Refresh Strategies in Continuous Active Learning

by

Nimesh Ghelani

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

Waterloo, Ontario, Canada, 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

High recall information retrieval is crucial to tasks such as electronic discovery and systematic review. *Continuous Active Learning* (CAL) is a technique where a human assessor works in loop with a machine learning model; the model presents a set of documents likely to be relevant and the assessor provides relevance feedback. Our focus in this thesis is on one particular aspect of CAL: *refreshing*, which is a crucial and recurring event in the CAL process. During a *refresh*, the machine learning model is trained with the relevance judgments and a new list of likely-to-be-relevant documents is produced for the assessor to judge. It is also computationally the most expensive step in CAL. In this thesis, we investigate the effects of the default and alternative refresh strategies on the effectiveness and efficiency of CAL. We find that more frequent refreshes can significantly reduce the human effort required to achieve certain recall. For moderately sized datasets, the high computation cost of frequent refreshes can be reduced through a careful implementation. For dealing with resource constraints and large datasets, we propose alternative refresh strategies which provide the benefits of frequent refreshes at a lower computation cost. In this thesis, we also discuss the design of a modern implementation of the CAL algorithm which is efficient and extensible. Our implementation can be used as a research tool as well as for practical applications.

Acknowledgements

I would like to thank my supervisor Dr. Mark D. Smucker for his effort and guidance throughout my time here. Mark gave me the freedom and support to work on problems I was interested in, for which I am very grateful.

I thank Dr. Gordon V. Cormack and Dr. Maura R. Grossman for agreeing to be the readers of this thesis. Their course on "High Stakes Information Retrieval" played a significant role in defining my research interests. Gordon's feedback and suggestions were crucial for the content in this thesis.

I had the privilege of being part of an amazing research group; Mustafa Abualsaud and Haotian Zhang were a joy to work with. I am also very thankful to the members of the Data System Group: Dr. Jimmy Lin, Amine Mhedhbi, Angshuman Ghosh, Chathura Kankanamge, Royal Sequeira, Shahin Rahbariasl, Siddhartha Sahu and Vineet John; who enriched my time in University of Waterloo in many different ways.

Finally, I would like to thank my parents and brother for their constant support. Their presence, despite being thousands of miles away, was a great source of encouragement.

Dedication

This thesis is dedicated to everyone who directly or indirectly, made it possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

High Recall Information Retrieval is crucial for applications where the goal is to find all or nearly all relevant documents using minimal human effort. In information retrieval, *recall* is defined as the fraction of relevant documents retrieved, and *precision* is defined as the fraction of retrieved documents which were relevant. High recall retrieval systems are designed to achieve high recall (finding all or nearly all relevant documents) while maintaining a high precision (using minimal human effort on non-relevant documents). This is in contrast to popular web-based search engines which are optimized for early precision. Such retrieval systems are built to deliver a small set of highly precise (but not necessarily high recall) results to its users.

High recall retrieval systems are useful in tasks such as (but not limited to) electronic discovery, systematic review, patent search and literature review. In evidence based medicine, *systematic review* is the process of summarizing all the evidence pertaining to a research question. This often involves performing an exhaustive literature review of relevant biomedical research and studies done in the past. Finding all the relevant citations from a vast biomedical literature is a high recall problem. *eDiscovery* is another such high recall problem. In the legal domain, *eDiscovery* refers to the process followed by a party which is required to find and present a (practically) complete set of documents relevant to a particular matter. Finding and reviewing documents from potentially millions of documents can become very expensive in absence of specialized retrieval systems. The requirements of retrieval tasks in eDiscovery and systematic review share various similarities [22, 28]. Both these tasks have traditionally relied on extensive manual review with limited computer assistance. The search space these retrieval tasks operate on has been expanding at a massive rate [3, 29]. As a result, systems which help achieve high recall while reducing the work of human reviewers are becoming more desirable.

Various approaches addressing the problem of high recall information retrieval under various applications exist [5, 20, 23]. Technical Assisted Review (TAR) is an umbrella term for computer assisted techniques used to perform eDiscovery. In a TAR method, a computer system works alongside humans to classify documents as either relevant or non-relevant. TAR methods usually incorporate some form of relevance feedback mechanism to gain a better understanding of the search task. An effective TAR process would require the human assessor to process a small fraction of the entire document set and use that information to precisely predict the relevance of the remaining documents. TAR methods have shown to outperform manual review in legal eDiscovery by reducing the cost spent on human assessors [14, 33]. Continuous active learning (CAL) [5, 6] is a TAR protocol where a machine learning algorithm suggests most likely relevant documents for human assessment and continuously incorporates relevance feedback to improve its understanding of the search task. In a previous study, Cormack and Grossman [5] showed that CAL outperforms other TAR protocols on review tasks from actual legal matters and TREC 2009 Legal Track. The Total Recall track in TREC 2015 and 2016 evaluated different systems under a simulated TAR setting [17, 32]. Baseline Model Implementation (BMI) based on CAL was used as the baseline in these tracks. None of the participating systems were able to consistently outperform the baseline. BMI implements the AutoTAR (Autonomous TAR) algorithm [6], which enhances the autonomy of CAL by requiring only a single seed relevant document or query to bootstrap the TAR process.

In this thesis, we modify and extend the AutoTAR CAL algorithm. We isolate an important step of the algorithm and call it *refreshing*. During a *refresh*, the relevance judgments from the assessor are used to train a new classifier. This classifier generates an ordered list of documents most likely to be relevant, which is later processed by the assessor. Apart from being a crucial factor in the effectiveness of the retrieval algorithm, this step also has a high computation cost because it involves training a classifier and computing relevance likelihood scores for potentially all the documents in the corpus. A *refresh strategy* determines when and how to perform the refresh. For example, in the AutoTAR algorithm, refreshing is done after a certain number of documents are assessed. This number increases exponentially over time. During a refresh, a classifier is trained using all the available relevance judgments, and the entire document collection is scored to produce an ordered set of most-likely-to-be-relevance documents for human assessment.

We propose various refresh strategies and compare their impact on the effectiveness and efficiency of CAL. Following are the contributions we made through the work described in this thesis:

- Effectiveness of CAL (specifically, its ability to achieve higher recall with less effort)

can be improved by performing more frequent refreshes. We found that refreshing after every assessment can consistently outperform other refresh strategies.

- Frequent refreshing can be computationally expensive for large datasets or low resource environment. We propose strategies which perform refresh on a smaller subset of data or use precision as a refresh criteria. These strategies achieve similar effectiveness as the most effective strategy (i.e., refreshing after every assessment), while being efficient with computation.

- We investigated prioritization of recently judged documents as a way to enhance the training of the classifier. While recency weighting didn't result in any improvements, it did help recover a fraction of the lost effectiveness when used with a weak but faster setting of the training algorithm.

- The work described in this thesis required a suitable implementation of CAL. We designed a modern and efficient implementation of CAL which can support aggressive refresh strategies. Our tool is open source [1] and designed to be used both as a research tool and in real world applications.

1.1 Thesis Organization

The organization of the remainder of this thesis is described below.

In Chapter 2, we introduce prerequisites to understanding the subsequent chapters of this thesis. We discuss the Continuous Active Learning algorithm and define the concept of *refresh* and *refresh strategy*. We then review related work approaching similar problems and work which can be potentially applied to the problem addressed in this thesis.

In Chapter 3, we describe the design and features of the implementation using which all the experiments in this thesis were performed.

In Chapter 4, we discuss the design of our experiments, along with the dataset and evaluation metrics used.

In Chapter 5, we define and explain various refresh strategies. In Chapter 6, we evaluate and compare the performance of these refresh strategies.

In Chapter 7, we discuss the conclusions of our work and the future work addressing various practical applications.

[1] https://github.com/HiCAL/HiCAL/tree/master/CALEngine

Chapter 2

Background and Related Work

2.1 High Recall Information Retrieval

High Recall Information Retrieval deals with retrieval problems where the goal is to find all or nearly all relevant documents while maintaining a high precision. *Recall* of a retrieval system is defined as the fraction of relevant documents it retrieved, while *precision* is defined as the fraction of retrieved documents which were relevant. Legal eDiscovery is one such high-recall problem where all the evidence related to some matter is required to be retrieved from a digital collection of information [27, 28]. The eDiscovery process involves lawyers reviewing documents and it is desirable to keep the review costs reasonable by minimizing the amount of non-relevant documents reviewed. High recall problems also extend to patent retrieval [25] and systematic reviews [40] in the medical/health space. In information retrieval, test collections are used to evaluate and compare different retrieval systems. A test collection comprises of a document corpus, a set of queries (or information needs) and for each query, relevance judgments on a set of documents. To build test collections, conferences such as the Text REtrieval Conference (TREC) employ human assessors to manually label a fixed number of documents retrieved by multiple retrieval systems. Building test collections can also be interpreted as a high-recall problem because it is desirable to find all relevant documents using fixed amount of assessor effort.

Traditional web search systems address a different set of information retrieval problems where the goal is to deliver a small ordered list of highly relevant documents, which are preferably at top. Evaluation of such systems is usually done using the Cranfield paradigm [38], using metrics such as precision at certain ranks, MAP (Mean Average Precision), nDCG (Discounted Cumulative Gain), and so on. Such retrieval systems and

evaluation techniques usually favour early precision and cannot be directly used in the context of high recall retrieval [26, 31]. In addition to measuring recall, it is important to factor in the effort spent by the user when evaluating high recall systems. The search tasks in many high recall applications are complex and even the user's understanding of the task requirements may change during the search process. Some systems used for high recall tasks allow the users to perform series of complex queries to express their information needs while some systems automatically try to learn and adapt to the information needs of the user.

Earlier retrieval systems such as IBM STAIRS relied on keyword queries provided by the assessor along with various options (such as boolean combinations) to retrieve documents [4]. Cormack et al. [10] proposed Interactive Search and Judging (ISJ) where multiple searchers used an ad-hoc search engine to build a test collection using a fraction of assessor cost when compared to the traditional NIST pooling [19]. In the traditional NIST pooling, human assessors judged a pool of documents formed by taking fixed number of top ranked documents from multiple retrieval systems.

Technology Assisted Review (TAR) is a set of computer assisted techniques used to perform eDiscovery. TAR systems typically uses judgments made by human assessors to classify documents as either relevant or non-relevant [15]. The classifier can control the documents which should be assessed during a TAR process. TAR methods outperform manual review or traditional keyword based search methods in legal eDiscovery by reducing the cost spent on human assessors [14, 33]

Cormack and Grossman [5] compared three TAR protocols, namely Continuous Active Learning (CAL), Simple Active Learning (SAL), and Simple Passive Learning (SPL). During a CAL process, a machine learning algorithm suggests most-likely-to-be-relevant documents for review and continuously incorporates relevance feedback to improve its understanding of the search task. SAL differs from CAL by having a separate training and review phase. During the training phase, SAL uses uncertainty sampling to select documents to be reviewed until a stable classifier is obtained. In the second phase, the classifier is used to produce a ranked list of documents for review. SPL relies on the user or random sampling to select documents for training a classifier. Cormack and Grossman [5] showed that CAL outperforms other TAR protocols on review tasks from actual legal matters and TREC 2009 Legal Track. The Total Recall track in TREC 2015 and 2016 evaluated different systems under a simulated TAR setting [17, 32]. None of the participating teams were able to consistently beat the BMI (Baseline Model Implementation), which implemented the AutoTAR CAL algorithm [6]. The AutoTAR (Autonomous TAR) CAL algorithm differs from the previous flavours of CAL by only requiring the user to provide a relevant seed document or seed query to bootstrap. AutoTAR also samples random documents as neg-

ative examples for training and processes relevance judgments in exponentially increasing batch sizes.

