An Empirical Study of BM25 and BM25F Based Feature Location Techniques

Zhendong Shi[1, 2], Jacky Keung[2], Qinbao Song[1]

[1]Department of Computer Science and Technology
Xi'an Jiaotong University
Xi'an, China

[2]Department of Computer Science
City University of Hong Kong
Hong Kong, China

zdshi0@stu.xjtu.edu.cn; jacky.keung@cityu.edu.hk; qbsong@mail.xjtu.edu.cn

ABSTRACT
Feature location is a software comprehension activity which aims at identifying source code entities that implement functionalities. Manual feature location is a labor-insensitive task, and developers need to find the target entities from thousands of software artifacts. Recent research has developed automatic and semiautomatic methods mainly based on Information Retrieval (IR) techniques to help developers locate the entities which are textually similar to the feature. In this paper, we focus on individual IR-based methods and try to find a suitable IR technique for feature location, which could be chosen as a part of hybrid methods to achieve good performance. We present two feature location approaches based on BM25 and its variant BM25F algorithm. We compared the two algorithms to the Vector Space Model (VSM), Unigram Model (UM), and Latent Dirichlet Allocation (LDA) on four open source projects. The result shows that BM25 and BM25F are consistently better than other IR methods such as VSM, UM and LDA on the four selected software systems in their best configurations respectively.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *documentation, enhancement*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Retrieval models.*

General Terms
Algorithms, Experimentation

Keywords
Feature Location, Bug Localization, Software Maintenance, Information Retrieval, BM25, BM25F.

1. INTRODUCTION
During the software development process and maintenance, developers usually need to implement new features, improve existing functionalities, and fix bugs. The aim of feature location is to help developers quickly find which part of source code needs to be added or modified, to implement the required functionalities or remove the undesired functionalities like bugs.

InnoSWDev'14, November 16, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3226-2/14/11...$15.00
http://dx.doi.org/10.1145/2666581.2666594

Feature location is defined as identifying an initial location in the source code corresponding to a specific functionality. Many feature location techniques used bug reports as features in case studies, since the feature functionality descriptions in requirement documents are usually hard to obtain. Some researchers use bug localization instead of feature location when the features of interest are bug reports [15, 23, 26, 29, 31, 33].

The functionalities in feature location include implementing new features, improving existing functions, and removing bugs which are similar to removing unwanted functionalities [9]. No matter what kinds of data are used (i.e., bug reports, patch descriptions, enhancement notes, or other feature descriptions), they are all treated as queries in the IR techniques based location task. In the case study of this paper, most of features in the dataset are bug reports, and the others are patch and enhancement descriptions.

There are mainly three types of analysis used in feature location: dynamic, static and textual methods. Some studies have combined two or the whole of those three techniques to get a better result than using the individual technique [9]. Dynamic methods gather information during the execution of programs for analysis. Static methods extract the structure like control or dependence graph from the source code to help developers analyze where the feature is located. Textual methods, i.e. IR techniques, analyze the code text including identifiers and comments. Rubin and Chechik [24] divided the types of feature location analysis into two categories: dynamic and static, according to whether the program is executed in the location process or not.

The basis of information retrieval techniques used in feature location is utilizing the similarities between feature terms and source code artifacts to rank software entities according to their relevance to the feature query. Some IR models like Latent Semantic Indexing (LSI) [18, 21] and Latent Dirichlet Allocation (LDA) [15, 16] have been applied in the area of feature location. In this paper, we only focus on information retrieval based techniques and compare the feature location performance of different IR methods. Many hybrid methods introduced in Section 3 combined textual methods and other information from historical data, static and dynamic analysis results. The hybrid methods usually achieved better performance than the individual methods. Choosing a suitable IR method in these hybrid techniques is an important issue to achieve good performance in feature location.

BM25 [25] is a popular ranking method in information retrieval, which uses two parameters to control the weight of term frequencies and document lengths. Its variant BM25F [36] divides the documents into different fields and assigns different scaling parameters to terms in each field. BM25 has previously been applied in duplication detection for software reports [35] and used

as a weighting scheme in a structured source code based bug localization approach [26]. We have conducted experiments on a feature location benchmark containing 4 open source projects [9] and show that BM25 family algorithms achieved the best results when compared to other three IR methods in their best configurations.

2. BACKGOURND

This section we present how IR techniques perform in feature location. The general framework of the IR method used in feature location is shown in Figure 1. Software source code and features like bug reports are inputs of this framework. The goal of this IR system is to find software artifacts in the source code relevant to the feature. According to different granularity setting, the retrieving source code level can be method/function, class, or file level. In this framework, the source code elements are treated as documents in the search engine. A document is a method/function, a class, or a file depending on the predetermined granularity. The output of this framework is a ranked list of documents. Higher ranks of the relevant documents usually mean better performance of feature location.

