An Implicit Feedback-based Approach to the Evaluation of Text Analysis Techniques for Software Engineering

Kostadin Damevski
Virginia State University
Petersburg, VA
kdamevski@vsu.edu

David Shepherd
ABB, Inc.
Raleigh, NC
david.shepherd@us.abb.com

Lori Pollock
University of Delaware
Newark, DE
pollock@cis.udel.edu

***Abstract*—One of the key challenges facing the community of researchers in text-based analysis for software engineering is evaluation of new techniques and tools that leverage those techniques. In this paper, we explore the paired interleaving approach, used to evaluate Internet search engines, as an alternative to creating annotated corpora for gold sets. In particular, we examine this approach in the context of evaluating feature location techniques, which collectively are a key software engineering client application for text-based analysis of software artifacts. This paper describes the paired interleaving approach and presents the challenges in customizing this technique for comparative evaluation of feature location techniques with different underlying text analysis and preprocessing. Better evaluations of text-based analyses and their ultimate client tool applications would help remove the barriers to industry adoption of text analysis in software tools.**

***Keywords*-Evaluation, annotated corpora, feature location, paired interleaving.**

I. Introduction

A common challenge remains - comparative evaluation of new text analysis techniques and their targeted client applications to assess the relative effectiveness of different strategies and inform future advancements [2], [7], [8]. Evaluation is hindered by the lack of annotated corpora for many of the client applications and individual text analyses of software. This is mostly due to the need to involve human subjects to judge the output since software engineering is an inherently human task. Most studies involve only a few human subjects on a few examples because it is too costly and time consuming to scale up these evaluations.

More concretely, we focus here on evaluation of feature location techniques, FLTs, as client applications of text analysis. Locating code related to a specific feature set is often a software developer's first step in performing a software maintenance task [1]. Researchers have developed Feature Location Techniques (FLTs), including static, dynamic, and hybrid approaches, using various forms of text analysis, to help software engineers identify relevant code that is often scattered across a large, complex software system [2], [3], [4], [5]. Feature location is one of the key software maintenance tasks used to evaluate the usefulness of different text analysis techniques for software [6].

When they have been performed, comparative studies of FLTs have compared each FLT's "computed relevant set of code units" against an expected "gold set of relevant code units" [7], [2]. Effectiveness is measured in terms of precision, recall and F-measure using the gold set of code units as an oracle. The results of the comparative studies rely heavily on the reliability of the gold set. Unfortunately, there are several concerns with existing gold sets of relevant code for FLT evaluation: (1) they fail to capture real developer intent as they are typically generated post hoc; (2) they are often produced by researchers, not actual developers, making them one step farther from actual use case; (3) researchers have observed that humans vary in their opinion of what comprises the relevant code to a particular feature and thus there is subjectivity [9]; (4) researchers have also found that the relevant code to a particular feature might be dependent on the maintenance task to be performed in relation to the feature, implying the gold set is context dependent [9].

In this paper, we propose a novel approach to the evaluation of FLTs that enables comparison between FLTs as software developers use a feature location tool during their regular software maintenance process, and without explicitly creating a gold set of relevant code. The key insight is to leverage the paired interleaving technique used to evaluate Internet search engines [10]. The comparative study design is similar to sensory analysis, where the tastes of two products are compared by asking the test subjects to express a preference for a particular product, instead of rating each of the products on an absolute (i.e., Likert) scale. Two FLTs are implemented within the same feature location tool interface, and results from the two FLTs are presented to the user interleaved alternately unbeknownst to the user, as they make queries. Users' preferences for results from different queries are collected and analyzed. Thus, implicit feedback is gathered from actual developers in the context of their normal software maintenance process, and large deployment of the tool can enable significant data collection.

The main contributions of this paper are:

1) introducing a novel approach to evaluating text analysis techniques as applied to feature location, and
2) describing the challenges to instantiating the approach

for various FLT categories.

The paired interleaving approach has the potential to eliminate the problems associated with explicit construction of gold sets of relevant code and common measurements of precision, recall and F-measure with respect to the gold set.