Li et al. proposed a double loop process [23] where an outer loop uses a set of queries to retrieve a pool of documents and an inner loop uses a classifier to select documents from the pool for assessment. The classifier uses the relevance feedback from the assessor to update itself. Once the classifier is stable, a new set of queries are added to the outer loop based on the newly retrieved relevant documents. Their work is very similar to how CAL approaches the TAR problem.

Scalable Continuous Active Learning (or "S-CAL") [8] was designed to work with large document collections where it is desirable to build an effective classifier using minimal labelling effort or estimate metrics like recall and precision. It is a modification to the AutoTAR algorithm where only a sample from a larger batch is assessed by a human.

HiCAL[1] [1] is a "system for efficient high-recall retrieval" which combines ISJ and CAL. It allows assessors to find relevant documents by switching between an ad-hoc search engine and a CAL-powered review interface. To improve assessment throughput, the system by default only presents a summary of the document and assessors can optionally click an extra button to view the full document.

2.2 Continuous Active Learning

A general version of the AutoTAR CAL algorithm is described in Algorithm 1. The CAL process bootstraps using a user provided query and 100 randomly sampled documents from the document collection. The former is treated as a relevant document and the rest are treated as non-relevant in the training set. The training set is then used to train a Logistic Regression classifier. Using the classifier, relevance likelihood scores are computed for unjudged documents in the collection and top documents are pushed to the review queue. The assessor judges documents from the review queue as relevant or non-relevant. These judgments are added to the training set. This feedback loop continues until some stopping criteria is met. The stopping criteria could be the assessor time allotted for the task, some target number of relevant documents or reaching an estimated value of recall [7]. In the experiments reported in this thesis, we simulate human assessors using a set of existing relevance judgments (Step 8). Unlabelled documents are considered non-relevant during the simulation.

[1] https://hical.github.io/

We define the term *refresh* as the set of steps in the CAL process which deals with processing user judgments. This includes training the classifier, scoring documents and selecting documents for review. The choice of refresh strategy can control when to perform a refresh (step 10), as well as the behaviour of training (step 4) and scoring (step 7). Figure 2.1 shows a simplified view of the relevance feedback loop in CAL, highlighting the steps in a refresh. In the AutoTAR algorithm, refresh is performed after a batch of judgments is received. The size `k` of the batch is initially set to 1. After each refresh, this size is updated using

$$k \leftarrow k + \lfloor \frac{k+9}{10} \rfloor$$

Algorithm 1: AutoTAR CAL Algorithm (assuming an arbitrary refresh strategy). A refresh strategy can alter/control behaviour of steps 4, 7 and 10

1. Construct a seed document whose content is a user provided query
2. Label the seed document as relevant and add it to the training set
3. Add 100 random documents from the collection, temporarily labeled as "not relevant"
4. Train a Logistic Regression classifier using the training set
5. Remove the random documents from the training set added in step 3
6. Flush the review queue
7. Using the classifier, order documents by their relevance scores and put them into a review queue
8. Review a document from the review queue, coding it as "relevant" or "not relevant"
9. Add the document to the training set
10. Repeat steps 8-9 until a refresh is needed (defined by the refresh strategy)
11. Repeat steps 3-10 until some stopping condition is met.

The organizers of the TREC 2015 Total Recall Track distributed the Baseline Model Implementation (BMI) of AutoTAR as a virtual machine. The implementation was a collection of C++ programs invoked and orchestrated using few BASH scripts. In BMI, documents are represented as a vector of unigram tf-idf features which are used for training the classifier and calculating relevance likelihood scores. BMI relies on sofia-ml[2] [36] to train a logistic regression classifier using the *logreg-pegasos* learner with 200000 iterations of *roc* sampling. A training iteration involves randomly sampling a relevant and a non-relevant

[2]https://code.google.com/archive/p/sofia-ml/

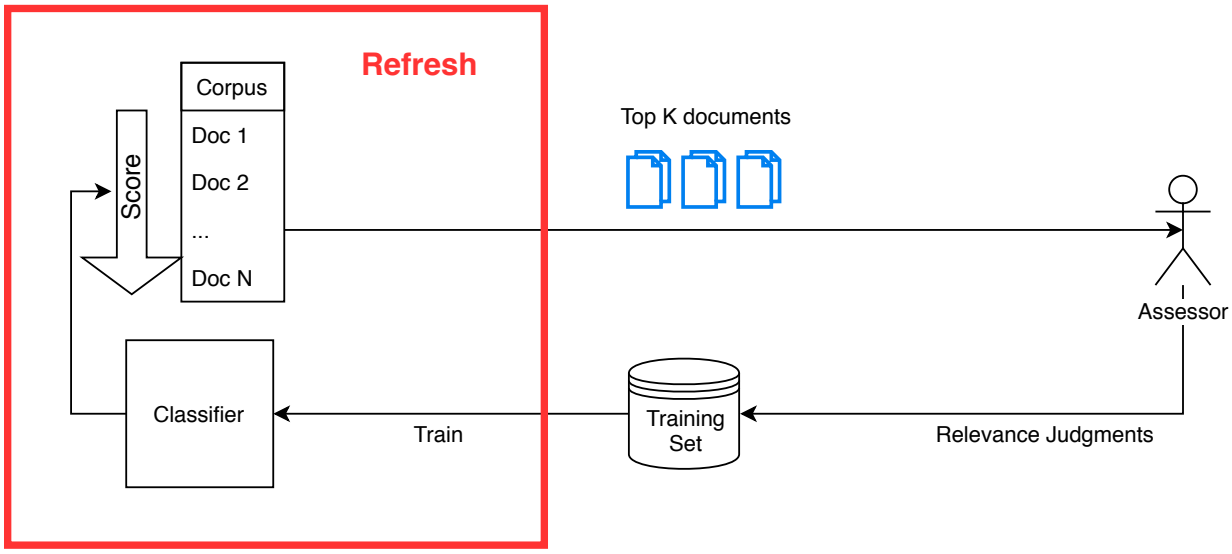


Figure 2.1: A simplified view of the relevance feedback loop in Continuous Active Learning

document from the training set, computing the loss and adjusting the classifier weights accordingly. The relevance likelihood score for any document is obtained by computing the dot product of the classifier weights and document feature vector.

2.3 Related Work

The Baseline Model Implementation (BMI) was made available to the participants by the organizers of the Total Recall Track in TREC 2015. We take a look at few approaches of participants which are relevant to our interest.

The UvA.ILPS team [37] modified the way batch sizes were set in BMI. They started with a batch size of 100. After a batch was assessed, the batch size was set to some value proportional to the number of relevant documents assessed in the previous batch.

The Webis team [18] used BM25 using the topic description as query to get an initial set of documents which were used to train a SVM classifier. The batch size of to-be-assessed-documents was initially set to 32. After every iteration, it was either halved, doubled, or untouched depending on the ratio of documents assessed relevant and non-relevant. Another run submitted by this team used keyphrase extraction from document assessed as relevant to perform ad-hoc searches at every step and use it to enhance the SVM classifier.

The TUW team [24] tweaked the term weighting in the document feature vectors used by the BMI. In another run, they used all the learners available in sofia-ml separately and then used voting to select relevant documents.

There is a vast literature which separately addresses efficiency and scalability of various steps in the AutoTAR algorithm. Online learning algorithms can significantly improve the running times of the training step. Such algorithms have been investigated in detail for spam filtering [9, 34, 35]. Cormack et al. [11] proposed an online logistic regression based spam filter for large datasets. Crammer et al. [12] proposed a SVM-based Passive-Aggressive algorithm for online binary classification. Both of these online learning methods make a prediction for an incoming training examples and depending on the ground truth, updates the classifier weights.

Chapter 3

Implementation

In this chapter, we discuss our implementation of CAL. We used this implementation to perform all the experiments described in the future chapters. We start by reviewing an early implementation of CAL, followed by our goals and expectations from the new implementation. We then briefly discuss the design decisions, along with the capabilities of our system.

The code for our system is open source and available with documentation online[1].

3.1 Auto-TAR BMI

As mentioned in the previous chapter, BMI(Baseline Model Implementation) is an implementation of CAL which was used as a baseline in the TREC 2015 and 2016 Total Recall track. Most of BMI is written as a BASH script, which coordinates various external C/C++ programs to perform more more specific tasks like training and scoring. The intermediate data is stored in text files since BASH is not designed to support advanced data structures.

The AutoTAR BMI was released by the Total Recall track organizers as a virtual machine[2]. A local version of the tool also available[3].

[1] https://hical.github.io/

[2] https://plg.uwaterloo.ca/~gvcormac/trecvm/

[3] https://github.com/HTAustin/CAL

3.2 A modern CAL implementation

3.2.1 Motivation and Goals

The design of BMI limit us from designing refresh strategies which require more computation or involve complex logic. Moreover, while BMI is suitable for simulation purposes, it is difficult to use in real world applications.

As part of our TREC 2017 Core effort [41], we designed a CAL-powered review tool called HiCAL [1]. We intended this tool to process relevance feedback as quickly as possible (in other words, more frequent refreshes). For a smooth user experience, we wanted to achieve this with minimum possible system delay. Due to these reasons, we decided to implement CAL from scratch in C++. We chose C++ because it provides a good control over efficiency and is usually easy to maintain/extend.

We wanted to build a system which could satisfy all the requirements of HiCAL, and could be easily used for other applications and future research, such as the work presented in this thesis. We designed this system to meet the following goals:

- Fast and efficient.
- Support parallel tasks (for parallel simulations and multiple users).
- Easy to use as a standalone tool or as a part of an external application.
- Easy to extend and modify any step of the CAL algorithm.

3.2.2 Design

In this section, we briefly discuss the design details of our CAL system.

Prior to using the CAL system, a one-time preprocessing step is required to convert a given text collection to a machine readable set of feature vectors. The corpus processor takes as input an archive (a tar.gz file) of text documents, computes the feature vectors for every document, and writes them to a binary file. We use binary files over human-readable svmlight files [21] to significantly improve the output file size and loading times. When working with the athome1 test collection (see Chapter 4 for more details), the document feature vectors when stored in the svmlight format took 734 megabytes of disk space and 8.6 seconds to process+load into the memory by the CAL system. On the same machine,

our binary file format took 341 megabytes of disk space and 2.7 seconds to process+load into the memory. The athome1 collection has only 290k documents and these differences become more significant when working with larger collections. For the New York Times collection which has around 1.8 million documents, the svmlight format took 7.6 gigabytes of disk space and 90.1 seconds to process+load into the memory, while our binary file format took 3.5 gigabytes of disk space and 30.1 seconds to process+load into the memory.

Each document is assigned an ID which is its base filename in the input archive (filename ignoring the directory structure). This ID is used to judge and retrieve documents in the CAL system. The files are treated as plaintext files and are assumed to be cleaned as per the needs of the user. Figure 3.1 shows a preprocessing pipeline example with the New York Times collection. By default, the corpus preprocessor computes tf-idf unigram features for every document as explained in Section 2.2.

