, that incorporates ad-hoc search, CAL for document and paragraph ranking, and provides a web user interface for performing these tasks. This interface was used for an user study involving 50 participants using the TREC Core Track dataset from TREC 2017 [71]. This was an extension of the authors' TREC core submission [70] with additional participants. The TREC 2017 run using CAL entered by Zhang et al. [70] had the highest MAP score and found the second most unique documents in the Track. The authors also performed a user study showing that displaying a small excerpt of a document could lead to higher levels of recall than showing assessors the whole document [70]. This confirms

the finding from the study performed by Zhang et al. [72] which showed that a single sentence could provide sufficient information for an assessor to provide relevance judgment while maintaining similar level of performance as seeing the full document. They also argued that since judging a single sentence is faster, a version of CAL selecting sentences for display to users could substantially reduce annotation overhead.

# Chapter 8

## Conclusion

In this thesis, we propose a framework that allows developers to discover information about a specific app issue while minimizing the amount of effort required by the developers to do the same. We evaluate the various aspects of this framework separately. We investigate the performance of S-CAL for information discovery and model training via a controlled user study. We investigate the accuracy of the models built via experimentation on a test set. We also investigate transfer learning methods to improve the performance of these models via simulating scenarios where developers perform additional training on the existing models.

### 8.1 Contributions

The major contributions of this thesis are summarized below:

1. Our results suggest that S-CAL can be used by developers to discover app reviews about a specific app issue. We showed that S-CAL manages to retrieve more relevant reviews than ad-hoc keyword search. We showed that S-CAL has a high precision up to the first 100 results displayed, indicating that developers can use S-CAL to quickly and precisely identify reviews that are related to a specific app issue.
2. Our study was the first evaluation of S-CAL via a controlled user experiment. Our work shows that S-CAL outperforms ad-hoc keyword search at the task of discovering documents containing relevant information.



3. Our results suggest that the models trained by participants using S-CAL for one app can be used to retrieve reviews for a separate app facing the same issue with high precision and recall. This suggests that the models trained using S-CAL were accurate. This indicates that developers will likely be able to reuse models that have already been trained for a known issue and thus further reducing the effort required.
4. We investigated transfer learning methods in an attempt to improve the performance of models that were trained by S-CAL on separate apps. Using additional training on reviews from the separate app, there was a consistent improvement noticed for cases where the initial performance was low, but for cases where the initial performance was high, we noticed a slight drop in performance.
5. The results indicate that the performance when using a greater amount of training data was always higher than when using a lesser amount of training data, indicating that models can also be used as initialization for new models to boost performance. Investigating the use of models as initialization for S-CAL remains future work.
6. We compared the performance of the models trained using S-CAL against the performance of two state-of-the-art app review analysis tools. Results strongly suggest that the performance of S-CAL for the task of retrieving reviews about a particular topic was consistently better than that of the state-of-the-art.

In summary, we conclude that S-CAL can be used to build a framework for the analysis of mobile app reviews that help developers identify reviews related to an issue their app faces. This framework would allow developers to efficiently discover reviews about a new app issue, while at the same time, training a model to retrieve reviews about the issue in the future. It would also allow developers to reuse existing models trained on a specific app issue, directly or with a small amount of additional training, to retrieve reviews related to the issue in question. This would allow developers to discover more information about a specific app issue while reducing the effort required to do so.

# Bibliography

- [1] Mustafa Abualsaud, Nimesh Ghelani, Haotian Zhang, Mark D Smucker, Gordon V Cormack, and Maura R Grossman. A system for efficient high-recall retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1317–1320. ACM, 2018.
- [2] Jason R Baron, David D Lecwis, and Douglas W Oard. Trec 2006 legal track overview. 2006.
- [3] David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, New York, NY, USA, 2006. ACM.
- [4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, pages 993–1022.
- [5] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.
- [6] Stefan Büttcher, Charles Clarke, and Gordon V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. The MIT Press, 2010.
- [7] Lei Cen, Luo Si, Ninghui Li, and Hongxia Jin. User comment analysis for android apps and cspi detection with comment expansion. In *PIR@ SIGIR*, pages 25–30. Citeseer, 2014.
- [8] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. Arminer: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014*, New York, NY, USA, 2014. ACM.

- [9] Ning Chen, Steven CH Hoi, Shaohua Li, and Xiaokui Xiao. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 305–314. ACM, 2015.
- [10] Yukun Chen, Thomas A Lasko, Qiaozhu Mei, Joshua C Denny, and Hua Xu. A study of active learning methods for named entity recognition in clinical text. *Journal of biomedical informatics*.
- [11] Adelina Ciurumelea, Sebastiano Panichella, and Harald C Gall. Automated user reviews analyser. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pages 317–318. ACM, 2018.
- [12] Gordon V. Cormack and Maura R. Grossman. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, New York, NY, USA, 2014. ACM.
- [13] Gordon V. Cormack and Maura R. Grossman. Scalability of continuous active learning for reliable high-recall text classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16. ACM, 2016.
- [14] Gordon V Cormack and Maura R Grossman. Continuous active learning for tar. *Electronic Discovery Bulletin*, April-May 2016.
- [15] Gordon V. Cormack and Maura R. Grossman. Engineering quality and reliability in technology-assisted review. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, New York, NY, USA, 2016. ACM.
- [16] Gordon V Cormack and Maura R Grossman. Technology-assisted review in empirical medicine: Waterloo participation in clef ehealth 2017. *Working Notes of CLEF*, 2017.
- [17] Gordon V. Cormack, Christopher R. Palmer, and Charles L. A. Clarke. Efficient construction of large test collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, New York, NY, USA, 1998. ACM.
- [18] Nick Craswell, David Hawking, Ross Wilkinson, and Mingfang Wu. Overview of the trec 2002 web track. In *TREC*, 2003.