II. Paired Interleaving for FLT Comparison

A. Overview

The key challenge in basing evaluation on usage data instead of expert judgements is properly interpreting the gathered data through carefully relating the observable statistics to the quality of the techniques under evaluation [11]. When applied to evaluate Internet search engines, paired interleaving achieves an online comparison of different result sets associated with different search engine techniques. The two result sets, each from different search engine technologies, are merged into a single interleaved set, which is presented to the user such that the observed user behavior, in the form of clickthroughs, is indicative of a preference for a particular search engine.

We believe that paired interleaving can also be applied to evaluate the relative effectiveness of two FLTs, or FLTs with different text analysis or preprocessing. The evaluation is conducted by recording implicit feedback (clickthrough data) during normal software developer interaction with a feature location tool that embeds and interleaves the responses of different complete approaches to feature location. The relative preference for one approach over another is indicated by a preference for the results of one FLT over another, in the span of a set of queries by a number of developers.

A set of properties, initially discussed by Joachims [11], are required for the paired interleaving process to be fair and robust to bias. First, while the feature location tool's backend is augmented to produce results of two separate FLTs for each developer query, the tool's frontend and the the developer's search experience must remain unaltered. Second, the two sets of FLT results, must be interleaved fairly and presented to the developer as a unified list, with no indication of the origin of each result. Thus configured, the evaluation experiment can determine developer preference for a specific FLT by counting developer clicks on its results. A statistically significant difference in the number of clicks is considered indicative of the relative effectiveness of a particular FLT.

B. Balanced Interleaving

Consider the process of evaluating, via paired interleaving, FLT_A and FLT_B, which produce result sets $A = \{a_1, a_2, ..., a_n\}$ and $B = \{b_1, b_2, ..., b_n\}$ for a particular developer query q. A reasonable assumption here is that each of the results sets are ranked in descending order of relevance to q. The evaluation is performed using a set of queries, $q_1, q_2, ..., q_m$, where m is a sufficiently large to

```
Input: result lists A and B
I := {}; $k_A$ := 0; $k_B$ := 0;
N := A.length();
aFirst := RandomBit();
while $k_A + k_B < N$ do
    if $k_A < k_B$ or ($k_A == K_b$ and aFirst == 1) then
        if not I.contains(A[$K_A$]) then
            I.insert(A[$k_A$]);
        end if
        $k_A$ := $k_A$ + 1;
    else
        if not I.contains(B[$K_A$]) then
            I.insert(B[$k_B$]);
        end if
        $k_B$ := $k_B$ + 1;
    end if
end while
Output: interleaved list I
```

Figure 1. Pseude-code of the Balanced Interleaving.

establish a statistically significant preference for a particular FLT. Interleaving the two result sets A and B produces an interleaved set I, also of size n, instead of twice that, in order to ensure that the developer's experience remains unaltered.

A commonly used algorithm for performing the interleaving is Balanced Interleaving [11], whose pseudo-code is given in Figure 1. This algorithm randomly determines whether the interleaving begins with a result from FLT_A or FLT_B, and then proceeds by inserting a non-duplicate result from each FLT. Given two result sets $A = \{a, b, c, d\}$ and $B = \{e, f, g, h\}$, a Balanced Interleaving with FLT_A first would produce the interleaved set $I = \{a, e, b, f\}$, while $I = \{e, a, f, b\}$ if FLT_B won the random coin toss. It can be proved that a Balanced Interleaving always exists for two sequences A and B of similar, non-zero size.

Balanced Interleaving scores each click on the interleaved set I into three categories: the result set A (i.e. FLT_A) wins, the result set B (i.e. FLT_B) wins, or neither wins (tie). The per-click scores are aggregated into a per-query preference using the same three categories, which, in turn, are combined into a single metric that expressed the entire paired interleaving experiment's result.