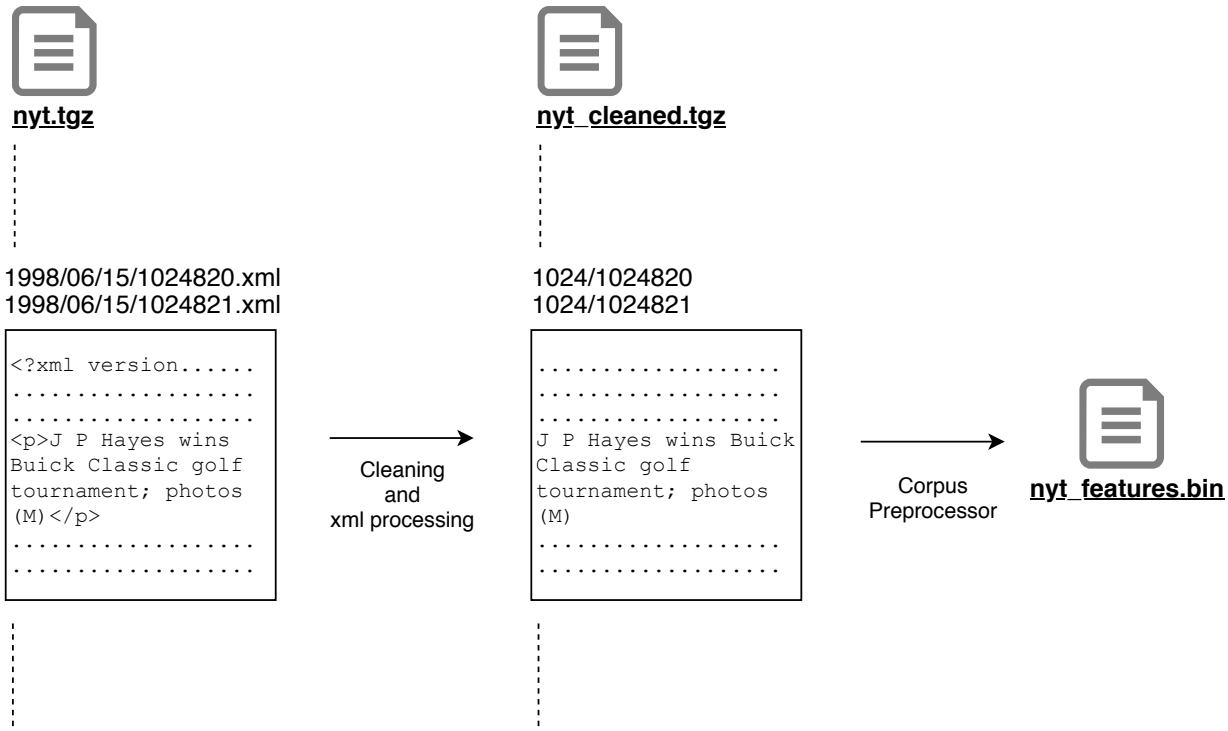


Figure 3.1: Preprocessing pipeline for the New York Times collection

The output binary file contains document frequency data followed by the document feature data. The binary file format is described in Figure 3.2.

The CAL system takes as input the corpus features (the binary output file of the corpus

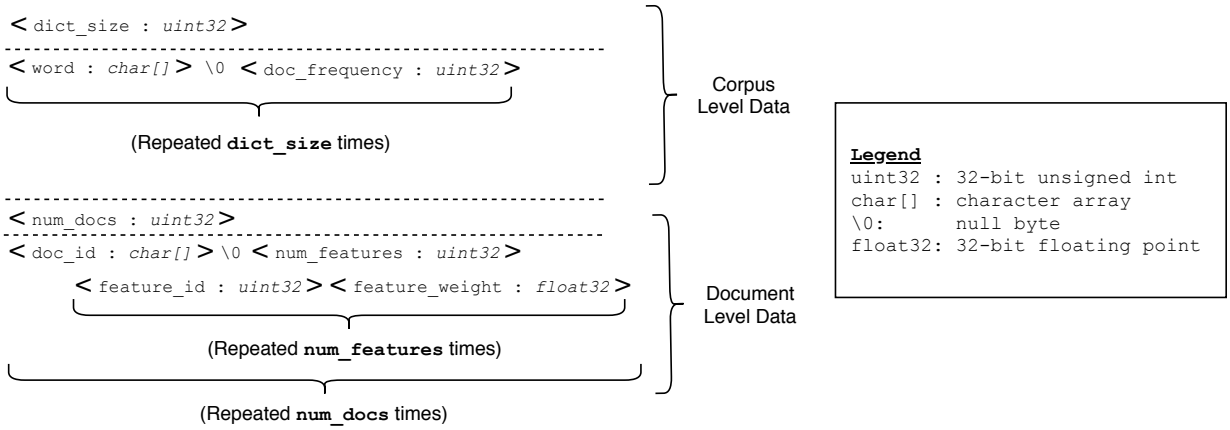


Figure 3.2: Description of the binary file outputted by the corpus preprocessor

preprocessor) and loads it in the memory. The document feature vectors are indexed and stored in the memory throughout the lifetime of a CAL process in order to ensure fast operations and reduce disk access. A review task is initiated by one or more seed queries (such as the topic description) provided by the user. The user can also specify seed document judgments and the choice of refresh strategy at the beginning of a task. Users can restart the review task from any checkpoint by just specifying the relevance judgments made until that time as the seed document judgments. The system handles all the review tasks concurrently, so that multiple users can use the system at the same time.

Training and scoring all the documents constitutes the majority of the computation. For training, we modified sofia-ml so that we can invoke its functions natively through our code and make it compatible with our data structures. We also stripped away unneeded parts from the sofia-ml source code. The weight vector obtained from training is used to fetch top documents from a set (depending on the refresh strategy). The score for a single document is obtained by computing the dot product of the document feature vector and the trained weight vector. The scoring of all the documents is parallelized across multiple threads (8 by default). By default, the code compiles with the `-O3` compiler optimisation flag, which contributes to significant reductions in the running time of various computations.

The CAL system is designed such that it is easy to extend or modify parts of the algorithm. Most of the refresh strategies mentioned in this thesis were implemented by extending a class and overriding few methods.

There are multiple ways to interact with the CAL system. The robust and user-friendly HTTP API should be used for most purposes. Python bindings for the CAL system is also

available (it is an abstraction over the HTTP API). The command line tool can be used for testing and simulation purposes. Interacting with CAL is documented in detail in the project's repository[4]. We also provide docker configuration which automatically installs all dependencies, configures the web server and spawn the required processes.

[4] https://github.com/HTAustin/HiCAL/blob/master/CALEngine/README.md

Chapter 4

Experimental Setup

4.1 Dataset

Our experiments were performed using multiple `athome` test collections.

The `athome1` and `athome4` collections were used in the TREC 2015 and 2016 Total Recall Track [17, 32], respectively. Both consist of 290,099 documents, which are redacted emails from Jeb Bush's eight year tenure as the Governor of Florida. `athome1` has 10 topics with an average 4398 documents labelled as relevant per topic. `athome4` has 34 topics with an average of 1059.44 documents labelled as relevant per topic. The topics for `athome1` and `athome4` are described in Table 4.1 and 4.4, respectively.

`athome2` and `athome3` collections were used in the TREC 2015 Total Recall Track [32] as well as in the TREC 2015 Dynamic Domain Track [39]. `athome2`, or the *Illicit Goods Collection*, consists of 465,147 web forum threads from BlackHatWorld[1] and Hack Forums[2]. It has 10 topics with an average of 2000.5 documents labelled as relevant per topic. `athome3`, or the *Local Politics Collection*, consists of 902,434 news article from various sources in United States and Canada. It has 10 topics with an average of 642.9 documents labelled as relevant per topic. The topics for `athome2` and `athome3` are described in Table 4.2 and 4.3, respectively.

The `athome` test collections are suitable for evaluating high recall systems due to the extensive judgments available for their topics. They were used in the Total Recall tracks,

[1] https://www.blackhatworld.com/
[2] https://hackforums.net/

which evaluated participating systems similar to how we evaluate different refresh strategies. Moreover, some of the `athome` collections were assessed (`athome1`) or re-assessed (`athome2` and `athome3`) using Continuous Active Learning based methods. These factors make the `athome` collections a suitable choice of dataset to conduct our experiments on.

4.2 Evaluation

We describe the performance metrics we used to compare the effectiveness and efficiency of refresh strategies in the following sections. For each refresh strategy, these metrics were computed at the dataset level by averaging the topic-level metric for each topic in that dataset. These dataset-level metrics were further averaged (macro average of topic-level metrics) to compute the average metric for a refresh strategy.

Effectiveness

To measure the effectiveness of various refresh strategies, we ran a CAL simulation for each topic and strategy. We can compare different strategies based on the recall values they achieved at various values of effort. For each topic, the recall is defined as

$$Recall = \frac{\text{No. of relevant documents found by the system}}{\text{Total no. of relevant documents in the corpus}} \tag{4.1}$$

Using absolute values of review effort can cause an imbalance among the recall values of individual topics. This is due to uneven number of relevant documents present across different topics (see Table 4.1-4.4). Instead of absolute effort, we use normalized effort E_{norm} for analyzing results. E_{norm} is calculated by dividing the absolute review effort by the total number of relevant documents.

The gain curve is a widely used method to evaluate high recall retrieval systems [16, 17, 32]. It is a plot of recall as a function of assessed documents. We use average recall gain curves to compare the effectiveness of different refresh strategies. A gain curve for a topic is a plot of recall (y-axis) against the normalized review effort ($E_{norm} = \frac{E}{R}$), where E is the number of judgments made since the beginning of the simulation and R is the total number of relevant documents for that topic. We get the dataset-level average gain curve by averaging the recall values over all its topics. We further average all the dataset-level recall values to obtain the average gain curve for a refresh strategy.

For the sake of readability, we also report certain points of interest from the gain curve in a tabular format. Specifically, we compare different refresh strategies based on their recall values when $E_{norm} \in \{1, 1.5, 2\}$. We also report the effort required to reach 75% recall.

To measure the statistical significance of performance difference between two strategies, we use the Student's paired t-test on a performance metric (such as recall at $E_{norm} \in \{1, 1.5, 2\}$). When comparing two refresh strategies, we perform statistical significance tests separately for each dataset.

Efficiency

A combination of experiment conditions (such as refresh strategy and its parameters) and a topic is referred as a "task". We use review effort as the stopping criteria for each task (step 11 of Algorithm 1). One judgment is equal to one unit of reviewer's effort. The CAL process for a task stops whenever the number of judgments processed by the system is equal to the maximum effort. The maximum normalized review effort (E_{norm}) in our efficiency experiments was set to 2.

To measure the computational efficiency, we record the running time of the simulation. Different refresh strategies are compared based on the running time of the simulation averaged over all the topics of a dataset. The dataset-level average running times are averaged to obtain the average running time of a strategy.

4.3 Runtime Environment

We used the command line interface of the CAL implementation discussed in Chapter 3 to run our experiments. We ran our experiments on a google cloud instance with 11 GB memory and twelve vCPUs (2.00 GHz Intel Xeon Processor). We ran a CAL process for each strategy and parameter setting sequentially. A CAL process simulated a maximum of twelve parallel tasks. For each task, we restricted the scoring of documents to a single thread.

4.4 Secondary Experiments

We performed few additional experiments to support certain intuitions behind the design of refresh strategies discussed in Section 5.3.1 and Section 5.4. The questions these

experiments were designed to answer are:

- If a judgment causes a refresh, a new review queue is populated. In the new queue, what is the position of the first unjudged document from the previous queue?
- To what extent does the classifier's notion of relevance change across various number of judgments?

The simulations were run across the 10 `athome1` topics and the reported metrics were averaged over them. The maximum review effort was set to 500.