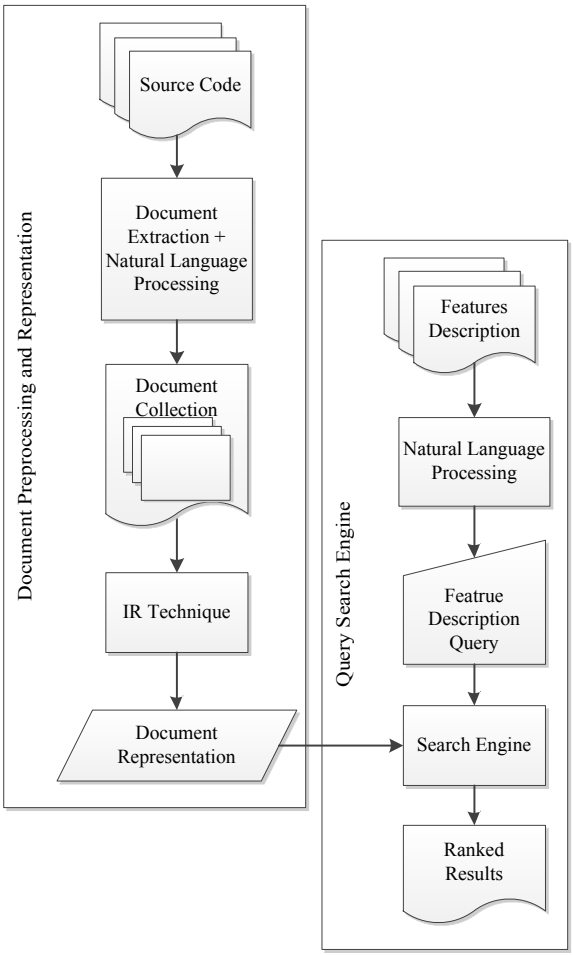

Figure 1. A general IR framework in feature location task

The framework can be mainly divided into two parts. The first part is document preprocessing and representation, which is shown in the left of Figure 1. After inputting the source code in this part, the source code terms are extracted from every document firstly. Usually, terms in identifiers and comments will be extracted. Then several Natural Language Processing (NLP) steps are applied to the extracted terms in every document, which include removal of stop words, splitting multi-words and stemming. The stop words which need to be removed include a general English stop words list, programming language keywords and punctuation. Multi-word identifiers like "OpenFile" are split into single terms "open" and "file" using techniques like CamelCase [8]. Then the stemmer like Porter [20] is applied on the terms to obtain their root words. The influence of choosing only identifiers or comments, and choosing different combination of preprocessing steps have been researched [31].

After preprocessing the raw text from source code, the documents of source code elements are usually represented as term frequency vectors, i.e. bag-of-words. This model assumes there is no order of words in the text [17]. All textual methods talked here are under this assumption. The document representation vectors differ based on different IR techniques. For example, one of documents representation in Vector Space Model (VSM) is *tf-idf* (term frequency-inverse document frequency), which considers the term occurring counts in a document and the number of document including this term. In LDA [5], each document is represented by a document-topic probability distribution, while each topic is represented by a topic-term probability distribution.

In the second part of the framework, the query search engine preprocesses the queries by the same as source code documents done. The similarity between a query and a document is depending on which IR technique is used. This IR system utilizes the text similarity between the text in the source code elements and the text description of the feature to give a ranked list of documents. Developers are expected to find the target entity quickly according to the ranked list.

3. PREVIOUS WORK

LSI (Latent Semantic Indexing) [18, 21] and LDA (Latent Dirichlet Allocation) [15, 16] have been applied in feature location. LSI [7] is an indexing and retrieval method that uses Singular Value Decomposition (SVD) to extract the latent semantic vectors from the term-by-document matrix. The documents in the LSI model are represented as latent semantic vectors. Poshyvanyk and Marcus [22] used FCA (Formal Concept Analysis) to organize and represent the ranked list methods from LSI-based feature location. LDA [5] is a topic model which assumes there is a latent topic level between documents and words. The LDA model uses topic distribution to represent the documents, while each topic is a multinomial distribution over terms.

Several papers [1, 19, 23, 31, 34] have shown the performance of simple IR models like Vector Space Model (VSM) and Unigram Model (UM) were better than sophisticated models like LSI and LDA in feature location significantly. So we choose the VSM and UM as the main comparison techniques. One reason of this result may be these papers chose the whole feature description as the query, while LSI [18, 21] and LDA [15, 16] work on feature location used the manual selected queries.

In this paper, we present how we utilized BM25 and BM25F to locate features in software. The result of our experiment shows that when all terms of feature descriptions are treated as queries without automatic or manual reformulation, BM25 and BM25F achieved better performance than UM, VSM and LDA, while the UM and VSM models performed good when using only

information retrieval techniques in previous work [1, 19, 23, 31, 34].

BLUiR [26] is a closed work related to this paper. It improved bug localization using structured information retrieval with the BM25 weighting scheme. The authors divided the source code terms into four fields and the query terms into two fields and calculate the similarity for each field combination then sum all scores.

Some hybrid techniques which combine information retrieval methods like LSI, LDA and other information such as source code structure [2, 27, 28], dynamic analysis [10, 14, 21] and version history [13, 29, 37] achieved good performance than only using information retrieval methods. But these hybrid methods usually need more input data rather than source codes text, and sometimes the additional input cannot provide enough information to improve the performance, for example, the version history of a new developed system.

3.1 Feature Location vs. Bug Localization

In the opinion of some researchers, bugs are treated as unwanted features [4, 9]. In their view, bug localization task is a subarea of feature location, and feature location could be called bug localization when all features are bugs. In some feature location approaches [3, 10, 30], bug reports are used as the main input features. But some other researchers [26, 31, 33] considered feature location as a different area from bug localization, and thought features do not contain bugs. In their opinion, features only indicate the functionalities or concepts that developers want to implement, and features usually contain few terms like only one term, while bug reports often contain a lot of terms. In this paper, we use the former definition, i.e., feature location includes bug localization. We do not use the phrase bug localization as our topic, since not all features in this case study are bug reports.

4. BM25-BASED APPROACH TO FEATURE LOCATION

The BM25 and BM25F approaches in feature location are introduced in this section.

4.1 BM25

BM25 [25] is a bag-of-words ranking function implemented in Okapi information retrieval system. Each query term has a score which depends on the occurrence of this term in all documents, regardless of the inter-relationship between the query terms within a document. There are two parameters in BM25 to control the scale of both term frequencies and document lengths. So the influence of frequent terms and long documents to the final result would be calibrated according to their term frequencies and document lengths.