To score each click, Balanced Interleaving considers only the rank of the clicked result within the lists A and B. In other words, clicks on results that appear higher in one of the original result sets are considered as wins for the corresponding FLT. We illustrate the Balanced Interleaving scoring algorithm for one specific query, using pseudo code in Figure 2. To aggregate the wins across a number of queries, and determine a magnitude for the preference for a particular FLT, the following equation (first proposed in [10]) can be used. A positive value for Δ_{AB} indicates a preference for FLT_A, a negative value a preference for FLT_B, while the magnitude of this metric is indicative of the strength of the preference.

```
Input: result lists A and B, interleaved list I, list of click
    indices C (into I)
 1: $C_{max}$ := max(C); $h_A$ := 0; $h_B$ := 0;
 2: $p_A$ := A.indexOf( I[$C_{max}$] );
 3: $p_B$ := B.indexOf( I[$C_{max}$] );
 4: $p$ := min($p_A$,$p_B$);
 5: for i := 0 $\rightarrow$ C.length do
 6:     $C_i$ := C[i];
 7:     $q_A$ := A.subList(0, p).indexOf( I[$C_i$] );
 8:     $q_B$ := B.subList(0, p).indexOf( I[$C_i$] );
 9:     if $q_A < q_B$ then
10:         $h_A$ := $h_A$ + 1;
11:     else if $q_B < q_A$ then
12:         $h_B$ := $h_B$ + 1;
13:     end if
14: end for
15: if $h_A > h_B$ then
16:     winner := A;
17: else if $h_B > h_A$ then
18:     winner := B;
19: else
20:     winner := tie;
21: end if
Output: per-query winner (A, B, or tie)
```

Figure 2. Balanced Interleaving's click scoring algorithm, aggregated per query.

$$\Delta_{AB} = \frac{wins(A) + \frac{1}{2}ties(A,B)}{wins(A) + wins(B) + ties(A,B)} - 0.5 \quad (1)$$

Similar results in the FLTs under comparison can present fairness problems to the Balanced Interleaving algorithm. For instance, let's consider a case where, for some query, FLT_A retrieves the result set $A = \{a,b,c,d\}$, while FLT_B produces $B = \{b,c,d,a\}$. If the Balanced Interleaving's random bit prefers FLT_A then the interleaved set $I = \{a,b,c,d\}$, while beginning with FLT_B produces $I = \{b,a,c,d\}$. A random click on either of these result sets is more likely to prefer FLT_B over FLT_A. This introduced bias in interleaving some similar result sets is a disadvantage of Balanced Interleaving, which has been attempted to be remedied by the proposal of new interleaving algorithms, most notably Team-Draft Interleaving [12]. However, simulations and large scale experiments have shown that these algorithms also have a tendency for bias in certain scenarios, and that, in aggregate, Balanced Interleaving performs well enough for paired interleaving evaluation [10].

In addition to producing a measurement for the preference for a particular FLT (i.e. Δ_{AB} in equation 1), it is useful to have a confidence bound for this measurement. This confidence bound can be useful in determining whether the results of a paired interleaving experiment are statistically significant as well as determine whether gathering more samples could be beneficial. To determine the confidence bound, without making assumptions about the distribution of the data, bootstrapping estimators can be used. A more detailed description on applying bootstrapping to Internet search results can be found in Chapelle et al. [10].

C. Challenges

To our knowledge, paired interleaving has thus far only been applied to traditional information retrieval (e.g. Internet search). We expect the following challenges when applying paired interleaving for evaluation of FLTs, and discuss each of them, in turn.

Program navigation vs. maintenance. A significant challenge in using paired interleaving is that the FLT evaluation is conducted with queries during user interaction with the feature location tool, and usage scenarios thus may not necessarily correspond to a specific maintenance task. For example, some of the queries used during use of the tool may be intended for some other purpose, for instance, project navigation. On the other hand, "gold" sets, by selecting query terms from text in maintenance issues, are more likely to only contain maintenance related queries.

FLTs may return few or no results. Unlike Internet search, there are many query terms that are a poor match to all the elements of a specific project, and therefore it is not uncommon for FLTs to retrieve few or no results. Since paired interleaving requires that the number of items displayed has to be consistent between queries, this may disqualify a great number of queries from being used in evaluation.