Table 4.1: List of `athome1` topics

Topic ID	**Description**	**Relevant Documents**
athome100	School and Preschool Funding	4542
athome101	Judicial Selection	5836
athome102	Capital Punishment	1624
athome103	Manatee Protection	5725
athome104	New medical schools	227
athome105	Affirmative Action	3635
athome106	Terri Schiavo	17135
athome107	Tort Reform	2375
athome108	Manatee County	2375
athome109	Scarlet Letter Law	506

Table 4.2: List of `athome2` topics

Topic ID	Description	Relevant Documents
athome2052	paying for amazon book reviews	265
athome2108	CAPTCHA Services	661
athome2129	Facebook Accounts	589
athome2130	Surely bitcoins can be used	2299
athome2134	paypal accounts	252
athome2158	Using TOR for anonymous browsing on the internet	1256
athome2225	Rootkits	182
athome2322	Web Scraping	9517
athome2333	article spinner spinning	4805
athome2461	Offshore Host Sites	179

Table 4.3: List of `athome3` topics

Topic ID	Description	Relevant Documents
athome3089	pickton murders	255
athome3133	pacific gateway	113
athome3226	traffic enforcement cameras	2094
athome3290	rooster turkey chicken nuisance	26
athome3357	occupy vancouver	629
athome3378	rob mckenna gubernatorial candidate	66
athome3423	rob ford cut the waist	76
athome3431	kingston mills lock murders	1111
athome3481	fracking	2036
athome3484	paul and cathy lee martin	23

Table 4.4: List of `athome4` topics

Topic ID	Description	Relevant Documents
athome401	Summer Olympics	229
athome402	Space	647
athome403	Bottled Water	1091
athome404	Eminent Domain	548
athome405	ewt Gingrich	122
athome406	Felon Disenfranchisement	131
athome407	Faith-Based Initiatives	1587
athome408	Invasive Species	116
athome409	Climate Change	206
athome410	Condominiums	1354
athome411	"Stand Your Ground"	89
athome412	2000 Recount	1422
athome413	James V. Crosby	552
athome414	Medicaid Reform	841
athome415	George W. Bush	12106
athome416	Marketing	1452
athome417	Movie Gallery	5952
athome418	War Preparations	187
athome419	Lost Foster Child Rilya Wilson	1989
athome420	Billboards	742
athome421	Traffic Cameras	21
athome422	Non-Resident Aliens (NRA)	33
athome423	National Rifle Association (NRA)	286
athome424	Gulf Drilling	500
athome425	Civil Rights Act of 2003	718
athome426	Jeffrey Goldhagen	121
athome427	Slot Machines	246
athome428	New Stadiums and Arenas	466
athome429	Cuban Child, Elian Gonzales	828
athome430	Restraints and Helmets	999
athome431	Agency Credit Ratings	150
athome432	Gay Adoption	140
athome433	Abstinence	112
athome434	Bacardi Trademark Lobbying	38

Chapter 5

Refresh Strategies

In this chapter, we discuss some of the refresh strategies we investigated. Along with the description of the strategies, we also describe the motivation which led us to designing them, and their potential impact on the behaviour of the CAL algorithm, in terms of effectiveness and efficiency. When applicable, we also report results of supporting experiments which can help gain more insight into some refresh strategies.

5.1 Basic Concepts

Before discussing the refresh strategies, it is useful to define a few basic concepts.

Effectiveness

Effectiveness is the measure of utility the CAL system provides to its users. A more effective system would let users find more relevant documents with less review effort. Chapter 4 defines in detail how we measure the effectiveness.

Efficiency

We use the term efficiency to indicate the computation costs of the CAL system. While memory costs should also be a factor when evaluating efficiency, we are mostly concerned with the CPU costs and running times in this thesis.

Responsiveness

Responsiveness of a CAL system becomes important in the context of live and interactive systems. In a real world application, system delays in delivering documents for review can be detrimental to user experience and waste valuable assessor time. A responsive system should minimize all such delays without harming its effectiveness. One way to improve responsiveness is through efficiency. Other ways include utilizing the idle time when the reviewer is reading the document.

Full Refresh

We use the term *full refresh* to denote a refresh in which all available judgments are used in training and relevance likelihood scores for all the documents are calculated.

A full refresh runs in $O(t + n \log n)$ time where n is the number of documents in the corpus and t is the number of training iterations. The number of iterations t required for convergence depends on the size of the training data (or number of judgments). We set t to a high constant value (100000) for our dataset. Although the length of documents (specifically, the number of non-zero features in a document feature vector) affect the running time of training and scoring, we treat it as a constant to simplify our analysis. Scoring all the documents takes $O(n)$ time and sorting them takes $O(n \log n)$ time. In most cases, only top k documents (where $k << n$) are needed. In such cases, complete sorting is not required and the refresh can be performed in $O(t + n \log k)$ time.

5.2 Exponential Batch Refresh Strategy

In the original BMI AutoTAR, full refreshes are performed after receiving a batch of judgments. The size of this batch increases exponentially with number of refreshes. The batch size is initially set to $k = 1$ and after every refresh, is updated using

$$k \leftarrow k + \lfloor \frac{k+9}{10} \rfloor$$

The smaller batch size during the beginning of a task results in frequent refreshes and thus allows the classifier to frequently update its understanding of relevance. This strategy scales well with the number of judgments (E) made during the CAL process since only $O(\log E)$ number of refreshes are done. According to Cormack and Grossman [6], the

motivation behind this strategy was to "reap the benefits of early precision, while avoiding downside risk and excessive running time, by using exponentially increasing batch size".

5.3 Static Batch

In *static batch refresh strategy*, full refreshes are performed after a fixed number of judgments are received. When batch size is fixed to 1, a full refresh happens after every judgment. The only parameter in this strategy is the judgment batch size.

This strategy incurs a high computation cost and introduces scalability issues since it requires $O(E)$ number of refreshes and each refresh takes $\Omega(n)$ time, where E is the number of documents judged during the CAL process and n is the number of documents in the dataset.

5.3.1 Responsiveness

If a judgment triggers a refresh, the system waits for the refresh to complete before returning a fresh set of documents for the user to judge. For very small batch sizes (such as 1) and large values of n, full refreshes will be frequent and expensive. Pauses as small as half a second after every few judgments can disrupt the user experience.

One way to address this problem is to perform asynchronous refreshes and immediately show the users documents from the old review queue [1]. During a refresh, we fill the review queue with a few extra top documents in addition to the batch size. Whenever a judgment triggers a refresh, we delegate the refreshing to a background process and continue serving the user the aforementioned extra documents from the review queue. Meanwhile, when the refresh in the background is finished, the review queue is replaced with a newer version. This modification delays the effect of user feedback on the review queue by $\lceil \frac{t_r}{t_u} \rceil$ documents, where t_r is the time it takes to complete a refresh, and t_u is the time a user takes to review one document.

UWaterlooMDS in TREC 2017 Core Track [41] used CAL to power their manual review tool. They used asynchronous static batch strategy with batch size of 1. The tool presented summaries in addition to the full document to the reviewers. The authors and a group of graduate students used the review tool to judge numerous documents in the New York Times collection across 250 topics. The average and median time spent on a judgment (t_u) were 13 and 4.1 seconds respectively. We simulated their judgments using our system

and the average refresh time (t_r) was 1.71 seconds. For both the average and median user judgment times, the effect of user feedback would have been delayed by 1 document ($\lceil \frac{1.71}{13} \rceil$ and $\lceil \frac{1.71}{4.1} \rceil$).

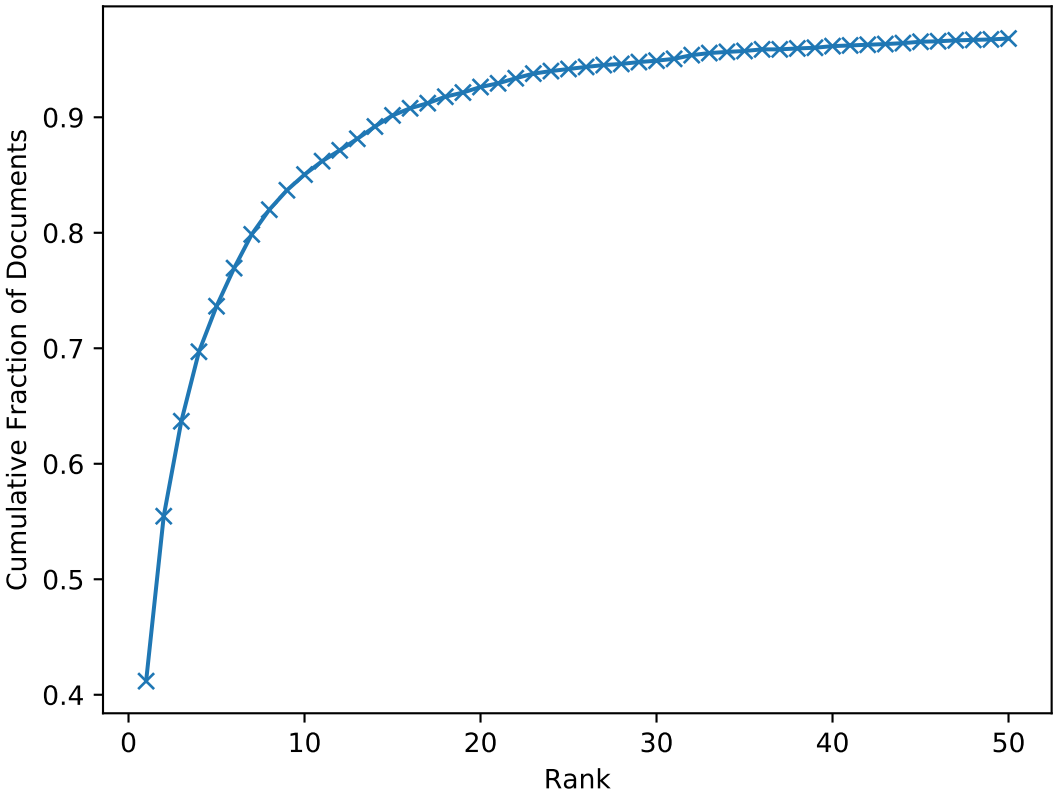


Figure 5.1: Avg. fraction of currently second-ranked documents vs the rank below which they were placed after the background refresh

When the effect of user feedback is delayed by 1 document, the assessor ends up judging the top two documents from the current review queue before the new review queue is ready. To estimate the quality of the second document showed to the user, we obtained the rank of that second document in the next review queue (the review queue populated by the background refresh). The setup for this experiment is described in Section 4.4. We found that 41.2% of these documents were top ranked after the background refresh. Moreover, 85% and 96.8% of these documents were ranked top 10 and top 50 respectively, after the

background refresh. Figure 5.1 plots the percentage of documents which were within a given rank after the background refresh.

If the CAL system can perform two refreshes while the user is reading a document (i.e. $2t_r \leq t_u$), we can achieve a responsive system without any delay in processing of judgments. If the user is reading a document and his judgment will trigger a refresh, a background process can just perform two refreshes; one assuming the upcoming judgment will be relevant, and another assuming it will be non-relevant. The two possible future review queues are therefore ready while the user is reading the document and the correct one can be immediately served once the user judgment is received. For the numbers provided in the previous paragraph, the constraint $2t_r \leq t_u$ hold true.

5.4 Partial Refresh

In this strategy, a full refresh is performed after every fixed number of judgments, similar to the *static batch strategy*. At the end of each full refresh, a small set of documents with the highest relevance likelihood scores are stored in a *partial refresh set*. After every judgment, a *partial refresh* is performed. During a partial refresh, all available judgments are used in training but relevance likelihood scores are only calculated for the documents in the partial refresh set. A single partial refresh runs in $O(s \log s)$ average time, where s is the size of the *partial refresh set*. The logarithmic factor in running time is a result of the partial set stored as a binary search tree and after every judgment, it takes $O(\log(s))$ time to remove a document from that set. Since the scores for the document in this set are recomputed after every judgment, only the document with maximum score is returned to the user. Figure 5.2 demonstrates the CAL relevance feedback loop with partial refresh strategy.