Given a query Q with terms q_1, q_2, ..., q_n, the BM25 score of a document D is:

$$Score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{tf(q_i,D)(k_1+1)}{tf(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)} \quad (1)$$

Here, $tf(q_i, D)$ is the term frequency of q_i in the document D, $|D|$ is the length of document D, and $avgdl$ is the average of document lengths in the whole corpus. The term containing $|D|$, $avgdl$, and b indicates the normalization of document frequency. Free parameters k_1 and b control the scale of term frequency and document length respectively. While $k_1 = 0$, it is a binary model (no consideration of term frequency). A b value of 0 corresponds to no length normalization. $IDF(q_i)$ is the Inverse Document Frequency (IDF) of term q_i in the whole corpus:

$$IDF(q_i) = log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \quad (2)$$

Here, N is the total number of documents, $n(q_i)$ is the number of documents contain the term q_i. 0.5 is a smoothing constant to deal with the situation $n(q_i) = 0$. The IDF value would be negative when $n(q_i)$ is greater than half of N. So the query term which has occurred in larger than half of all documents is ignored in the algorithm.

Because the feature queries in the case study are usually long documents, a normalizing scheme is used to deal with the term counts in a query as follows [17]:

$$Score(D,Q) = \sum_{i=1}^{n} \frac{(k_3+1)tf(q_i,Q)}{k_3 + tf(q_i,Q)} IDF(q_i) \frac{tf(q_i,D)(k_1+1)}{tf(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)} \quad (3)$$

where $tf(q_i,Q)$ is the term frequency of q_i in the query Q, and k_3 is the weighting parameter which calibrates term frequency scaling of the query. The other part is the same as Equation (1). In our research, we tried several values for the three parameters k_1, b and k_3 to get a good result.

After the scores between a query and all documents are derived by Equation (3), every document is ranked according to the corresponding score to the query. Documents with high scores are ranked in the top of final result, which means these documents are more relevant to the query.

4.2 BM25F

BM25F [36] is a variant of BM25 in which the terms in a document are classified into several fields with different degrees of importance. The original BM25F considered that the webpage text is composed from title text, main body text and anchor text.

A field-dependent normalized term frequency $tf'(t, f, D)$ of a term t which is considered in the field f of the document D is calculated as follows:

$$tf'(t,f,D) = \frac{tf(t,f,D)}{1 - b_f + b_f\frac{l(f,D)}{avgl(f)}} \quad (4)$$

where $tf(t, f, D)$ is the original term frequency of a term t occurs in the field f of the document D, $l(f, D)$ is the total number of terms in the field f of the document D, $avgl(f)$ is the average field f length in the whole corpus, and b_f is a field-dependent parameter for scaling field length like b in BM25.

The final *pseudo-frequency* of term t in the document D is a linear weighted sum of $tf'(t,f,D)$ from different fields:

$$tf'(t,D) = \sum_f w_f \cdot tf'(t,f,D) \quad (5)$$

where w_f is a field-dependent weight parameter. The larger value of w_f means higher importance of the corresponding field.

Then the score between the query Q with terms q_1, $q_{2,...,}$ q_n and a document D is calculated as follows:

$$Score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{tf'(t,D)}{k_1 + tf'(t,D)} \quad (6)$$

where $IDF(q_i)$ is calculated by Equation (2) like BM25, and k_1 is the term frequency scaling parameter.

For the reason of long feature description, we use the same query term normalizing scheme as Equation (3) in BM25 as follows:

$$Score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{tf'(t,D)}{k_1 + tf'(t,D)} \frac{(k_3+1)tf(q_i,Q)}{k_3 + tf(q_i,Q)} \quad (7)$$

At last, the documents are ranked according to the scores to a specific query like BM25.

For simplicity, we divide the source code into two fields for BM25F: method invocations text and the other text. The field of method invocations only includes the terms from the invoked method names, and it is called *invocation field*. The other terms in the class are in another field called *main field*. This division of source codes fields is a primary trial. We leave the experiment with more fields to the future work. The parameters of both fields are named as follows:

Table 1. Two-field BM25F parameters in feature location

Field \Parameter	b_f	w_f
Main	b_1	w_1
Invocation	b_2	w_2

Apart from the four parameters in Table 1, there are two other parameters k_1, k_3. Because we only care the relative value of the weighting parameter w_f instead of the absolute value, w_1 is set to 1 in all parameter setting while different values of w_2 can refer to different relative weight. So in this two fields based BM25F scheme, there are totally five parameters to be determined before running the algorithm.

5. CASE STUDY

We compare the performance of feature location using VSM, UM, LDA, BM25 and BM25F on four open source software systems. Simple methods VSM and UM were reported to have a better performance than other sophisticated methods like LDA in feature location. The result of this case study shows that BM25 and BM25F outperform other three methods in their best configuration respectively.

5.1 Data Set

We use four open source softwares: ArogUML[1], JabRef[2], jEdit[3] and muCommmander[4]. The features including bug reports, patch notes and enhancement descriptions of the four systems and the gold set which is the set of modified methods relevant to the given feature are from the Benchmarks [9]. They are available online[5].

[1] http://argouml.tigris.org/

[2] http://jabref.sourceforge.net/

[3] http://www.jedit.org/

[4] http://www.mucommander.com/

[5] http://www.cs.wm.edu/semeru/data/benchmarks/

Table 2 lists size metrics of four projects and number of features for each system. LOC stands for lines of codes.

Table 2. Metrics of Software systems

System	Version	LOC	Classes	Features
ArgoUML	0.22	131020	1474	91
JabRef	2.6b	74138	590	39
jEdit	4.3	106481	513	150
muCommander	0.8.5	76649	1069	92

Since not all the gold methods in the benchmark, which is the relevant methods to the specific query, exist in the modified version of software, we set the feature location granularity in class level, and all the gold methods are changed to classes. Then every gold class can be found in the modified version of software.