FLTs may produce heterogeneous result sets. Paired interleaving has been shown to be effective when the results are completely uniform in type, as in Internet search where all the results are documents. However, FLTs often retrieve heterogeneous results, such as distinct program elements (e.g., comments, methods, classes, line numbers). The anonymity of the FLTs under evaluation may be compromised if they are easy to distinguish by the program elements they report (e.g. an FLT that produces only method elements).

Evaluating FLTs that strongly rely on a specific visualization technique. Developers may not be closely familiar with the project under maintenance, and therefore, a visualization of the feature location result set may be beneficial to developers. A number of FLTs have proposed visualizations to improve their effectiveness (e.g., [4]). Visualized result sets cannot be interleaved without the developer recognizing the FLT that originated each result, and therefore are a poor fit for this evaluation technique.

III. Related Work

There has been little work on improving the infrastructure for evaluation of text analysis techniques and their client software engineering tools. A recent comprehensive survey of FLTs by Dit et al. [2] supports the motivation for this kind of work, concluding that a major impediment to progress in FLT research is the difficulty in comparing approaches.

The TraceLab [7] research environment's goal is to facilitate rapid experimentation and evaluation of FLTs. Using TraceLab, FLTs are rapidly constructed by composing a set of TraceLab components, and evaluated using a set of (five) provided gold sets. While TraceLab is efficient at constructing and evaluating FLTs, we argue that it suffers from limitations associated with using gold sets, such as their limited size, subjectivity, and distance from actual developers. However, due to TraceLab's very rapid FLT experiment design, we envision that future FLT evaluation may combine both paired interleaving and TraceLab, by starting with a TraceLab study as a proof-of-concept, followed by a comprehensive comparative experiment using paired interleaving.

Sando [8] is a research-extensible local code search tool, which has also been proposed as a means to improve the effectiveness of FLT research by improving evaluation and dissemination. As Sando is a practical FLT tool intended for developers, it could also be used as a platform for conducting FLT evaluation via the paired interleaving approach proposed in this paper.

IV. Summary and Future Work

This paper introduced an alternative to creating gold sets for evaluation of text analysis of software artifacts and their client software engineering tools. We are currently designing and implementing several studies to validate and assess the paired interleaving approach for comparing feature location techniques under different text analysis configurations.

References

[1] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. on Soft. Eng.*, vol. 32, no. 12, pp. 971–987, 2006.

[2] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Soft. Maint. and Evolution: Research and Practice*, 2011.

[3] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Working Conference on Reverse Engineering*, 2004, pp. 214–223.

[4] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *Int. Conf. on Software Engineering (ICSE)*, 2011.

[5] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. V. Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *Int. Conf. on Aspect-Oriented Software Development*, 2007.

[6] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *19th IEEE Int. Conf. on Program Comprehension*, 2011.

[7] B. Dit, E. Moritz, and D. Poshyvanyk, "A tracelab-based solution for creating, conducting, and sharing feature location experiments," in *IEEE Int. Conf. on Program Comprehension*, 2011.

[8] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: An extensible local code search framework," in *Proceedings of the 20th Int. Symp. on the Foundations of Software Engineering*, 2012, Demo Track.

[9] E. Hill, "Integrating natural language and program structure information to improve software search and exploration," Ph.D. dissertation, University of Delaware, August 2010.

[10] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue, "Large-scale validation and analysis of interleaved search evaluation," *ACM Trans. on Information Systems*, vol. 30, no. 1, Mar. 2012.

[11] T. Joachims, "Evaluating retrieval performance using clickthrough data," in *Text Mining*, J. Franke, G. Nakhaeizadeh, and I. Renz, Eds. Physica/Springer Verlag, 2003, pp. 79–96.

[12] F. Radlinski, M. Kurup, and T. Joachims, "How does clickthrough data reflect retrieval quality?" in *Proceedings of the 17th Conference on Information and Knowledge Management*, 2008, pp. 43–52.