Searching for Relevant Lessons Learned Using Hybrid Information Retrieval Classifiers: A Case Study in Software Engineering

Tamer Mohamed Abdellatif, Luiz Fernando Capretz and Danny Ho
Dept. of Electrical & Computer Engineering
Western University
{tmohame7, lcapretz}@uwo.ca, danny@nfa-estimation.com

Abstract

The lessons learned (LL) repository is one of the most valuable sources of knowledge for a software organization. It can provide distinctive guidance regarding previous working solutions for historical software management problems, or former success stories to be followed. However, the unstructured format of the LL repository makes it difficult to search using general queries, which are manually inputted by project managers (PMs). For this reason, this repository may often be overlooked despite the valuable information it provides. Since the LL repository targets PMs, the search method should be domain specific rather than generic as in the case of general web searching. In previous work, we provided an automatic information retrieval based LL classifier solution. In our solution, we relied on existing project management artifacts in constructing the search query on-the-fly. In this paper, we extend our previous work by examining the impact of the hybridization of multiple LL classifiers, from our previous study, on performance. We employ two of the hybridization techniques from the literature to construct the hybrid classifiers. An industrial dataset of 212 LL records is used for validation. The results show the superiority of the hybrid classifier over the top achieving individual classifier, which reached 24%.

Keywords — Professional search, lessons learned recall, project management lessons learned, information retrieval, hybrid classifiers

1 Introduction and Background

Software organizations supposedly store their historical data in lessons learned (LL) repositories. This data can be success stories or solutions to issues that were discovered in previous projects, which can be reused in similar future situations. On the other hand, this data can also include failure stories, pitfalls or mistakes from previous projects to be avoided in similar future projects. Accordingly, the LL repositories contain information that can be useful in guiding project managers (PMs) to leverage opportunities or avoid repeating past mistakes. For example, an LL record concerning a decision about whether to implement a mobile application in-house or outsource the implementation can take the following form:

Context: the project scope includes an implementation of a small-sized mobile application. This mobile application is not reusable, i.e., it will only be used in this project.

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

Problem: if the mobile application is of a small size, then the organizational process overhead, such as quality assurance and management reporting tasks, will affect the profit of implementing the mobile application in-house.

Recommended actions: outsource the implementation to an external mobile application specialized company. Contact the purchase team for a trusted partners list.

However, this LL information can only be beneficial if project managers refer to it frequently to solve present issues or to seek potential opportunities, which is not always the case. Unfortunately, LL are often abandoned by PMs due to the effort and time needed to manually search for relevant LL records within the unstructured, i.e., natural language, LL repository [11]. Also, it has been found to be difficult to search for relevant LL records using a general search methodology or search terms manually defined by PMs. This calls for automatic domain specific, i.e., professional search, LL recall solutions [9]. By automatic we mean that there should be no need for manual searching to facilitate the exploitation of LL. In a previous study, we worked on satisfying this need by providing an automatic domain specific LL recall, i.e., retrieval, system [1].

Regarding the software engineering literature, there is a paucity of software engineering research addressing LL recall solutions [9]. As per our knowledge, the most relevant studies have been conducted by Sary and Mackey [5] and by Weber et al. [10]. Both studies employed case-based reasoning techniques in order to build their systems. Also, these studies have the common limitation of the need to arrange the LL repository records in a question-answer format. This transformation is difficult to achieve in reality as it demands extra effort and time. This limitation is not valid in our solution since the classifier is constructed using the LL repository records as is, with no transformation needed. Also, these studies are different from our solution since we employ information retrieval (IR) techniques instead of case-based reasoning. As per our knowledge, we are the first to apply the IR models to the software project management LL recall context [2].

In order to make our solution specific to the project management LL domain, we relied on two of the existing and most influential project management artifacts, namely project management issues and risks, to automatically invoke our constructed classifiers. Since these artifacts are already associated with the software development project lifecycle, there is no need for the manual involvement of PMs. We relied on some of the most popular IR models from the literature to construct the LL classifiers. In addition, we evaluated our solution through an empirical case study using a real dataset from industry [1]. The results of the case study proved the effectiveness of our solution by achieving an accuracy rate of about 70%.

In this paper, we conduct an extension case study. In this study, we constructed hybrid LL classifiers by combining multiple LL classifiers from our previous work in order to examine the impact of this hybridization on the performance of the LL classifiers. Our main motive for conducting such an extension was that although several domains have studied the hybridization of classifiers [4, 8], it has not been studied in the LL recall context.

The results from our extension study showed an improvement in the majority of the hybrid cases that were studied. Although some of the cases showed no improvement or showed a decrease in the performance of the hybrid classifiers, the fact that the hybridization was successful in most of the other cases, and in other software engineering domains [4, 8], makes considering the hybrid classifiers interesting for future studies regarding LL recall.

