Refresh Strategies in Continuous Active Learning

Nimesh Ghelani, Gordon V. Cormack, and Mark D. Smucker
University of Waterloo, Canada
{nghelani,gvcormac,mark.smucker}@uwaterloo.ca

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 reviewer works in loop with a machine learning model; the model presents a set of documents likely to be relevant and the reviewer provides relevance feedback. Our focus in this paper 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 reviewer to judge. It is also computationally the most expensive step in CAL. In this paper, 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.

1 Introduction

High recall information retrieval is crucial to tasks such as electronic discovery and systematic review - where the goal is to find all or nearly all relevant documents using minimal human effort. Technical Assisted Review (TAR) methods outperform manual review in legal eDiscovery by reducing the cost spent on human reviewers [4, 7]. In a TAR process, a computer system uses judgments made by human reviewers to classify documents as either relevant or non-relevant. Continuous active learning (CAL) [2] is a TAR protocol where a machine learning algorithm suggests the most likely relevant documents for review and continuously incorporates relevance feedback to improve its understanding of the search task. In a previous study, Cormack and Grossman [2] showed that CAL outperforms other TAR protocols on review tasks from actual legal matters and the TREC 2009 Legal Track. The Total Recall track in TREC 2015 and 2016 evaluated different systems under a simulated TAR setting [5, 6]. Baseline Model Implementation (BMI) based on CAL was used as the baseline in these tracks. BMI implements the AutoTAR algorithm. None of the participating systems were able to consistently outperform the baseline. In this paper, we modify and extend the AutoTAR algorithm.

Refreshing is an important step in the CAL process. During a *refresh*, the relevance judgments from the reviewer 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 reviewer. This step has a high computation cost because it involves

In: Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH18). Co-located with SIGIR 2018, Ann Arbor, Michigan, USA – 12 July 2018, published at http://ceur-ws.org

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. By studying various refresh strategies, we aim to:

- Improve effectiveness of CAL; specifically its capability to achieve higher recall with lesser effort
- Improve computational efficiency of CAL; so that it is responsive and feasible in a production environment.

In this paper, we propose different refresh strategies and compare their effect on the behaviour of CAL. By default, refreshing in AutoTAR is done after a certain number of judgments is received. This number increases exponentially over time. We find that we can make CAL more effective by refreshing after every judgment. However, this comes with significant computation overhead, which with careful implementation and modern hardware, can be minimized for reasonably sized datasets. We also discuss other refresh strategies which have lower computation cost and achieve similar effectiveness. These strategies can be considered when dealing with resource constraints or large datasets.

2 Continuous Active Learning

A general version of the AutoTAR CAL algorithm is described in Algorithm 1. 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 5). In this paper, we simulate human assessors using a set of existing relevance judgments (Step 8). Unlabelled documents are considered non-relevant during the simulation.

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

```
1  Construct a seed document whose content is the topic description
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.
```

Documents are represented as a vector of unigram tf-idf features which are used for training the classifier and calculating relevance likelihood scores. BMI AutoTAR uses sofia-ml[*] [8] to train a logistic regression classifier. It uses the *logreg-pegasos* learner with 100000 iterations of *roc* sampling. A training iteration involves randomly sampling a relevant and a non-relevant document from the training set, computing the loss and adjusting the classifier weights accordingly. The classifier weights are used to calculate relevance likelihood score for any document.

The BMI tool provided by the TREC Total Recall organizers is written in BASH. For efficiency and extensibility, we implemented Algorithm 1 in C++[†] [1]. Our implementation organizes all the document feature vectors in the memory to eliminate disk accesses and enable faster operations. New refresh strategies can be easily implemented and tested using this tool. The tool provides a command line interface to run simulations and a HTTP API for use in real world applications.

3 Experiment Setup

We used the Athome1 test collection from the TREC 2015 Total Recall track [6]. The collection has 290,099 documents and 10 topics. Each topic has an average of 4398 documents labelled as relevant.

[*] https://code.google.com/archive/p/sofia-ml/
[†] https://nims11.github.io/tarcal.html

To measure the effectiveness, we ran a CAL simulation for each topic and refresh strategy until E_{max} number of judgments were made. E_{max} is also known as the max review effort. In our experiments, E_{max} was equal to $2 \times R$ where R is the total number of relevant documents for that topic. 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. We get the average gain curve by averaging the recall over the 10 topics. For the sake of readability and space, we excluded those plots from this paper. Instead, we report certain points of interest from the plots in a table. 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 for each refresh strategy.

To measure the computational efficiency, we fixed $E_{max} = 10000$ and ran the simulation across four 2.20GHz CPUs (Step 7 of Algorithm 1 is the only parallelized step) for each topic and refresh strategy. Different refresh strategies are compared based on the running time of the simulation averaged over the 10 topics.

4 Refresh Strategies

In this section, we describe the refresh strategies we investigated. 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 $O(n \log n)$ [‡] time where n is the number of documents in the corpus. Training takes $O(1)$ time (number of iterations is fixed), computing the relevance likelihood scores take $O(n)$ time (assuming length of the documents to be constant) and sorting the scores take $O(n \log n)$ time. However, the constant costs associated with training and scoring are significantly higher than sorting.