Partial refresh strategy has two parameters:

- k: number of judgments between two full refreshes
- s: size of the partial refresh set ($s \geq k$)

Our motivation behind designing this strategy was to efficiently replicate the behaviour of *static batch strategy* (batch size = 1). We hypothesized that the classifier's notion of relevance does not change dramatically between two nearby judgments and we can safely avoid computing scores for low ranked documents. To get some evidence, we constructed an experiment (the setup for this experiment is explained in Section 4.4). After each

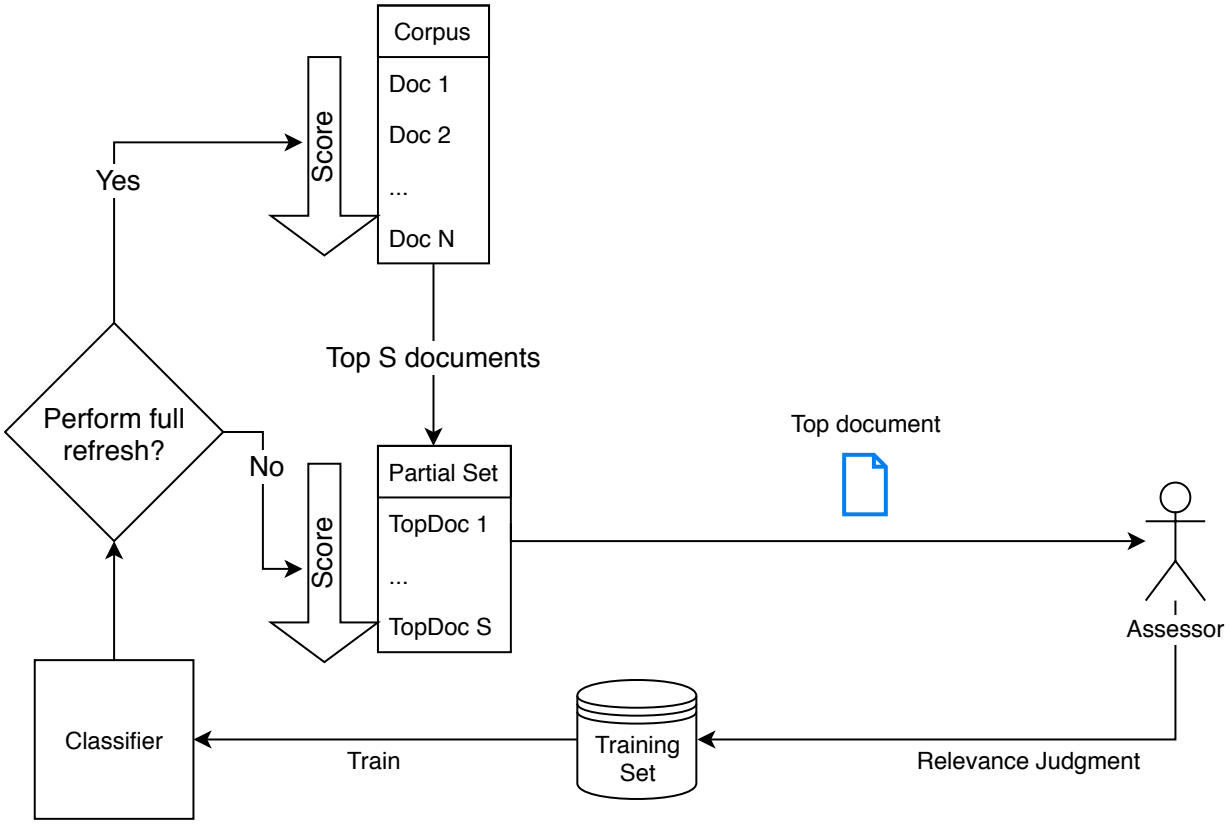


Figure 5.2: Partial Refresh Strategy

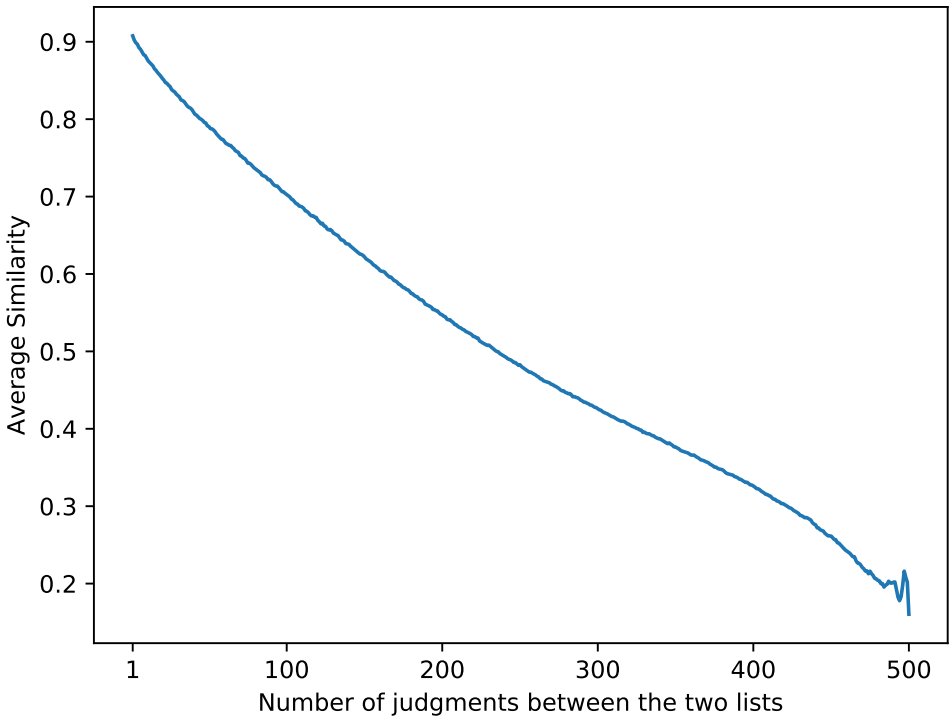


Figure 5.3: Average similarity between two ranked lists (truncated at 1000) separated by various number of judgments. Similarity between two lists is computed as the fraction of common documents between them.

judgment, we stored the list of top 1000 unjudged documents scored by the classifier. Since we employed the static batch strategy with batch size of 1, a refresh occurred after every judgment. Similarity between two lists was computed as the fraction of documents present in both lists. Figure 5.3 reports the average similarity of these document lists separated by various number of judgments. 90.8% of the top documents remain the same across consecutive judgments (i.e., separated by 1 judgment). For the any two lists separated by 100 judgments, there were 70.3% common documents in the top 1000. The results of the above experiment can also be qualitatively analysed to determine the parameters for partial refresh strategy. For example, based on our results, setting partial set size to 1000 and full refreshing every 100 judgments looks like a reasonable starting point.

With some enhancements, this strategy can also help reduce the memory costs when

working with low physical memory or very large datasets (such as ClueWeb). As mentioned in Section 3.2.2, the documents are loaded in memory to enable faster operations and improve the responsiveness of the system. Partial refreshes are faster than full refreshes [1] and performed on a small set of data which can be stored in the memory. Full refresh can be performed in the background, and can thus afford reads from the disk without sacrificing the user experience or effectiveness of this strategy.

5.5 Precision Based Refreshing

The previous strategies we discussed use the elapsed number of judgments as a criteria to determine when to refresh. Instead of number of judgments, we can perform a full refresh when the "output quality" of the CAL system falls below some threshold. The output quality of a CAL system is considered high if the user judges more documents as relevant. There could be various ways to concretely define "output quality". Our motivation behind designing this strategy was to investigate a more meaningful criterion for triggering a refresh. Figure 5.4 demonstrates the CAL relevance feedback loop with precision based refreshing.

In *precision based refreshing*, we work with a very simple definition of "output quality". It is the fraction of relevant judgments in some fixed number of latest judgments made by the reviewer. A full refresh is performed whenever this fraction falls below some threshold. There are two parameters in this strategy:

- m: number of recent judgments to compute precision on
- p: the threshold below which a full refresh is triggered

Our aim is to find more meaningful factors which can help us better understand the effectiveness of various refresh strategies, and as a result, help us design better refresh strategies. For example, in certain cases, it is desirable to save computation by not refreshing when the output quality is high and force more frequent refreshes when the output quality is low. However, during later stages of a task when the output quality is always low and the likelihood of finding relevant documents is very low, *precision based refreshing* behaves similar to *static batch refreshing* with batch size of 1.

[1] In our experiments with `athome1`, scoring a partial set of 1000 documents and scoring the entire collection took on average of 148ms and 1ms, respectively.

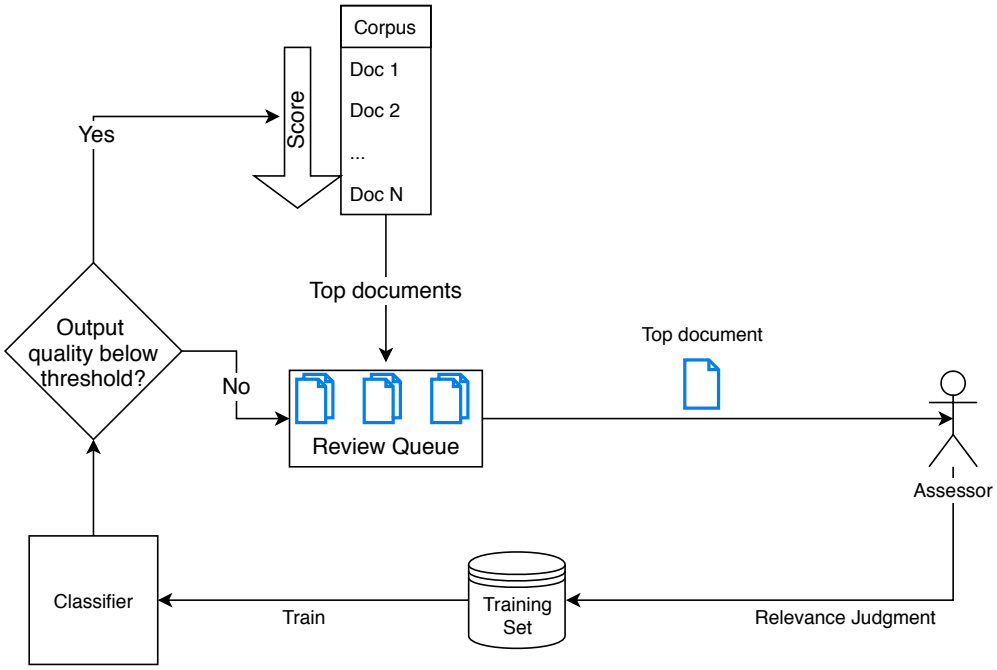


Figure 5.4: Precision Based Refreshing

5.6 Recency Weighting Strategy

This strategy modifies the training step (step 4 in Algorithm 1) in the CAL process by favoring documents which were recently judged. As described in Section 2.2, training is done over several iterations. In each iteration of the original training, a relevant and a non-relevant document is randomly sampled from the training set. The loss computed using them is used to update the classifier weights. To incorporate recency weighting, we modified the uniform random sampling such that the probability of selecting a document increases if it was judged recently.

Given a list of documents $[D_1, D_2, ..., D_n]$ ordered by the time they were judged, our modified random sampling will select a document D_x with probability $P(D_x)$ where

$$P(D_x) = P(D_1) + \frac{P(D_1)(x-1)(w-1)}{N-1}$$

Therefore, $P(D_x)$ is a linear function such that the latest judged document D_n is w times more likely to be selected than the oldest judged document D_1. A full refresh (with

modified training) is performed after every judgment. The only parameter for this strategy is w.

Implementing the modified random sampling by computing the individual probabilities of documents as shown in the above expression is inefficient. The probabilities will have to be computed for all the documents in the training set before every refresh. To achieve minimum overhead from the modified sampling, we use a transformation on the uniform random number generator to achieve our desired distribution. In the next subsection, we cover the implementation of our modified sampling in further detail.

5.6.1 Weighted Random Sampling

Our objective is to develop an efficient weighted random sampling method for the recency weighting strategy. Given an integer N, we want to generate a random integer between 0 and N. Additionally, we want each integer to have a linearly increasing chance of getting sampled, and N being w times more likely to be sampled than 0. The expression below describes the probability of an integer x being sampled

$$P(x) = P(0) + \frac{P(0)(x)(w-1)}{N} \tag{5.1}$$

Rejection Sampling is a method to design such random number generators, but they require the probabilities of each integer to be precomputed. We can instead apply Inverse Transformation Sampling [13] to a uniform random number generator to efficiently generate our desired distribution. Given a uniform random variable U in $[0, N]$ and a cumulative distribution function F over $[0, N]$, $F^{-1}(U)$ has our desired distribution (which is F).