5.2 Preprocessing

JRipples [6] is an Eclipse plug-in supporting feature location. We modified JRipples to extract two fields terms of each class from the dataset for BM25F techniques and whole text of each class for other four algorithms. Thomas et al. [31] recommended to use all text information of bug reports and software entities. In our experiments, bug titles, descriptions, software entities text includes identifiers, comments and so on are considered as the source of text.

We removed a list of English stop words and Java key words. The terms having less than 4 characters or longer than 14 characters were removed. We split multi-word terms based on camel case and underscores using splitting tool in JRipples. The Porter stemmer [20] is used to get term stems.

5.3 Setting

We used R lda v1.3.2 [6] package to run LDA models. We implemented VSM, UM, BM25 and BM25F using R language by ourselves.

5.3.1 Vector Space Model

We used the regular *tf-idf* weight in Vector Space Model (VSM) as follows [17]:

$$weight(w_i,d) = \frac{tf(w_i,D)}{max_j tf(w_j,D)} \log \frac{N}{df(w_i)} \quad (8)$$

Here, $tf(w_i, D)$ is the term frequency of term w_i in the document D, which is normalized by the maximum term frequency in the document D. N is the number of total documents, and $df(w_i)$ is the document frequency of term w_i which is the number of documents containing term w_i.

After documents are represented as a vector of weight value for each term, the similarity between the query and the document is computed by *overlap score measure*, which is the sum of *tf-idf* weights of terms occurs in the query [17]:

$$Sim(Q,D) = \sum_{q_i \in Q} weight(q_i,d) \quad (9)$$

Because most of the queries are long feature descriptions, we used similar query normalizing scheme as Equation (3):

[6] http://cran.r-project.org/web/packages/lda/index.html

$$Sim(Q,D) = \sum_{q_i \in Q} weight(q_i, d) \frac{(k+1)tf(q_i,Q)}{k + tf(q_i,Q)} \quad (10)$$

The query scaling parameter k_3 is set from 0 to 5 with interval 0.5 in our experiment.

5.3.2 Unigram Model

Unigram Model (UM) is a language model which employs a single multinomial distribution over terms for each document. The probability of each term is calculated as the maximum likelihood of each term [17]. We used UM with Jelinek-Mercer smoothing [12] to avoid zero probability of query term which do not occur in documents. The probability of term w_i in document D with smoothing is calculated as follows:

$$P(w_i|D) = u\frac{tf(w_i,D)}{\sum_{j=1}^{|D|} tf(w_j,D)} + (1-u)\frac{tf(w_i,C)}{\sum_{j=1}^{|C|} tf(w_i,C)} \quad (11)$$

Here, $tf(w_i, D)$ is the term frequency of term w_i in the document D, $tf(w_i, C)$ is the term frequency of term w_i in the whole corpus C. The denominators of two terms in Equation (11) are the total term number in the document D and the corpus C. The parameter u is used to weight the UM probability from the document and the whole corpus, and the value of u is set from 0 to 0.95 with interval 0.05 in our experiment. The similarity between the query and the document is the product of the likelihoods of each term in the query [17]:

$$Sim(Q,d_i) = P(Q|d_i) = \prod_{q_k \in Q} P(q_k|d_i) \quad (12)$$

where q_k is the k^{th} word in the query Q, d_i represents the document. The query terms which do not occur in the whole document corpus are ignored.

5.3.3 Latent Dirichlet Allocation

The Latent Dirichlet Allocation (LDA) [5] model has three parameters, the number of topics K, and two super parameters α and β. Biggers et al. [4] have studied how the configurations of parameters in LDA influence the feature location performance on these four software systems. Since they studied configuration in method-level which is different from our setting, we tried different parameter values for LDA as shown in Table 3. The similarity between the query and the document is also calculated by Equation (12), which followed the LDA work in feature location [15, 16]. R-lda package is based on Gibbs sampling [11] to infer the model. We set Gibbs sampling iteration 2000 for all experiments to make the LDA model convergence.

Table 3. LDA parameters setting

Parameter	Value
α	0.1, 0.25, 0.5, 1
β	0.1, 0.25, 0.5, 1.
K	50, 100, 150, 200

5.3.4 BM25 and BM25F

We also tried different combination of parameter values for BM25 and BM25F. The BM25 and BM25F parameters setting are shown in Table 4 and Table 5 respectively. Because BM25F has more parameters than BM25, we set parameters of BM25F in longer intervals than BM25. The total number of parameters combination in BM25 is 891, while the total number of parameters combination in BM25F is 6300.

Table 4. BM25 parameters setting

Parameter	Value
k_1	1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5
b_1	0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1
k_q	1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5

Table 5. BM25F parameters setting

Parameter	Value
k_1	1, 2, 3, 4, 5
b_1	0.5, 0.6, 0.7, 0.8, 0.9, 1
b_2	0.5, 0.6, 0.7, 0.8, 0.9, 1
w_1	1
w_2	0.1, 0.2, 0.5, 1, 2, 5, 10
k_q	1, 2, 3, 4, 5

5.4 Evaluation Metric

We used Mean Reciprocal Rank (MRR) [32] to evaluate the result of feature location. It is a statistic measure for evaluating an algorithm which produces a list of possible responses to a query. The reciprocal rank of a query in feature location is the multiplicative inverse of the effective measure. Effective Measure (EM) is the highest rank of source code entities implementing the query feature in the final ranked list [21]. The mean reciprocal rank is the average of the reciprocal ranks for a set of queries QS:

$$MRR = \frac{1}{|QS|}\sum_{i=1}^{|QS|}\frac{1}{EM_i} \quad (13)$$

The higher value of the MRR metric indicates the better performance of the retrieval algorithm.