BMI

In BMI AutoTAR, a full refresh is performed after every k judgments. Initially $k = 1$, and after every refresh, is updated using $k \leftarrow k + \lfloor \frac{k+9}{10} \rfloor$. Therefore, k increases exponentially over time.

This strategy scales well with the number of judgments (E) made during the CAL process since only $O(\log E)$ refreshes are done. According to the authors of BMI, 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*" [3].

Static Batch

In *static batch refresh strategy*, a full refresh is performed after every k judgments (k is fixed).

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. For very small values of k (such as 1) and large values of n, pauses as small as half a second due to frequent refreshes can disrupt the user experience. One way to work around this problem is to perform asynchronous refreshes and immediately show the users documents from the old review queue [1]. This 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. While this delay is usually tiny, additional experiments need to be performed to measure the impact of these delays on effectiveness.

Partial Refresh

In this strategy, a full refresh is performed after every k judgments. At the end of each full refresh, s documents with the highest relevance likelihood scores are stored in a *partial refresh set*. After every judgment, a *partial refresh* is performed. In 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 $O(s \log s)$ time.

With some enhancements, this strategy can also help reduce the high memory costs when working with low physical memory or very large datasets (such as ClueWeb). As mentioned in Section 3, the documents are loaded in memory to enable faster operations and improve the responsiveness of the system. Partial refreshes are fast 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.

[‡]In most cases, only top k documents ($k << n$) are needed, and the refresh can be performed in $O(n \log k)$ time

Table 1: Summary of the refresh strategies and their parameters.

Strategy	**Parameters**
bmi_refresh	None
static_batch	$k =$ *no. of judgments between full refreshes*
partial_refresh	$k =$ *no. of judgments between full refreshes* $s =$ *no. of documents in the partial refresh set*
precision_strategy	$m =$ *no. of recent judgments to compute precision on* $p =$ *full refresh is triggered if the precision of last k documents fall below this value*
recency_weighting	$w =$ *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)*

Precision Based Refreshing

In this strategy, a full refresh is performed when the “output quality” of the CAL system falls below some threshold. We define “output quality” as the fraction of relevant judgments in the last m judgments made by the reviewer. A full refresh is performed whenever this fraction falls below some threshold p.

Unlike previous strategies, we don’t define our refresh criteria in terms of elapsed number of judgments. 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.

Recency Weighting

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, 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.

5 Results and Discussion

For sake of space and readability, we encode each strategy with their parameter settings as *strategy_name(param*$1 = x, \ldots$*)*. Table 1 lists all the strategies and their parameters. Table 2 lists the results for different parameter settings of *bmi_refresh*, *static_batch*, *partial_refresh* and *precision_strategy.* We report the recall achieved at different 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 3. For example, “Avg. recall@($E_{norm} = 1.5$)” denotes the average recall achieved across all the topics when $1.5 \times R$ documents haven been judged, where R is the total number of relevant documents for a topic.

With *static_batch*($k = 1$), CAL achieves significantly higher recall of 75% at $E_{norm} = 1$ than *bmi_refresh* which achieves 71.5% recall. *static_batch*($k = 100$) performs worse than *bmi_refresh*, managing to achieve 70.4% recall at the same effort. These results establish that frequent refreshing helps to achieve higher recall. 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_batch(*$k = 100$*)*. *bmi_refresh* is also extremely cheap in terms

Table 2: 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)**
bmi_refresh	0.715	0.827	0.905	1.128	0.22
static_batch($k = 1$)	0.750	0.865	0.926	1.021	49.29
static_batch($k = 100$)	0.704	0.807	0.887	1.167	0.47
partial_refresh($k = 10$,$s = 1000$)	0.753	0.862	0.926	1.008	40.92
partial_refresh($k = 100$,$s = 500$)	0.753	0.855	0.923	1.022	38.28
partial_refresh($k = 100$,$s = 1000$)	0.754	0.856	0.922	1.013	39.57
partial_refresh($k = 100$,$s = 5000$)	0.756	0.855	0.921	1.016	40.70
partial_refresh($k = 500$,$s = 1000$)	0.700	0.785	0.815	1.324	38.63
precision_strategy($m = 25$,$p = 0.4$)	0.698	0.849	0.915	1.129	35.68
precision_strategy($m = 25$,$p = 0.6$)	0.735	0.859	0.923	1.059	40.20
precision_strategy($m = 25$,$p = 0.8$)	0.750	0.862	0.926	1.024	44.64
precision_strategy($m = 25$,$p = 1.0$)	0.752	0.865	0.926	1.014	47.41

of computation cost since it only performs a logarithmic number of refreshes relative to *static_batch* strategies. *bmi_refresh* simulation finished in less than a minute while *static_batch*($k = 1$) took 49 minutes.

We evaluate rest of the refresh strategies by comparing them to *static_batch*($k = 1$).