2 Case Study Design

2.1 Previous Case Study Summary

In our previous work [1], we provided a solution to improve the retrieval of the software LL and make them available to PMs. We relied on two of the software project management artifacts, namely project management issues and risk register. These two artifacts were used to construct a query string on-the-fly. The constructed query string was then used to *automatically* call an LL classifier in order to retrieve LL records relevant to the project at hand. Regarding the LL classifiers, we employed three of the popular IR models to construct the classifiers, and we considered multiple parameter values to configure and construct multiple classifiers. The employed IR models were the Vector Space Model (VSM) [6], the Latent Semantic Indexing (LSI) [3], and the Latent Dirichlet Allocation (LDA) [6, 7]. The parameter values considered for the VSM were as follows: *term weight* (tf-idf, sublinear tf-idf, boolean), where tf is term frequency and idf is inverse document frequency, and *similarity* (cosine, overlap) [6, 7]. In this paper, we use the following notation to refer to a constructed VSM classifier: *VSM+term weight+similarity method+preprocessing step*. For example, the classifier *VSM+tf-idf+cosine+none* refers to the LL classifier constructed by employing the VSM IR model with the configuration parameter *term weight* set to tf-idf, the *similarity* method set to the cosine method and applying none of the preprocessing steps on the data.The parameter values for LSI were *term weight* (tf-idf, sublinear tf-idf, boolean), *similarity* (cosine), and *number of topics* (32, 64, 128, 256). For LDA, the parameter values were *number of topics* (32, 64, 128, 256) and *similarity* (conditional probability). Like VSM, the notations used to refer to LSI and LDA classifiers are *LSI+term weight+similarity method+number of topics+preprocessing step* and *LDA+number of topics+preprocessing steps*, respectively.

In order to reduce the noise in the input data, we considered two of the preprocessing steps from the natural language processing literature, namely *stemming* and *stopping* steps [7]. In the stemming step, the word is replaced by its morphological root. In the stopping step, commonly used words, such as "the" in the English language, are removed [7]. We considered studying the constructed classifier by applying four combinations of these steps to the input data: 1) none of the preprocessing steps were applied, 2) only the stemming step was applied, 3) only the stopping step was applied, and 4) both steps were applied together.

In our previous case study, we considered all combinations for all IR models, parameter values and input preprocessing steps, which led to the construction of 88 LL classifiers. The performance of all the considered classifiers was validated using the top-K performance metric from the IR literature [7, 8]. Top-K, top-20 in our study, examines the number of queries where the classifier returns at least one relevant record within the first K items of the retrieved list.

Also, for our solution validation, we relied on a real industrial dataset provided by a multinational software development partner. The validation dataset included 212 real LL records from 30 projects and 55 project management issues and risk records. In addition, both the performance results and the impact of each parameter value on the results were statistically analyzed. A satisfactory maximum performance result of 70% for the top-20 was recorded, which positively proved the effectiveness of our provided solution [1].

2.2 Lessons Learned Hybrid Classifiers

Different classifiers can perform in different ways in relation to the same dataset and inputs. This means that different classifiers can exhibit different errors and advantages. Thus, combining multiple classifiers together can lead either optimistically to better performance as they complement each other to avoid individual errors, or negatively to worse performance when they distract each other. This depends heavily on the chosen classifiers. Based on this information, we aim, in our case study, to combine multiple classifiers from our previous work to construct a hybrid classifier, and then study the impact of this combination on performance.

Hybridization Techniques

We employed two hybridization techniques from the literature [8] to combine individual classifiers into one hybrid classifier. These two techniques are *Borda count* and *Score Addition.* The *Borda* technique is a rank-based technique. This means that it relies on the rank, (i.e., the order in the retrieved list of the retrieved item, the relevant LL in our case), within the classification results list from each individual classifier, to assign this item a rank score. The *Borda* count can be calculated as stated in [8] as

$$Borda(d_K) = \sum_{C_i \epsilon C} M_i - r(d_K|C_i) + 1 \qquad [8]$$

where d_K is the retrieved list item for which the Borda count is calculated, C is the collection of the hybrid classifiers, C_i is the ith classifier within the C collection, M_i is the number of retrieved items that received a non-zero score in the retrieved list by the classifier C_i, and $r(d_K|C_i)$ is the d_K rank or order within the C_i retrieved list [8].