5.5 Result and Analysis

The MRR results of 5 methods with best four and worst four configurations are shown in Table 6. The best configuration and MRR metric of each method in each project is in bold type, and the rank is based on MRR values. The parameter configurations are ranked by the MRR value. The MRR of the best configurations of BM25 and BM25F are similar for each project, and they are higher than the best configurations of UM, VSM and LDA for each project. A more clear comparison of best configurations for each method and project is shown in Figure 2.

As shown in Table 6(A), the parameter k in VSM has little influence on the performance. The parameter k value in VSM is set from 1.5 to 3.5 to get best performance for ArgoUML, JabRef and jEdit projects, while $k = 0$ in muCommander performs best in which the query term frequency is totally ignored. The performances of UM in different parameter configurations show that a large weight of document generating probability with a small weight of collection generating probability is reasonable in UM-based feature location. When $u = 0$, the scores of terms only

count the collection generating probabilities with no document discrimination, so it is the worst configuration in UM.

The performance of the LDA model varies according to the different parameters. The hyperparameters setting $\alpha = 1$, $\beta = 0.25$ are reasonable choice for all systems except the muCommander project. 150 or 200 topic numbers exist mostly in the best four configurations. Because the inference procedure in LDA is probabilistic based iteration sampling, the result varies every time. The MRR results of LDA in Table 6(A) are only calculated based on running the LDA model once, but 64 LDA results with different parameter values show that the performance of LDA is worse than others except in the JabRef project.

BM25 and BM25F have similar MRR values in feature location with the best four and worst four configurations. It may due to only considering two simple fields of source codes. The term frequency scaling parameters k_1 in two algorithms also have similar values for each project as shown in Table 6(B). For example, $k_1 = 4$ is suitable for both BM25 and BM25F in the jEdit project. In the setting range, larger k_q value (k_q = 4, 5) is reasonable for both algorithms in four projects, which indicates some terms in the query with large frequency need to be scaled.

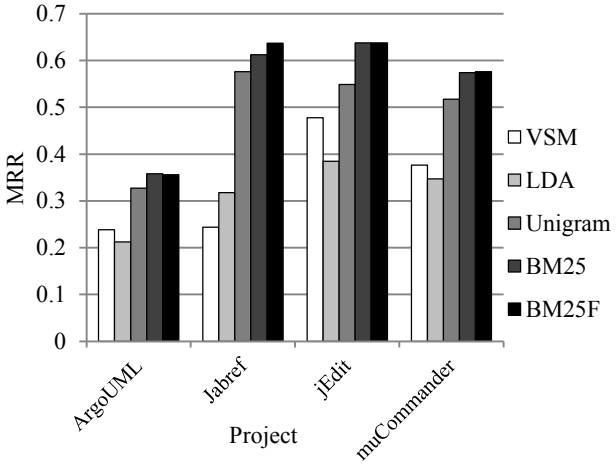

Figure 2. MRR of IR techniques in best configuration

Table 6(A). The best four and worst four configurations of VSM, UM and LDA in feature location

VSM			UM			LDA		
Rank	Configuration	MRR	Rank	Configuration	MRR	Rank	Configuration	MRR
ArgoUML								
1	**k = 1.5**	**0.2382**	1	**u = 0.9**	**0.3271**	1	**α = 1, β = 0.5, K = 150**	**0.2125**
2	k = 1	0.2357	2	u = 0.95	0.3244	2	α = 1, β = 0.25, K = 150	0.1940
3	k = 2	0.2336	3	u = 0.85	0.3204	3	α = 1, β = 0.1, K = 150	0.1931
4	k = 4	0.2305	4	u = 0.8	0.3149	4	α = 1, β = 0.5, K = 200	0.1898
8	k = 0.5	0.2275	17	u = 0.15	0.2356	61	α = 0.25, β = 1, K = 150	0.0901
9	k = 3.5	0.2264	18	u = 0.1	0.2281	62	α = 1, β = 0.1, K = 50	0.0895
10	k = 5	0.2253	19	u = 0.05	0.2138	63	α = 0.1, β = 1, K = 50	0.0886
11	k = 0	0.2226	20	u = 0	0.0014	64	α = 0.1, β = 1, K = 100	0.0621
JabRef								
1	**k = 3.5**	**0.2438**	1	**u = 0.8**	**0.5763**	1	**α = 0.5, β = 0.25, K = 150**	**0.3177**
2	k = 5	0.2412	2	u = 0.85	0.5702	2	α = 0.1, β = 0.1, K = 200	0.2920
3	k = 4.5	0.2397	3	u = 0.7	0.5606	3	α = 1, β = 0.1, K = 100	0.2784
4	k = 3	0.2378	4	u = 0.75	0.5580	4	α = 0.25, β = 0.1, K = 100	0.2740
8	k = 1.5	0.2331	17	u = 0.15	0.2357	61	α = 0.1, β = 1, K = 200	0.0841
9	k = 1	0.2186	18	u = 0.1	0.2281	62	α = 0.5, β = 1, K = 150	0.0838
10	k = 0.5	0.2085	19	u = 0.05	0.2138	63	α = 0.1, β = 1, K = 150	0.0834
11	k = 0	0.1906	20	u = 0	0.0014	64	α = 0.1, β = 0.5, K = 100	0.0579
jEdit								
1	**k = 2**	**0.4778**	1	**u = 0.8**	**0.5488**	1	**α = 1, β = 0.25, K = 200**	**0.3847**
2	k = 2.5	0.4732	2	u = 0.75	0.5467	2	α = 1, β = 0.1, K = 200	0.3746
3	k = 3	0.4691	3	u = 0.85	0.5431	3	α = 0.5, β = 0.1, K = 200	0.3690
4	k = 3.5	0.4688	4	u = 0.9	0.5427	4	α = 1, β = 0.1, K = 150	0.3604
8	k = 1.5	0.4628	17	u = 0.15	0.4300	61	α = 0.1, β = 1, K = 200	0.2103
9	k = 5	0.4619	18	u = 0.1	0.3997	62	α = 0.1, β = 1, K = 50	0.2008
10	k = 0.5	0.4462	19	u = 0.05	0.3836	63	α = 0.25, β = 1, K = 150	0.1966
11	k = 0	0.4262	20	u = 0	0.0039	64	α = 0.1, β = 1, K = 100	0.1859
muCommander								
1	**k = 0**	**0.3765**	1	**u = 0.85**	**0.5174**	1	**α = 0.5, β = 0.1, K = 200**	**0.3467**
2	k = 0.5	0.3645	2	u = 0.9	0.5170	2	α = 0.1, β = 0.1, K = 200	0.3207
3	k = 1	0.3617	3	u = 0.95	0.5112	3	α = 0.25, β = 0.1, K = 150	0.3177
4	k = 2	0.3580	4	u = 0.8	0.5082	4	α = 0.1, β = 0.1, K = 150	0.3136
8	k = 4.5	0.3557	17	u = 0.15	0.3822	61	α = 0.1, β = 1, K = 50	0.1460
9	k = 3	0.3537	18	u = 0.1	0.3510	62	α = 0.1, β = 1, K = 150	0.1450
10	k = 3.5	0.3515	19	u = 0.05	0.3188	63	α = 0.25, β = 1, K = 150	0.1257
11	k = 2.5	0.3515	20	u = 0	0.0019	64	α = 0.1, β = 1, K = 200	0.1194