Equation 5.1 is also valid for any real number $x \in [0, N]$. Substituting $p = P(0)$ and $m = \frac{p(w-1)}{N}$, we get $P(x) = p + mx$. The cumulative distribution function for $P(x)$ is

$$\begin{aligned} F(x) &= \int_0^x P(x)dx \\ &= \frac{mx^2}{2} + px \end{aligned} \tag{5.2}$$

Solving for the inverse, $u \in U$

$$\begin{aligned} F(F^{-1}(u)) &= u \\ \frac{mF^{-1}(u)^2}{2} + pF^{-1}(u) - x &= 0 \end{aligned} \tag{5.3}$$

To obtain the target random number $F^{-1}(u)$, we can solve the quadratic equation 5.3. We implemented[2] the sampling in recency weighting strategy using the above method.

5.6.2 Number of Training Iterations

Scoring can be made efficient by using strategies like *partial refreshing* or simply performing it over multiple processes. The choice of training methods in the CAL algorithm makes it difficult to parallelize. There are alternative training methods which are designed for efficiency and scalability [2, 30], but that is a direction for future work. A parameter in our training algorithm which can be easily controlled and has a significant impact on performance is the number of training iterations. Reducing the training iterations by some factor dramatically improves running time by sacrificing the effectiveness of the system.

Our initial motivation behind *recency weighting* was to improve the default training by prioritizing newer judgments. However, we found no difference in effectiveness of CAL between the original and modified training. We shifted our goal towards lowering the number of training iterations while maintaining quality using recency weighting.

[2] https://github.com/HTAustin/HiCAL/blob/426955e7f1c88e450b05d3c8d42d58544f864b51/CALEngine/src/classifier.h#L37

Table 5.1: List of the refresh strategies and their parameters.

Strategy	**Parameters**
`exponential`	None
`static`	k = *no. of judgments between full refreshes*
`partial`	k = *no. of judgments between full refreshes* s = *no. of documents in the partial refresh set*
`precision`	m = *no. of recent judgments to compute precision on* p = *full refresh is triggered if the precision of last* m *documents fall below this value*
`recency`	w = *factor by which the latest judged document is more likely to be sampled than the oldest judged document*
`*`	it = *no. of training iterations* *(global parameter,* 100000 *unless specified)*

Chapter 6

Results and Discussion

In this chapter, we dive into the results of our experiments and compare all the refresh strategies.

For sake of readability, we encode each strategy with their parameter settings as `strategy_name(param1=x,...)`. For reference, Table 5.1 lists all the strategies and their parameters. We summarize the average results obtained across all the datasets for different parameter settings of `exponential`, `static`, `partial` and `precision` in Table 6.1. Table 6.3, 6.4, 6.5 and 6.6 reports the dataset-level results obtained when using `athome1`, `athome2`, `athome3` and `athome4` , respectively. In these tables, we report the recall achieved at certain values of effort, effort required to achieve 75% recall, and the average running time. Instead of absolute effort, we use normalized effort E_{norm} as defined in Section 4.2. For example, "Avg. recall@($E_{norm} = 1.5$)" refers to the average recall achieved across all the topics when $1.5 \times R$ documents haven been assessed, where R is the total number of relevant documents for a topic.

Figure 6.1 shows the effectiveness vs efficiency scatter plot for selected refresh strategies. In this plot, we use "effort required to achieve 75% recall" as the effectiveness measure, and "simulation running time until 75% recall is achieved" as the efficiency measure. Strategies placed lower are more effective and the ones placed towards the right are more efficient.

Static Batch Refresh Strategy

Figure 6.2a compares the gain curves for `exponential`, `static(k=1)` and `static(k=100)`.

With `static(k=1)`, CAL achieves noticeably higher recall of 0.754 at $E_{norm} = 1$ than `exponential` which achieves 0.723 recall. `static(k=100)` performs worse than

Table 6.1: Summary of results for `bmi_refresh`, `static_batch`, `partial_refresh` and `precision_strategy`

Strategy	Avg. Recall @(E_{norm}=1)	Avg. Recall @(E_{norm}=1.5)	Avg. Recall @(E_{norm}=2)	E_{norm} for 75% recall	Running Time (in min)
exponential	0.723	0.832	0.887	2.126	0.33
static(k=1)	0.754	0.856	0.900	1.962	34.19
static(k=100)	0.652	0.769	0.840	2.427	0.34
partial(k=10,s=1000)	0.753	0.855	0.899	1.961	18.22
partial(k=100,s=500)	0.731	0.839	0.893	1.998	16.50
partial(k=100,s=1000)	0.737	0.844	0.893	1.994	16.57
partial(k=100,s=5000)	0.747	0.852	0.896	1.948	17.34
partial(k=500,s=1000)	0.686	0.770	0.804	2.723	16.09
precision(p=0.4,m=25)	0.679	0.844	0.892	2.074	17.63
precision(p=0.6,m=25)	0.728	0.853	0.897	2.060	20.22
precision(p=0.8,m=25)	0.749	0.855	0.900	1.991	20.24
precision(p=1.0,m=25)	0.755	0.855	0.900	1.957	21.02

`exponential`, managing to achieve 0.652 recall at the same effort. These results establish that frequent refreshing improves the effectiveness of a CAL system. Although the batch sizes in BMI increases exponentially with time, it still does frequent refreshes during the early stages of the CAL process, thus performing better than `static(k = 100)`. `exponential` is also extremely cheap in terms of total computation cost since it only performs a logarithmic number of refreshes compared to the `static` strategies. The `exponential` simulation finished in less than a minute while `static(k=1)` took little more than a half hour.

Using Student's paired t-test, we found that `static(k=1)` was significantly better than `exponential` in terms of recall achieved at $E_{norm} \in \{1, 1.5\}$ ($p < 0.05$ for `athome1` and `athome3`, $p < 0.0001$ for `athome4`). However, for the `athome2` collection, the improvement was negligible. When $E_{norm} = 2$, the difference in recall achieved by both the strategies are insignificant for `athome1` and `athome3` ($p > 0.1$). As evident from the gain curve in Figure 6.2a, the recall values at $E_{norm} = 2$ plateaus at a high value for these strategies.

We evaluate rest of the refresh strategies by comparing them to `static(k = 1)`.

Partial Refresh Strategy

Figure 6.2b compares the gain curves for `static(k=1)`, `partial(k=100,s=1000)` and `partial(k=500,s=1000)`.

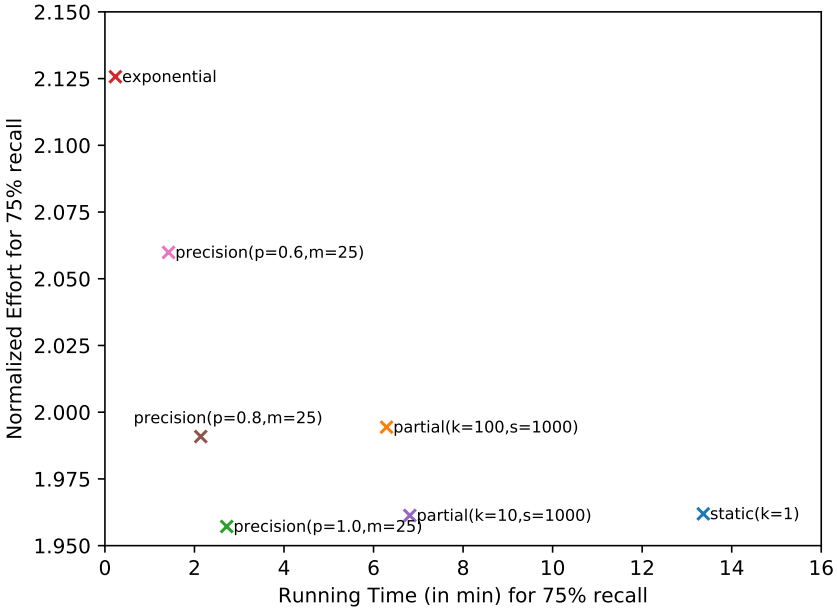


Figure 6.1: Effectiveness vs Efficiency plot for selected refresh strategies

By fixing the partial set size `s=1000` and varying the full refresh period `k` in `partial`, we observe that for `k=10` and `k=100`, the difference in recall remains insignificant throughout the CAL process. Their recall values are also very similar to `static(k=1)`. They achieve 0.855 and 0.844 recall, respectively at $E_{norm} = 1.5$. For `k=500`, we observe 0.770 recall at the same effort, which is worse than `exponential` (0.832). This is in agreement with our previous observation that more frequent full refreshes increases CAL's effectiveness. `static(k=100)` consistently achieved lower recall when compared to `static(k=1)` while `partial(k=100,s=1000)` is much closer to the latter. Based on this, it can be established that partial refreshing contributes noticeable improvements to recall over plain static batch refreshing with same frequency of full refresh.

Figure 6.2c shows the effect of varying partial set size `s` when the full refresh period `k` is fixed to 100. We observe no changes to the recall values when the size of partial set is 500 or higher. At $s = 100$, the decrease in performance is noticeable.

`partial` simulations on average ran 50.44% faster than `static(k=1)`. While achieving similar effectiveness as `static(k=1)`, `partial(k=10,s=1000)` ran 46.71% faster.

`partial(k=10,s=1000)` was the optimal setting across all our experiments. For any dataset, there was no significant difference ($p > 0.1$) between the recall achieved by

`partial(k=10,s=1000)` and `static(k=1)` at $E_{norm} \in \{1, 1.5, 2\}$. For `athome1`, even `partial(k=100,s=1000)` was an optimal setting as there was no significant difference ($p > 0.45$) between the recall achieved by `partial(k=100,s=1000)` and `partial(k=10,s=1000)` at $E_{norm} \in \{1, 1.5, 2\}$.

Precision Based Refreshing

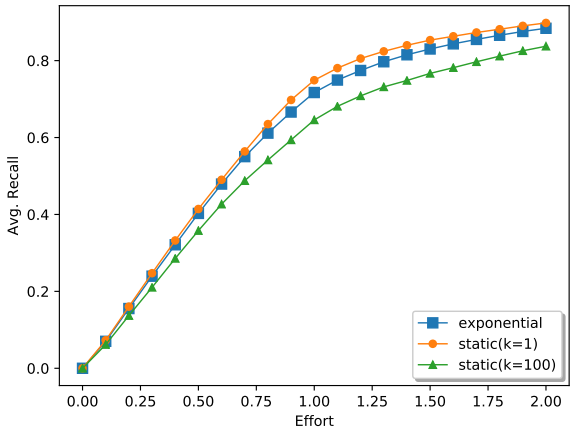
Figure 6.2d compares the gain curves for various settings of `precision`.

In this strategy, we fixed `m=25` and varied `p`. For `p=0.8` and `p=1.0`, `precision` achieves 0.75 recall at $E_{norm} = 1$ which is similar to `static(k=1)`. This similarity of recall is also seen at $E_{norm} = 1.5$ and $E_{norm} = 2$. For `precision(p=1.0)`, CAL refreshes whenever a non-relevant judgment is made, thus behaving very similar to `static(k=1)`. For smaller values of `p`, we observe lower recall values during the initial stages; 0.728 and 0.679 recall at $E_{norm} = 1$ for `precision(p=0.6)` and `precision(p=0.4)`, respectively. However, they catch up to `static(k=1)` at higher E_{norm}, as relevant documents become rarer; 0.853 and 0.844 recall at $E_{norm} = 1.5$ for `precision(p=0.6)` and `precision(p=0.4)`, respectively.