On the other hand, the score addition technique relies on the items weight, or the score given by the individual classifiers. The total hybrid score of each retrieved item is calculated as the summation of the individual score of this item from each of the combined classifiers [8]. In order to avoid any mistaken bias to a certain classifier due to the weighting scale, the items weights in each of the combined classifiers list are scaled to be within the same range of [0-1]. Accordingly, the individual items score addition can be calculated as follows:

$$ScoreAddition(d_K) = \sum_{C_i \epsilon C} S(d_K|C_i)$$

where $S(d_K|C_i)$ is the score of d_K given by the classifier C_i [8]. Finally, the items are placed in a descending order, based on their total score.

Hybrid Classifiers Selection

The selection of the combined classifiers has a crucial impact on the performance of the constructed hybrid classifier. For this reason, we tried to choose the classifiers that can positively complement each other. Thus, we chose the classifiers that had been exposed to different formats of the input data, because such classifiers would have a higher chance of coming up with different insights and conclusions regarding the dataset at hand, which we thought could improve their combined performance. That said, we decided to proceed with the classifiers that were constructed by applying different input preprocessing step combinations.

These preprocessing steps were employed in four combinations, as described in Section 2.1, leading to four classifier subspaces. So, for each IR model, we considered a top performer classifier from each of the four classifier subspaces. This resulted in the selection of four classifiers from each of the VSM, LSI, and LDA models that included: the top classifier when none of the preprocessing steps were applied, the top classifier when the stemming step was applied, the top classifier when the stopping step was applied, and finally the top performer classifier when both the stemming and stopping steps were applied together. In our experiment, we examined the performance of the hybrid classifiers constructed by combining the four selected classifiers of each IR model in pairs. In addition to studying these pairs of classifier combinations, we studied the performance of the combination of the four selected classifiers in each IR model as well. Finally, we combined all of the selected classifiers from all IR models together (four classifiers from each of the three IR models considered). All the classifier combinations are shown in Table 1.

Table 1: Hybrid Classifiers

Top-20 Hybrid Classifiers	
LDA Top Classifiers	LDA_T1:LDA+32+None LDA_T2:LDA+32+Stopping LDA_T3:LDA+32+Stemming LDA_T4:LDA+32+Stemming and stopping
Combination ID	Combined Classifiers
1	LDA_T1, LDA_T2
2	LDA_T2, LDA_T3
3	LDA_T1, LDA_T4
4	LDA_T2, LDA_T4
5	LDA_T3, LDA_T4
6	LDA_T1, LDA_T3
7	LDA_T1, LDA_T2, LDA_T3, LDA_T4
LSI Top Classifiers	LSI_T1: LSI+Tf-idf+Cosine+128+None LSI_T2: LSI+Sublinear+Cosine+64+Stopping LSI_T3: LSI+Sublinear+Cosine+256+Stemming LSI_T4: LSI+Tf-idf+Cosine+128+Stemming and stopping
Combination ID	Combined Classifiers
8	LSI_T2, LSI_T3
9	LSI_T3, LSI_T4
10	LSI_T1, LSI_T2
11	LSI_T1, LSI_T3
12	LSI_T1, LSI_T4
13	LSI_T2, LSI_T4
14	LSI_T1, LSI_T2, LSI_T3, LSI_T4
VSM Top Classifiers	VSM_T1: VSM+Sublinear+Cosine+None VSM_T2: VSM+Sublinear+Cosine+Stopping VSM_T3: VSM+Tf-idf+Cosine+Stemming VSM_T4: VSM+Sublinear+Cosine+Stemming and stopping
Combination ID	Combined Classifiers
15	VSM_T1, VSM_T2
16	VSM_T1, VSM_T4
17	VSM_T2, VSM_T4
18	VSM_T1, VSM_T3
19	VSM_T2, VSM_T3
20	VSM_T3, VSM_T4
21	VSM_T1, VSM_T2, VSM_T3, VSM_T4
22	Combinations: 7, 14, 21

3 Case Study Results and Discussion

The performance results for each of the constructed hybrid classifiers were compared to the performance results of each of the combined classifiers using the relative performance improvement (RI) percentage. The RI calculation is formulated as follows:

$$RI = \frac{P(HC) - HighestP(Combined\,Classifiers)}{HighestP(Combined\,Classifiers)}$$

where $P(HC)$ is the value of the performance metric P for the hybrid classifier HC, and $HighestP()$ method returns the highest performance metric value among the performance values of the combined classifiers [8].

The results for the hybrid classifiers that we considered and the impact on the top-20 are shown in Table 2. In the case of using the score addition method, the hybrid classifier results showed either an improvement or no effect against the individual classifiers in about 77% of the cases considered. In other words, the score addition combination led to a decrease in performance in only five cases. Regarding the Borda method, there was an improvement or no effect in about 59% of the cases. The maximum improvement was 15% for the score addition method and 24% for the Borda method.