Table 6(B). The best four and worst four configurations of BM25 and BM25F in feature location

BM25			BM25F		
Rank	Configuration	MRR	Rank	Configuration (w_1=1)	MRR
ArgoUML					
1	**k_1 = 1.5, b = 0.75, k_q = 4.5**	**0.3577**	1	**k_1 = 3, b_1 = 0.5, b_2 = 1, w_2 = 0.5, k_q = 4**	**0.3561**
2	k_1 = 1.5, b = 0.75, k_q = 5	0.3556	2	k_1 = 3, b_1 = 0.5, b_2 = 0.6, w_2 = 0.2, k_q = 4	0.3560
3	k_1 = 1, b = 0.9, k_q = 2.5	0.3549	3	k_1 = 3, b_1 = 0.5, b_2 = 0.5, w_2 = 0.2, k_q = 4	0.3560
4	k_1 = 3, b = 0.55, k_q = 5	0.3522	4	k_1 = 3, b_1 = 0.5, b_2 = 0.7, w_2 = 0.2, k_q = 4	0.3560
888	k_1 = 5, b = 1, k_q = 4.5	0.2416	6297	k_1 = 5, b_1 = 1, b_2 = 0.9, w_2 = 5, k_q = 4	0.2221
889	k_1 = 5, b = 1, k_q = 5	0.2416	6298	k_1 = 5, b_1 = 1, b_2 = 1, w_2 = 5, k_q = 4	0.2221
890	k_1 = 4.5, b = 1, k_q = 1	0.2385	6299	k_1 = 5, b_1 = 1, b_2 = 0.9, w_2 = 5, k_q = 1	0.2210
891	k_1 = 5, b = 1, k_q = 1	0.2353	6300	k_1 = 5, b_1 = 1, b_2 = 0.8, w_2 = 5, k_q = 1	0.2271
JabRef					
1	**k_1 = 1, b = 0.65, k_q = 4.5**	**0.6124**	1	**k_1 = 3, b_1 = 0.5, b_2 = 0.5, w_2 = 5, k_q = 4**	**0.6369**
2	k_1 = 1, b = 0.75, k_q = 4	0.6013	2	k_1 = 3, b_1 = 0.5, b_2 = 0.1, w_2 = 10, k_q = 4	0.6368
3	k_1 = 1, b = 0.8, k_q = 5	0.6013	3	k_1 = 3, b_1 = 0.5, b_2 = 0.9, w_2 = 10, k_q = 4	0.6368
4	k_1 = 1, b = 0.6, k_q = 4.5	0.6005	4	k_1 = 3, b_1 = 0.5, b_2 = 0.7, w_2 = 10, k_q = 4	0.6347
888	k_1 = 4, b = 1, k_q = 1	0.4286	6297	k_1 = 5, b_1 = 1, b_2 = 0.5, w_2 = 1, k_q = 1	0.4167
889	k_1 = 5, b = 0.95, k_q = 1	0.4267	6298	k_1 = 5, b_1 = 1, b_2 = 0.8, w_2 = 2, k_q = 1	0.4166
890	k_1 = 4.5, b = 1, k_q = 1	0.4192	6299	k_1 = 5, b_1 = 1, b_2 = 0.9, w_2 = 2, k_q = 1	0.4166
891	k_1 = 5, b = 1, k_q = 1	0.4167	6300	k_1 = 5, b_1 = 1, b_2 = 1, w_2 = 2, k_q = 1	0.4166
jEdit					
1	**k_1 = 4, b = 0.55, k_q = 4.5**	**0.6376**	1	**k_1 = 4, b_1 = 0.5, b_2 =0.5, w_2 = 0.5, k_q = 5**	**0.6373**
2	k_1 = 4, b = 0.55, k_q = 5	0.6367	2	k_1 = 4, b_1 = 0.5, b_2 =0.8, w_2 = 1, k_q = 5	0.6373
3	k_1 = 4.5, b = 0.5, k_q = 5	0.6366	3	k_1 = 4, b_1 = 0.5, b_2 =0.8, w_2 = 0.5, k_q = 5	0.6373
4	k_1 = 4, b = 0.55, k_q = 4	0.6363	4	k_1 = 4, b_1 = 0.5, b_2 =1, w_2 = 0.5, k_q = 5	0.6373
888	k_1 = 3.5, b = 1, k_q = 1	0.5506	6297	k_1 = 4, b_1 = 1, b_2 =0.6, w_2 = 10, k_q = 1	0.5203
889	k_1 = 5, b = 1, k_q = 1	0.5488	6298	k_1 = 5, b_1 = 1, b_2 =0.7, w_2 = 10, k_q = 1	0.5179
890	k_1 = 4.5, b = 1, k_q = 1.5	0.5484	6299	k_1 = 5, b_1 = 1, b_2 =0.8, w_2 = 10, k_q = 1	0.5174
891	k_1 = 4.5, b = 1, k_q = 1	0.5431	6300	k_1 = 5, b_1 = 1, b_2 =0.9, w_2 = 10, k_q = 1	0.5170
muCommander					
1	**k_1 = 3, b = 0.55, k_q = 5**	**0.5743**	1	**k_1 = 4, b_1 = 0.5, b_2 = 0.5, w_2 = 2, k_q = 3**	**0.5760**
2	k_1 = 4, b = 0.5, k_q = 2.5	0.5737	2	k_1 = 4, b_1 = 0.5, b_2 = 0.6, w_2 = 2, k_q = 3	0.5741
3	k_1 = 4.5, b = 0.5, k_q = 2.5	0.5727	3	k_1 = 4, b_1 = 0.5, b_2 = 0.8, w_2 = 2, k_q = 3	0.5732
4	k_1 = 2.5, b = 0.6, k_q = 4.5	0.5600	4	k_1 = 4, b_1 = 0.5, b_2 = 0.7, w_2 = 2, k_q = 3	0.5732
888	k_1 = 5, b = 1, k_q = 3.5	0.4425	6297	k_1 = 5, b_1 = 1, b_2 = 0.5, w_2 = 2, k_q = 1	0.4345
889	k_1 = 5, b = 1, k_q = 5	0.4413	6298	k_1 = 5, b_1 = 1, b_2 = 0.7, w_2 = 2, k_q = 1	0.4343
890	k_1 = 5, b = 1, k_q = 1.5	0.4390	6299	k_1 = 5, b_1 = 1, b_2 = 0.6, w_2 = 2, k_q = 1	0.4342
891	k_1 = 5, b = 1, k_q = 1	0.4370	6300	k_1 = 5, b_1 = 1, b_2 = 1, w_2 = 5, k_q = 4	0.4341