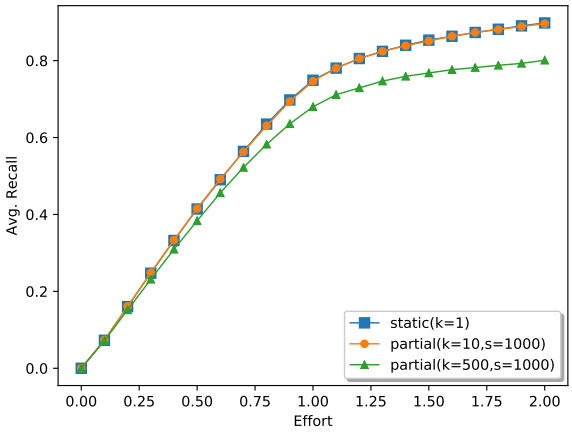
This strategy improved the running time of simulations by 42.15% on average when compared to `static(k=1)`. `precision` strategies triggers lower number of refreshes during the beginning of the CAL process when relevant documents are easier to find. During the later stages when the relevant documents are harder to find, `precision` strategies tend to keep refreshing after every judgment. Figure 6.3 compares the `precision` strategies with `static(k=1)` based on how many refreshes were performed to achieve a certain recall. The `precision` strategies find a lot of relevant documents initially since they are easy to find and the precision of CAL's output is high. `static(k=1)` keeps refreshing steadily irrespective of the output quality. As relevant documents become harder to find, we find that the `precision` strategies start refreshing as frequently as `static(k=1)`.

Effect of Training Iterations

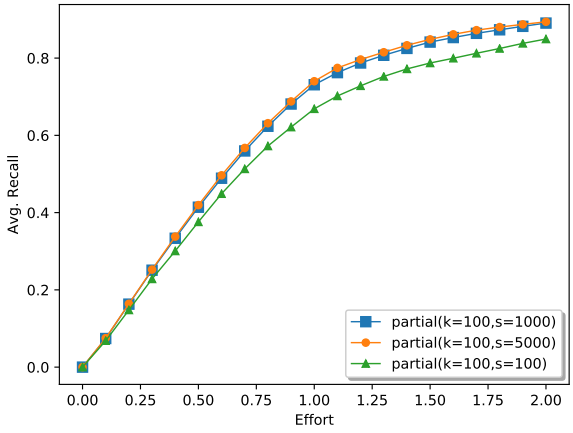
In all the previously discussed strategies, the improvement in running time over `static(k=1)` is either due to refreshing less often (`exponential`, `precision`) or by optimizing scoring during a refresh (`partial`). The running time of scoring can be further improved by just increasing the number of threads. However, training is another expensive step which cannot be trivially distributed across multiple threads. One way to dramatically reduce the



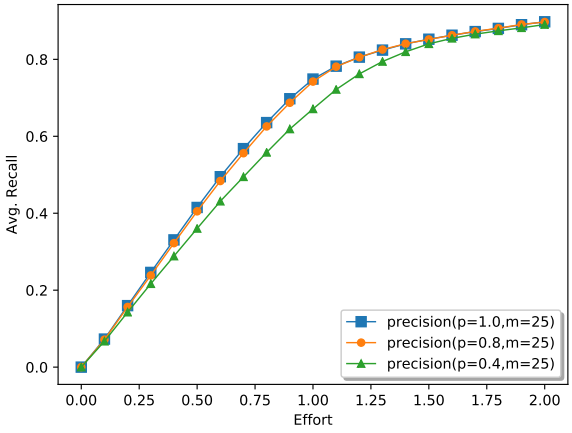
(a) Comparison of `exponential` and `static`. `static(k=1)` consistently outperforms the rest.



(b) Comparison of `partial` and `static(k=1)`. We fix the partial set size to 1000 and observe that `partial(k=10,s=1000)` performs very similar to `static(k=1)`.



(c) Effect of partial set size on effectiveness. We observe that increasing the partial set size beyond 1000 doesn't have any benefit.



(d) Comparison of various settings in `precision`.

Figure 6.2: Comparison of various refresh strategies

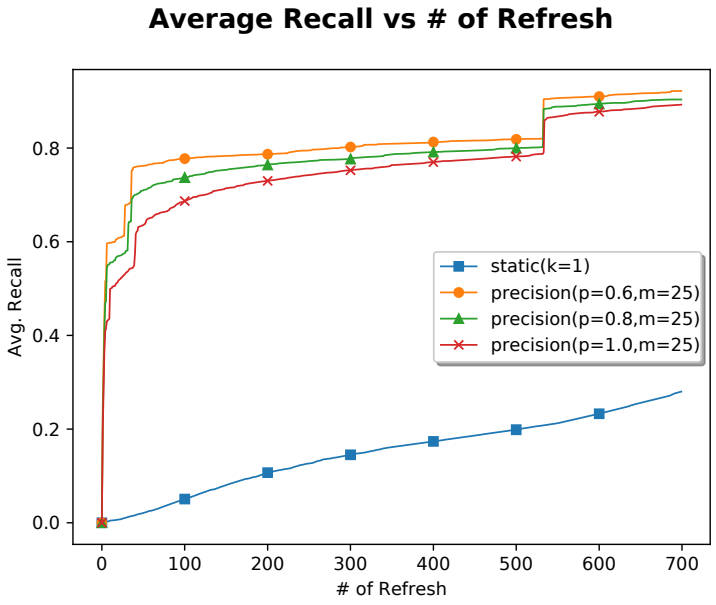


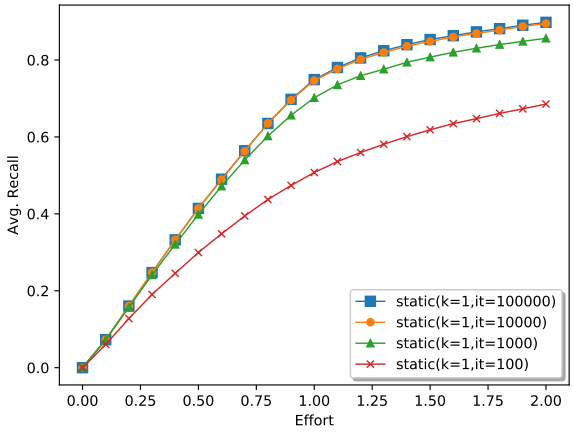
Figure 6.3: Number of refreshes required to achieve a certain recall – `athome1`

Table 6.2: Summary of results for recency weighting

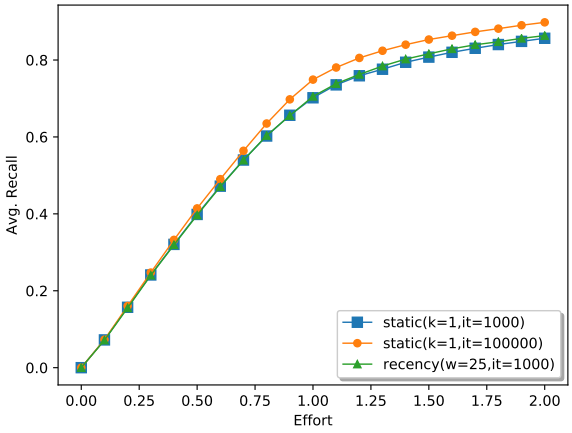
Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
`recency(w=1,it=1000)`	0.708	0.810	0.859	2.471	15.51
`recency(w=5,it=1000)`	0.715	0.817	0.865	2.449	16.73
`recency(w=10,it=1000)`	0.713	0.817	0.867	2.435	15.99
`recency(w=25,it=1000)`	0.712	0.819	0.866	2.470	16.05

training time is by simply reducing the number of training iterations (`it`). In our experiments, the average time to train a classifier for 100000 (default), 10000, and 1000 number of iterations was 265ms, 33ms, and 4ms respectively.

Despite these massive improvements in the running time, reducing the number of iterations `it` beyond a certain point will directly reduces the quality of the classifier, and thus harming the effectiveness of CAL. Figure 6.4a shows the impact of `it` on the gain curves of `static(k=1)`. We find that reducing `it` from 100000 to 10000 didn't do any noticeable affect on the recall. This suggests that setting `it=10000` for all our experiments might have been enough for the classifier to converge. Beyond this, we observe noticeable loss in performance.



(a) Effect of number of training iterations on recall



(b) Effect of recency weighting on settings with low number of training iterations

Figure 6.4: Gain curves comparing the effect of training iterations and recency weighting

Recency Weighting

During our initial experiments, recency weighting seemed to have no impact on the recall. By reducing the number of training iterations to 1000, we introduced significant degradation in the system's effectiveness. Reducing the number of training iterations also reduced the running time of simulation by 54.64% when compared to `static(k=1)`. We used recency weighting to recover the lost effectiveness. `recency(w=1,it=1000)` is equivalent to `static(k=1,it=1000)` and it achieves 0.708 and 0.810 recall when E_{norm} is equal to 1 and 1.5 respectively. For $w > 1$, we observe a negligible increase in recall. Moreover, the recall is consistently lower when compared to `static(k=1,it=100000)`. For example, `recency(w=10,it=1000)` is only able to achieve 0.817 recall at $E_{norm} = 1.5$, while `static(k=1,it=100000)` achieves 0.856 recall at the same effort. Table 6.2 reports the effectiveness and efficiency measures averaged over all the datasets. Table 6.7-6.10 reports the effectiveness and efficiency measures for individual datasets.

Table 6.3: Summary of results for `bmi_refresh`, `static_batch`, `partial_refresh` and `precision_strategy` − `athome1` collection

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
exponential	0.718	0.826	0.904	1.127	0.30
static(k=1)	0.751	0.865	0.925	1.020	59.59
static(k=100)	0.705	0.806	0.887	1.169	0.59
partial(k=10,s=1000)	0.753	0.863	0.926	1.008	38.72
partial(k=100,s=500)	0.753	0.855	0.923	1.022	36.62
partial(k=100,s=1000)	0.754	0.856	0.922	1.013	36.52
partial(k=100,s=5000)	0.756	0.855	0.921	1.015	37.19
partial(k=500,s=1000)	0.701	0.785	0.815	1.323	35.77
precision(p=0.4,m=25)	0.700	0.849	0.917	1.127	35.16
precision(p=0.6,m=25)	0.735	0.860	0.923	1.064	39.44
precision(p=0.8,m=25)	0.749	0.862	0.927	1.029	35.52
precision(p=1.0,m=25)	0.753	0.864	0.927	1.009	36.96

Table 6.4: Summary of results for `bmi_refresh`, `static_batch`, `partial_refresh` and `precision_strategy` − `athome2` collection

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
exponential	0.679	0.801	0.861	1.417	0.23
static(k=1)	0.685	0.809	0.862	1.403	28.88
static(k=100)	0.637	0.778	0.848	1.533	0.29
partial(k=10,s=1000)	0.686	0.809	0.862	1.405	12.08
partial(k=100,s=500)	0.669	0.806	0.861	1.429	9.74
partial(k=100,s=1000)	0.681	0.808	0.861	1.417	10.39
partial(k=100,s=5000)	0.686	0.810	0.861	1.402	12.10
partial(k=500,s=1000)	0.614	0.726	0.784	1.704	9.69
precision(p=0.4,m=25)	0.657	0.807	0.861	1.425	12.64
precision(p=0.6,m=25)	0.683	0.809	0.863	1.403	15.02
precision(p=0.8,m=25)	0.690	0.811	0.862	1.390	16.65
precision(p=1.0,m=25)	0.690	0.810	0.861	1.388	17.41

Table 6.5: Summary of results for `bmi_refresh`, `static_batch`, `partial_refresh` and `precision_strategy` − `athome3` collection