- [19] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, New York, NY, USA, 2016. ACM.
- [20] Laura V Galvis Carreño and Kristina Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 582–591. IEEE Press, 2013.
- [21] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu. Paid: Prioritizing app issues for developers by tracking user reviews over versions. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2015.
- [22] Cuiyun Gao, Hui Xu, Junjie Hu, and Yangfan Zhou. Ar-tracker: Track the dynamics of mobile apps via user review mining. In *2015 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 284–290. IEEE, 2015.
- [23] Cuiyun Gao, Yichuan Man, Hui Xu, Jieming Zhu, Yangfan Zhou, and Michael R Lyu. Intelliad: assisting mobile app developers in measuring ad costs automatically. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 253–255. IEEE Press, 2017.
- [24] Cuiyun Gao, Jichuan Zeng, Michael R. Lyu, and Irwin King. Online app review analysis for identifying emerging issues. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 48–58, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5638-1. doi: 10.1145/3180155.3180218. URL <http://doi.acm.org/10.1145/3180155.3180218>.
- [25] Maria Gomez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier. *A recommender system of buggy app checkers for app store moderators*. PhD thesis, Inria Lille; INRIA, 2014.
- [26] Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C Gall. Exploring the integration of user feedback in automated testing of android applications. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 72–83. IEEE, 2018.
- [27] Maura R Grossman and Gordon V Cormack. Technology-assisted review in e-discovery can be more effective and more efficient than exhaustive manual review. *Rich. JL & Tech.*, 2010.

- [28] Maura R Grossman, Gordon V Cormack, and Adam Roegiest. Trec 2016 total recall track overview. 2016.
- [29] X. Gu and S. Kim. "what parts of your apps are loved by users?" (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2015.
- [30] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug 2014.
- [31] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, Washington, DC, USA, 2015. IEEE Computer Society.
- [32] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and Kon Mouzakis. A preliminary analysis of vocabulary in mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 245–248. ACM, 2012.
- [33] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, May 2013.
- [34] Yohan Jo and Alice H Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 815–824. ACM, 2011.
- [35] Evangelos Kanoulas, Dan Li, Leif Azzopardi, and Rene Spijker. Clef 2017 technologically assisted reviews in empirical medicine overview. In *CEUR Workshop Proceedings*, volume 1866, 2017.
- [36] Deguang Kong, Lei Cen, and Hongxia Jin. Autoreb: Automatically understanding the review-to-behavior fidelity in android applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 530–541. ACM, 2015.
- [37] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

- [38] Yingjun Lyu, Jiaping Gui, Mian Wan, and William GJ Halfond. An empirical study of local database usage in android applications. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 444–455. IEEE, 2017.
- [39] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, Aug 2015.
- [40] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, 2016.
- [41] Ke Mao, Mark Harman, and Yue Jia. Sapienz: multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 94–105. ACM, 2016.
- [42] Stuart Mcilroy. *Empirical studies of the distribution and feedback mechanisms of mobile app stores*. PhD thesis, 2014.
- [43] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, 21(3):1067–1106, 2016.
- [44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS’13*, pages 3111–3119, USA, 2013. Curran Associates Inc.
- [45] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, July 2013.
- [46] Fabio Palomba, Mario Linares-Vasquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 291–300. IEEE, 2015.
- [47] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald Gall, Filomena Ferrucci, and Andrea De Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering, ICSE ’17, Piscataway, NJ, USA, 2017*. IEEE Press.

- [48] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ICSME '15, Washington, DC, USA. IEEE Computer Society.
- [49] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Software maintenance and evolution (IC-SME), 2015 IEEE international conference on*, pages 281–290. IEEE, 2015.
- [50] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, New York, NY, USA, 2016. ACM.
- [51] Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. Leveraging user reviews to improve accuracy for mobile app retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 533–542. ACM, 2015.
- [52] Lucas Pelloni, Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C Gall. Becloma: Augmenting stack traces with user review information. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 522–526. IEEE, 2018.
- [53] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, April 2009.
- [54] Adam Roegiest, Gordon V Cormack, Maura R Grossman, and C Clarke. Trec 2015 total recall track overview. *The Twenty-Fourth Text REtrieval Conference (TREC 2015) Proceedings*, 2015.
- [55] Mark Sanderson and Hideo Joho. Forming test collections with no system pooling. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '04*, pages 33–40, New York, NY, USA, 2004. ACM. ISBN 1-58113-881-4. doi: 10.1145/1008992.1009001. URL <http://doi.acm.org/10.1145/1008992.1009001>.