In BM25, b = 1, which means the full scaling of document length, exists in the worst four configurations of all four projects, indicating that full document length normalizing is not suitable in this case study. In BM25F, b_1 = 0.5, which is the lowest value of the corresponding setting range, existed in all best four configurations of all projects, while b_1 = 1, which is the highest value of the setting range, existed in all worst four configurations of all projects. This indicates that, for the main field, little scaling on the field length is enough. Since we did not test b_1 value lower than 0.5, the future work needs to focus on more parameter configurations. The field weight parameter w_2 has different suitable values in different projects. There is no obvious trend that the methods invocation field should be weighted higher or lower than the main field.

5.6 Threats to Validity

There are several threats to the validity of our results. First, we used several parameter values for each IR method in our case study, but we cannot set so many parameters without knowing the relevant entities of queries in practice. Some reasonable parameter settings of five methods for our selected four systems have been discussed in the previous section, but we cannot assure they are suitable in other software systems.

Although in each best configuration for the five methods, BM25 and BM25F achieve better performance than other three methods, the parameter tuning job in BM25 family algorithms is still a challenge, even we know the previous features and relevant entities of a system to learn a reasonable parameter setting. For BM25F, one choice is using the original BM25F parameter optimizing procedure [36], which is a local optimizing process and needs to know the relevant entities of features in advance. An empirical parameter recommendation of BM25 algorithm in feature location still needs to be studied in the future work.

The evaluation metric we used MRR is based on the effective measure, which is the rank of first relevant entity in the returned list. Because there are usually multiple entities relevant to a feature, the rank of other entities except the top rank one are not considered in this evaluation metric. Nevertheless, this metric is widely used in feature location and allow different techniques to be compared with each other.

Because the amount of computation is quite large, we did not use large number of parameter values. Some parameters are set in the smallest or largest values to get the best or worst performance. Use more parameters values might get more reasonable results.

6. CONCLUSION AND FUTURE WORK

Searching software entities related to specific functionalities manually cost a lot of time. Several information retrieval based feature location approaches have been proposed to help developers to find the initial location of relevant features. This paper proposes BM25 and BM25F based feature location approaches, and show that they are better than three other regular IR methods, namely VSM, UM, and LDA in four open source projects.

There are a lot of works to do in the future. We only used MRR as the evaluation metric and ranked the results by MRR scores. Some other metrics could be used in the future work: Top-N Rank, which indicate the number of features whose relevant documents are ranked in the top N of the return lists, and Mean Average Precision (MAP), which is the meaning of average precision scores for each query. We considered text terms in method invocations and other terms as two fields in this BM25F based feature location approach. Defining more fields in BM25F may improve the performance. For example, the text terms in class/method names, comments, identifiers, or the other source code could be considered as an individual field respectively in BM25F. Some existing hybrid feature location methods combing IR models and other information may use the BM25 family methods as the IR model part to increase the performance.

7. ACKNOWLEDGEMENT

This research is supported in part by the City University of Hong Kong research fund (Project No. 7200354, 7003032).