By fixing $s = 1000$ and varying k in *partial refresh strategy*, we observe that for $k = 10$ and $k = 100$, the difference in recall remains insignificant throughout the CAL process. Their recall scores are also very similar to *static_batch*($k = 1$). They achieve 86.2% and 85.5% recall at $E_{norm} = 1.5$, respectively. For $k = 500$, we observe 78.6% recall at the same effort, which is worse than *bmi_refresh* (82.6%). This is in agreement with our previous observation that more frequent full refreshes increases CAL's effectiveness. *static_batch*($k = 100$) consistently achieved lower recall when compared to *static_batch*($k = 1$) while *partial_refresh*($k = 100, s = 1000$) is as effective as the latter. Based on this, it can be established that partial refreshing contributes significant improvements to recall. On fixing $k = 100$ and varying s we observe no changes to the recall values. *Partial refresh strategies* also improve the running time of simulations by 20% when compared to *static_batch*($k = 1$).

In *precision based refresh strategy*, we fix $m = 25$ and vary p. For $p = 0.8$ and $p = 1.0$, *precision_strategy* achieves 75% recall at $E_{norm} = 1$ which is similar to *static_batch(k = 1)*. This similarity of recall is also seen at $E_{norm} = 1.5$ and $E_{norm} = 2$. When $p = 1$, *precision_strategy* refreshes whenever a non-relevant judgment is made, thus behaving very similar to *static_batch*($k = 1$). For lower values of p, we observe lower recall values during the initial stages (73.5% recall when $E_{norm} = 1$ for $p = 0.6$). However, they catch up to *static_batch*($k = 1$) at higher E_{norm}, as relevant documents become rarer (85.9% recall when $E_{norm} = 1.5$ for $p = 0.6$). *precision_strategy* improved the running time of simulations by 15% on average when compared to *static_batch*($k = 1$). *precision_strategy* 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_strategy* tends to keep refreshing after every judgment.

During our initial experiments, *recency_weighting* seemed to have no impact on the recall values. On further experiments, we found that the default number of training iterations ($it = 100000$) was very high for recency weighting to cause any difference. 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 76% when compared to *static_batch*($k = 1$). We used *recency_weighting* to see whether it could recover the lost effectiveness. *recency_weighting*($w = 1, it = 1000$) is equivalent to *static_batch*($k = 1, it = 1000$) and it achieves 70.4% and 79.8% recall when E_{norm} is equal to 1 and 1.5 respectively. By increasing w, we observe an increase in recall for $E_{norm} \in \{1.5, 2\}$. However, the recall is consistently and significantly lower when compared to *static_batch*($k = 1$). For example, *recency_weighting*($w = 10, it = 1000$) is only able to achieve 82.4% recall at $E_{norm} = 1.5$.

6 Future Work

There are many large scale datasets (ClueWeb, Twitter, etc) which far exceed the scale of dataset used in this paper. It is desirable to run CAL 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.

In this paper, we measure computational efficiency of strategies using the running time of the simulations. In a more practical setting where an actual human is judging documents, responsiveness of the CAL system also becomes important. Under *static batch refresh strategy*, we mentioned the idea of introducing a *delay* to improve responsiveness of CAL systems. Additional experiments which simulate and measure the impact of those delays could be designed.

7 Conclusion

We investigated a crucial part of the Continuous Active Learning (CAL) process called *refreshing*. We proposed and compared various *refresh strategies*. Our results show that by refreshing more often, CAL can be used to achieve higher recall. Refreshing after every judgment ($static\_batch(k = 1)$) results in consistently better performance than the original BMI AutoTAR in terms of recall. However, frequent refreshing is computationally more expensive than the BMI refresh strategy. But with an efficient implementation on modern computers, frequent refreshing can be practically used in real world CAL systems. We also defined and analysed alternative refresh strategies which are as effective as refreshing after every judgments. In our experiments, various settings of *partial refresh strategy* and *precision strategy* achieved recall scores similar to $static\_batch(k = 1)$ 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 provide an efficient C++ implementation of CAL and all the refresh strategies mentioned in this paper as an open source tool. The tool is designed to be used both as a research tool and in real world applications.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (Grants CRDPJ 468812-14 and RGPIN-2014-03642), in part by a Google Founders Award, and in part by the University of Waterloo.

References

[1] Abualsaud, M., Ghelani, N., Zhang, H., Smucker, M.D., Cormack, G.V., Grossman, M.R.: A system for efficient high-recall retrieval. In: Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM (2018)

[2] Cormack, G.V., Grossman, M.R.: 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, ACM (2014) 153–162

[3] Cormack, G.V., Grossman, M.R.: Autonomy and reliability of continuous active learning for technology-assisted review. arXiv preprint arXiv:1504.06868 (2015)

[4] Grossman, M.R., Cormack, G.V.: Technology-assisted review in e-discovery can be more effective and more efficient than exhaustive manual review. Rich. JL & Tech. **17** (2010) 1

[5] Grossman, M.R., Cormack, G.V., Roegiest, A.: Trec 2016 total recall track overview. In: TREC. (2016)

[6] Roegiest, A., Cormack, G.V., Grossman, M.R., Clarke, C.: Trec 2015 total recall track overview. Proc. TREC-2015 (2015)

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

[8] Sculley, D.: Combined regression and ranking. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2010) 979–988