- [56] S. Scalabrino, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Listening to the crowd for the release planning of mobile apps. *IEEE Transactions on Software Engineering*, pages 1–1, 2018. ISSN 0098-5589. doi: 10.1109/TSE.2017.2759112.
- [57] Burr Settles. Active learning literature survey. Technical Report 1648, 2009.
- [58] Ian Soboroff, Arjen P de Vries, and Nick Craswell. Overview of the trec 2006 enterprise track. 2006.
- [59] Fei Song and W Bruce Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM, 1999.
- [60] Simon Tong and Edward Chang. Support vector machine active learning for image retrieval. In *Proceedings of the Ninth ACM International Conference on Multimedia, MULTIMEDIA '01*, New York, NY, USA, 2001. ACM. ISBN 1-58113-394-4.
- [61] Lisa Torrey and Jude Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 2009.
- [62] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 241–244. ACM, 2012.
- [63] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 14–24, May 2016. doi: 10.1145/2884781.2884818.
- [64] Andreas Vlachos. A stopping criterion for active learning. *Comput. Speech Lang.*, July 2008.
- [65] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, Washington, DC, USA, 2015. IEEE Computer Society.
- [66] Phong Minh Vu, Tung Thanh Nguyen, Hung Viet Pham, et al. Alpaca-advanced linguistic pattern and concept analysis framework for software engineering corpora. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pages 284–285. ACM, 2018.



- [67] Byron C. Wallace, Kevin Small, Carla E. Brodley, and Thomas A. Trikalinos. Active learning for biomedical citation screening. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, New York, NY, USA, 2010. ACM.
- [68] Peter Willett. The porter stemming algorithm: then and now. *Program*, 40(3):219–223, 2006.
- [69] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
- [70] Haotian Zhang, Mustafa Abualsaud, Nimesh Ghelani, Angshuman Ghosh, Mark D Smucker, Gordon V Cormack, and Maura R Grossman. Uwaterloomds at the trec 2017 common core track.
- [71] Haotian Zhang, Mustafa Abualsaud, Nimesh Ghelani, Mark D Smucker, Gordon V Cormack, and Maura R Grossman. Effective user interaction for high-recall retrieval: Less is more. In *Proceedings of the 2018 ACM on Conference on Information and Knowledge Management*. ACM, 2018.
- [72] Haotian Zhang, Gordon V Cormack, Maura R Grossman, and Mark D Smucker. Evaluating sentence-level relevance feedback for high-recall information retrieval. *arXiv preprint arXiv:1803.08988*, 2018.

# APPENDICES

# Appendix A

## Participant Instructions

### A.1 General

- You will be given a category (e.g. Battery Drainage Complaints) and will be asked to do your best to find as many reviews as you can which belongs to the category. Both interfaces have buttons that let you mark these reviews as relevant (belongs to the category) or non-relevant (does not belongs to the category).
- However, while it is important to find as many of these reviews as you can, you have to make a best effort to not mark reviews which do not belong to the category as relevant.
- The idea is to find and mark as many reviews as possible, while reducing the total number of reviews viewed.

### A.2 Search Interface

- In this interface you will have to enter keywords (e.g. “drains battery”) to search for the reviews and mark them. The interface provides the following features:
  - The documents that are more likely to be relevant are ranked higher in the results.

- Once you have judged a review, you can use the buttons at the top of the screen to show or hide reviews that you have already marked. The buttons can be used to hide all reviews you marked relevant, all buttons you marked non-relevant, or both, and vice-versa.
- The page has infinite scrolling. There’s a button at the end of the page to load more results.
- The “mark all above Relevant” button can be used in two different ways:
  - \* To mark search results above as relevant
  - \* To mark search results above that has not been marked before as relevant
 Since the results higher up in the list are more likely to be relevant, this is one way to quickly mark a lot of reviews. However, you must remember that marking a lot of reviews out of which only few are actually relevant is also not the goal of this study. Please exercise caution while using this button.
- Judgments are not irreversible. You can change your judgment at any time. So a review that has been marked relevant can be marked non-relevant and vice versa any number of times.

- Search Features:

- You can use quotation marks to search for an exact phrase (i.e. If you searched for “phrase search”, you would get matches for documents where the words ‘phrase’ and ‘search’ appear together in that order.)
- You can use the plus sign to indicate that a word must be present. (i.e. if you searched for +battery, only documents that have the word will be present in the result list.)
- You can use the minus sign to indicate that a word must not be present. (i.e. if you searched for -battery, documents that have the word will be not present in the result list.)
- You can combine + or with quotation marks to make sure a phrase is present or absent in all search results. However, you cannot combine the plus and minus signs together.

### A.3 S-CAL Interface

- In this interface, you will be given a query (e.g. ”drains battery”) that you will input. After this, you will be shown one review which you will have to mark.

- There might be delays during which the algorithm re-ranks the results based on your responses. This is not cause to be concerned.