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
exponential	0.783	0.862	0.887	4.584	0.60
static(k=1)	0.827	0.878	0.899	4.244	32.45
static(k=100)	0.614	0.719	0.789	5.264	0.33
partial(k=10,s=1000)	0.818	0.878	0.895	4.247	11.99
partial(k=100,s=500)	0.769	0.848	0.889	4.286	10.51
partial(k=100,s=1000)	0.776	0.858	0.891	4.302	9.86
partial(k=100,s=5000)	0.800	0.877	0.899	4.135	10.72
partial(k=500,s=1000)	0.742	0.783	0.800	5.363	9.44
precision(p=0.4,m=25)	0.675	0.869	0.888	4.434	16.24
precision(p=0.6,m=25)	0.774	0.875	0.894	4.523	18.32
precision(p=0.8,m=25)	0.813	0.877	0.899	4.341	19.44
precision(p=1.0,m=25)	0.831	0.878	0.902	4.241	20.12

Table 6.6: Summary of results for `bmi_refresh`, `static_batch`, `partial_refresh` and `precision_strategy` − `athome4` collection

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
exponential	0.713	0.839	0.894	1.375	0.19
static(k=1)	0.754	0.872	0.913	1.181	15.85
static(k=100)	0.651	0.772	0.837	1.743	0.15
partial(k=10,s=1000)	0.754	0.871	0.913	1.185	10.07
partial(k=100,s=500)	0.733	0.847	0.898	1.254	9.16
partial(k=100,s=1000)	0.739	0.855	0.899	1.247	9.50
partial(k=100,s=5000)	0.746	0.864	0.903	1.239	9.34
partial(k=500,s=1000)	0.687	0.785	0.818	2.501	9.47
precision(p=0.4,m=25)	0.684	0.850	0.902	1.309	6.48
precision(p=0.6,m=25)	0.721	0.867	0.909	1.250	8.09
precision(p=0.8,m=25)	0.745	0.870	0.910	1.204	9.36
precision(p=1.0,m=25)	0.747	0.869	0.909	1.191	9.60

Table 6.7: Summary of results for recency weighting − `athome1`

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
recency(w=1,it=1000)	0.704	0.797	0.875	1.257	22.07
recency(w=5,it=1000)	0.705	0.813	0.887	1.191	26.71
recency(w=10,it=1000)	0.708	0.824	0.891	1.206	23.30
recency(w=25,it=1000)	0.708	0.827	0.893	1.193	23.15

Table 6.8: Summary of results for recency weighting – `athome2`

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
`recency(w=1,it=1000)`	0.630	0.749	0.806	1.829	15.36
`recency(w=5,it=1000)`	0.631	0.751	0.810	1.767	15.70
`recency(w=10,it=1000)`	0.633	0.751	0.812	1.745	15.46
`recency(w=25,it=1000)`	0.633	0.752	0.809	1.780	15.97

Table 6.9: Summary of results for recency weighting – `athome3`

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
`recency(w=1,it=1000)`	0.791	0.866	0.875	5.291	19.00
`recency(w=5,it=1000)`	0.808	0.869	0.880	5.413	18.87
`recency(w=10,it=1000)`	0.799	0.862	0.881	5.414	19.54
`recency(w=25,it=1000)`	0.800	0.864	0.879	5.426	19.52

Table 6.10: Summary of results for recency weighting – `athome4`

Strategy	**Avg. Recall @(E_{norm}=1)**	**Avg. Recall @(E_{norm}=1.5)**	**Avg. Recall @(E_{norm}=2)**	**E_{norm} for 75% recall**	**Running Time (in min)**
`recency(w=1,it=1000)`	0.709	0.830	0.881	1.508	5.59
`recency(w=5,it=1000)`	0.715	0.834	0.883	1.424	5.63
`recency(w=10,it=1000)`	0.711	0.831	0.885	1.375	5.65
`recency(w=25,it=1000)`	0.707	0.834	0.882	1.479	5.58

Chapter 7

Conclusion

The work described in this thesis is an attempt towards improving the AutoTAR algorithm, which utilizes Continuous Active Learning in the TAR framework (Technology Assisted Review). We explored various modifications to the original algorithm and measured their impact on its performance. A modern implementation of this algorithm is proposed, which increases the usability of Continuous Active Learning in real world eDiscovery and high recall retrieval applications.

We investigated a crucial part of the Continuous Active Learning (CAL) process called *refreshing*. We proposed and compared various modifications to it in form of *refresh strategies*. Our results show that by refreshing more often, we can achieve higher recall using less effort. Refreshing after every judgment (`static(k = 1)`) resulted in consistently better effectiveness than the exponentially increasing batch sizes in the AutoTAR. However, frequent refreshing is computationally more expensive. With an efficient implementation and a reasonably modern hardware, refresh strategies relying on frequent refreshing can be practically employed in real world applications. We also defined and analysed alternative refresh strategies which are as effective as `static(k=1)`. In our experiments, various settings of *partial refresh strategy* and *precision strategy* achieved recall scores similar to `static(k=1)` but at a lower computation cost. For situations where computational resources are limited or dataset is very large, *partial refresh strategy* can be used. *Precision based refreshing* is efficient when the relevant documents are abundant and easier to find.

We also briefly explored a way reduce training costs. By reducing the number of training iterations, we observed a significant improvement in running times of our simulation. However, it was also accompanied with noticeable loss of effectiveness. By prioritising recently assessed documents in training, we were able to recover some of that lost effectiveness.

However, this recovery wasn't enough to justify the use of this strategy.

Efficiency and effectiveness are important measures of a retrieval system. For interactive systems in the TAR framework, responsiveness is also crucial. In eDiscovery tasks, it is important to deliver a smooth and lag-free experience to the assessors. We discussed modifications to the refresh strategies which can eliminate any user-perceptible delay during documents assessment.

Finally, we provided an efficient C++ implementation of CAL and all the refresh strategies mentioned in this paper as an open source tool[1]. The tool is designed to be used as a research tool and in real world applications.

7.1 Future Work

There are many avenues for future work which this thesis can extend towards.

There are a many large scale datasets (ClueWeb, Twitter, etc) which far exceed the scale of dataset used in this paper. It is desirable to run our system efficiently on these datasets. We described an enhancement to the *partial refresh strategy* which could potentially achieve this. Additional strategies which deal with large amount of data could also be designed. In addition to runtime efficiency, these strategies will also need to optimize for memory efficiency.

There is a vast amount of literature which deal with scaling and optimizing various steps in the AutoTAR algorithm. Various work approach problems like online training, distributed and large-scale classification. The knowledge gained from exploring these works can be used to design better refresh strategies and enable the use of CAL in wider range of real world applications.

[1] https://github.com/HTAustin/CoreTrec/tree/master/CALEngine

References

[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 and Development in Information Retrieval*, pages 1317–1320. ACM, 2018.

[2] Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40. ACM, 2007.

[3] Hilda Bastian, Paul Glasziou, and Iain Chalmers. Seventy-five trials and eleven systematic reviews a day: how will we ever keep up? *PLoS medicine*, 7(9):e1000326, 2010.

[4] David C Blair and Melvin E Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM*, 28(3):289–299, 1985.

[5] 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 and Development in Information Retrieval*, pages 153–162. ACM, 2014.

[6] Gordon V Cormack and Maura R Grossman. Autonomy and reliability of continuous active learning for technology-assisted review. *arXiv preprint arXiv:1504.06868*, 2015.

[7] 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*, pages 75–84. ACM, 2016.

[8] 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, pages 1039–1048. ACM, 2016.

[9] Gordon V Cormack and Thomas R Lynam. Online supervised spam filter evaluation. *ACM Transactions on Information Systems (TOIS)*, 25(3):11, 2007.

[10] 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*, pages 282–289. ACM, 1998.

[11] Gordon V Cormack, Mark D Smucker, and Charles LA Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5): 441–465, 2011.

[12] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar): 551–585, 2006.

[13] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

[14] 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.*, 17:1, 2010.

[15] Maura R Grossman and Gordon V Cormack. Grossman-cormack glossary of technology-assisted review, the. *Fed. Cts. L. Rev.*, 7:85, 2014.

[16] Maura R Grossman, Gordon V Cormack, Bruce Hedin, and Douglas W Oard. Overview of the TREC 2011 legal track. In *TREC*, volume 11, 2011.

[17] Maura R Grossman, Gordon V Cormack, and Adam Roegiest. TREC 2016 total recall track overview. In *TREC*, 2016.

[18] Matthias Hagen, Steve Göring, Magdalena Keil, Olaoluwa Anifowose, Amir Othman, and Benno Stein. Webis at TREC 2015: tasks and total recall tracks. In *TREC*, 2015.

[19] Donna Harman. Overview of the first TREC conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 36–47. ACM, 1993.

[20] Christopher Hogan, Robert S Bauer, and Dan Brassil. Automation of legal sensemaking in e-discovery. *Artificial Intelligence and Law*, 18(4):431–457, 2010.

[21] Thorsten Joachims. Svmlight: Support vector machine. *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund*, 19(4), 1999.

[22] Matthew Lease, Gordon V Cormack, An T Nguyen, Thomas A Trikalinos, and Byron C Wallace. Systematic review is e-discovery in doctor's clothing. In *Proceedings of the 2nd SIGIR workshop on Medical Information Retrieval (MedIR)*, 2016.

[23] Cheng Li, Yue Wang, Paul Resnick, and Qiaozhu Mei. Req-rec: High recall retrieval with query pooling and interactive classification. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 163–172. ACM, 2014.

[24] Mihai Lupu. Tuw at the first total recall track. Technical report, Vienna University of Technology Vienna Austria, 2015.

[25] Mihai Lupu, Allan Hanbury, et al. Patent retrieval. *Foundations and Trends® in Information Retrieval*, 7(1):1–97, 2013.

[26] Walid Magdy and Gareth J.F. Jones. Pres: A score metric for evaluating recall-oriented information retrieval applications. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 611–618. ACM, 2010.

[27] Douglas W Oard, Jason R Baron, Bruce Hedin, David D Lewis, and Stephen Tomlinson. Evaluation of information retrieval for e-discovery. *Artificial Intelligence and Law*, 18(4):347–386, 2010.

[28] Douglas W Oard, William Webber, et al. Information retrieval for e-discovery. *Foundations and Trends® in Information Retrieval*, 7(2–3):99–237, 2013.

[29] George L Paul and Jason R Baron. Information inflation: Can the legal system adapt. *Rich. JL & Tech.*, 13:1, 2006.

[30] Haoruo Peng, Ding Liang, and Cyrus Choi. Evaluating parallel logistic regression models. In *Big Data, 2013 IEEE International Conference on*, pages 119–126. IEEE, 2013.

[31] Adam Roegiest. On design and evaluation of high-recall retrieval systems for electronic discovery. 2017.

[32] Adam Roegiest, Gordon V Cormack, Maura R Grossman, and Charles Clarke. TREC 2015 total recall track overview. *Proc. TREC-2015*, 2015.

[33] Herbert L Roitblat, Anne Kershaw, and Patrick Oot. Document categorization in legal electronic discovery: computer classification vs. manual review. *Journal of the Association for Information Science and Technology*, 61(1):70–80, 2010.

[34] D Sculley. Online active learning methods for fast label-efficient spam filtering. In *CEAS*, volume 7, page 143, 2007.

[35] D. Sculley and Gabriel M. Wachman. Relaxed online svms for spam filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 415–422. ACM, 2007.

[36] David Sculley. Combined regression and ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 979–988. ACM, 2010.

[37] David van Dijk, Zhaochun Ren, Evangelos Kanoulas, and Maarten de Rijke. The University of Amsterdam (ILPS) at TREC 2015 total recall track. In *TREC*, 2015.

[38] Ellen M Voorhees. The philosophy of information retrieval evaluation. In *Workshop of the cross-language evaluation forum for european languages*, pages 355–370. Springer, 2001.

[39] Hui Yang, John Frank, and Ian Soboroff. Overview of the trec 2015 dynamic domain track. In *Proc. TREC*, 2015.

[40] Zhe Yu, Nicholas A Kraft, and Tim Menzies. How to read less: Better machine assisted reading methods for systematic literature reviews. *CoRR, abs/1612.03224*, 2016.

[41] 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. TREC, 2017.