Table 2: Hybrid Classifiers Top-20 Results

Comb. ID	**Top Individual Perform. (%)**	**Score Addition (%)**	**RI (%)**	**Borda Count (%)**	**RI (%)**	**Comb. ID**	**Top Individual Perform. (%)**	**Score Addition (%)**	**RI (%)**	**Borda Count (%)**	**RI (%)**
1	46	50	8	56	20	12	70	74	5	69	-3
2	46	52	12	57	24	13	69	69	0	69	0
3	52	50	-4	50	-4	14	70	70	0	70	0
4	52	54	4	56	7	15	61	61	0	59	-3
5	52	46	-11	44	-14	16	61	70	15	65	6
6	41	46	14	44	9	17	61	61	0	59	-3
7	52	48	-7	48	-7	18	70	65	-8	63	-11
8	69	69	0	70	3	19	70	70	0	70	0
9	69	70	3	72	5	20	70	72	3	72	3
10	70	67	-5	70	0	21	70	70	0	65	-8
11	70	72	3	69	-3	22	70	72	3	72	3

Comb. ID = combination id, Perform. = Performance

An important additional observation is that the combination performance exceeded the 70% top-20, which was the top performance recorded among all the individual classifiers in our previous experimental work. For score addition, this was recorded in four cases where top-20 performance accuracies of 74% and 72% were recorded. In the case of Borda, this was achieved in three cases where a top-20 of 72% was recorded. Also, it is important to highlight that in approximately 73% of the cases, the score addition results outperformed or were comparable to the Borda results.

Although the hybridization did not prove to be an improvement in all cases within our experiment, the number of the improved cases, especially the 77% of cases for score addition, are considered satisfactory and encourage the consideration of hybrid classifiers within the LL retrieval context.

4 Conclusion

In this paper, we provided an extension of our previous empirical study regarding the construction of an automatic software management LL recall system. Our solution represented a domain specific search, i.e., professional search, as we constructed the search query using two of the existing project management artifacts instead of employing a generic manual search. In our extension, we studied the impact of the hybridization of the LL classifiers on performance. We relied on the existing LL classifiers from our previous study in constructing the hybrid classifiers. In this study, we employed two combination techniques in constructing the hybrid classifiers. A comparison was conducted between the performance of each hybrid classifier and the performance of the top performer from the combined individual classifiers. The top-K performance metric was employed to measure the retrieval accuracy of the classifiers considered. The study results showed a relative improvement, or no effect, of the hybrid classifiers performance against the individual classifiers performance in about 77% of the cases in the top-20 using the score addition method. Although, the improvement was not satisfactory in some cases, the overall results were encouraging and provided positive insights regarding employing hybrid IR models to provide a domain specific LL recall solution.

References

[1] Abdellatif, T.M., Capretz, L.F., Ho, D.: Automatic recall of software lessons learned for software project manager. Submitted (2018)

[2] Chen, T.H., Thomas, S.W., Hassan, A.E.: A survey on the use of topic models when mining software repositories. Empirical Software Engineering **21**(5) (2016) 1843–1919

[3] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science **41**(6) (1990) 391–407

[4] Kocaguneli, E., Menzies, T., Keung, J.W.: On the value of ensemble effort estimation. IEEE Transactions on Software Engineering **38**(6) (2012) 1403–1416

[5] Sary, C., Mackey, W.: A case-based reasoning approach for the access and reuse of lessons learned. In: Proceedings of the Fifth Annual International Symposium of the National Council on Systems Engineering. Volume 1. (1995) 249–256

[6] Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval. Volume 39. Cambridge University Press (2008)

[7] Thomas, S.W., Hassan, A.E., Blostein, D.: Mining unstructure software repositories. In Mens, T., Serebrenik, A., Cleve, A., eds.: Evolving Software Systems. Springer, Berlin, Heidelberg (2014) 139–162

[8] Thomas, S.W., Nagappan, M., Blostein, D., Hassan, A.E.: The impact of classifier configuration and classifier combination on bug localization. IEEE Transactions on Software Engineering **39**(10) (2013) 1427–1443

[9] Weber, R., Aha, D.W., Becerra-Fernandez, I.: Intelligent lessons learned systems. Expert Systems with Applications **20**(1) (2001) 17–34

[10] Weber, R., Aha, D.W., Branting, K., Lucas, J.R., Becerra-Fernandez, I.: Active case-based reasoning for lessons delivery system. In: Proceedings of the Florida Artificial Intelligence Research Society (FLAIRS) Conference. (2000) 170–174

[11] Weber, R.O., Aha, D.W.: Intelligent delivery of military lessons learned. Decision Support Systems **34**(3) (2003) 287–304