8. REFERENCES

[1] Abebe, S.L., Haiduc, S., Tonella, P. and Marcus, A. 2011. The Effect of Lexicon Bad Smells on Concept Location in Source Code. *Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on* (2011), 125–134.

[2] Ali, N., Sabane, A., Gueheneuc, Y. and Antoniol, G. 2012. Improving Bug Location Using Binary Class Relationships. *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on* (2012), 174–183.

[3] Bassett, B. and Kraft, N.A. 2013. Structural information based term weighting in text retrieval for feature location. *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on* (May. 2013), 133–141.

[4] Biggers, L., Bocovich, C., Capshaw, R., Eddy, B., Etzkorn, L. and Kraft, N. 2012. Configuring latent Dirichlet allocation based feature location. *Empirical Software Engineering*. (2012), 1–36.

[5] Blei, D.M., Ng, A.Y. and Jordan, M.I. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*. 3, 4-5 (2003), 993–1022.

[6] Buckner, J., Buchta, J., Petrenko, M. and Rajlich, V. 2005. JRipples: A Tool for Program Comprehension During Incremental Change. *Proceedings of the 13th International Workshop on Program Comprehension* (Washington, DC, USA, 2005), 149–152.

[7] Deerwester, S. and Deerwester, S. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*. 41, 6 (1990), 391–407.

[8] Dit, B., Guerrouj, L., Poshyvanyk, D. and Antoniol, G. 2011. Can Better Identifier Splitting Techniques Help Feature Location? *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* (2011), 11–20.

[9] Dit, B., Revelle, M., Gethers, M. and Poshyvanyk, D. 2013. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*. 25, 1 (2013), 53–95.

[10] Dit, B., Revelle, M. and Poshyvanyk, D. 2013. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Software Engineering*. 18, 2 (2013), 277–309.

[11] Griffiths, T.L. and Steyvers, M. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*. 101 Suppl 1, (2004), 5228–5235.

[12] JELINEK, F. 1980. Interpolated estimation of Markov source parameters from sparse data. *Pattern Recognition in Practice*. (1980).

[13] Kim, D., Tao, Y., Kim, S. and Zeller, A. 2013. Where Should We Fix This Bug? A Two-Phase Recommendation Model. *Software Engineering, IEEE Transactions on*. 39, 11 (Nov. 2013), 1597–1610.

[14] Liu, D., Marcus, A., Poshyvanyk, D. and Rajlich, V. 2007. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering* (Atlanta, Georgia, USA, 2007), 234–243.

[15] Lukins, S.K., Kraft, N.A. and Etzkorn, L.H. 2010. Bug localization using latent Dirichlet allocation. *Information and Software Technology*. 52, 9 (2010), 972–990.

[16] Lukins, S.K., Kraft, N.A. and Etzkorn, L.H. 2008. Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation. *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on* (2008), 155–164.

[17] Manning, C.D., Raghavan, P. and Schütze, H. 2008. *Introduction to information retrieval*. Cambridge university press Cambridge.

[18] Marcus, A., Sergeyev, A., Rajlich, V. and Maletic, J.I. 2004. An information retrieval approach to concept location in source code. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on* (2004), 214–223.

[19] Moreno, L., Bandara, W., Haiduc, S. and Marcus, A. 2013. On the Relationship between the Vocabulary of Bug Reports and Source Code. *Software Maintenance (ICSM), 2013 29th IEEE International Conference on* (2013), 452–455.

[20] Porter, M.F. 1980. An algorithm for suffix stripping. *Program: electronic library and information systems*. 14, 3 (1980), 130–137.

[21] Poshyvanyk, D., Gueheneuc, Y.-G., Marcus, A., Antoniol, G. and Rajlich, V. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *Software Engineering, IEEE Transactions on*. 33, 6 (2007), 420–432.

[22] Poshyvanyk, D. and Marcus, A. 2007. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on* (Jun. 2007), 37–48.

[23] Rao, S. and Kak, A. 2011. Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models. *Proceedings of the 8th Working Conference on Mining Software Repositories* (Waikiki, Honolulu, HI, USA, 2011), 43–52.

[24] Rubin, J. and Chechik, M. 2013. A Survey of Feature Location Techniques. *Domain Engineering*. I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, eds. Springer Berlin Heidelberg. 29–58.

[25] S. E. Robertson, S. Walker S. Jones M. M. HancockBeaulieu and Gatford, M. 1995. Okapi at TREC-3. *Overview of the Third Text REtrieval Conference (TREC–3)* (1995).

[26] Saha, R.K., Lease, M., Khurshid, S. and Perry, D.E. 2013. Improving bug localization using structured information retrieval. *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on* (Nov. 2013), 345–355.

[27] Scanniello, G. and Marcus, A. 2011. Clustering Support for Static Concept Location in Source Code. *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* (2011), 1–10.

[28] Singh, P. and Batra, S. 2012. A Novel Technique for Call Graph Reduction for Bug Localization. *International Journal of Computer Applications*. 47, 15 (Jun. 2012), 150000–5.

[29] Sisman, B. and Kak, A.C. 2012. Incorporating version histories in Information Retrieval based bug localization. *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on* (Jun. 2012), 50–59.

[30] Tantithamthavorn, C., Ihara, A., Hata, H. and Matsumoto, K. 2014. Impact Analysis of Granularity Levels on Feature Location Technique. *Requirements Engineering*. D. Zowghi and Z. Jin, eds. Springer Berlin Heidelberg. 135–149.

[31] Thomas, S.W., Nagappan, M., Blostein, D. and Hassan, A.E. 2013. The Impact of Classifier Configuration and Classifier Combination on Bug Localization. *Software Engineering, IEEE Transactions on*. 39, 10 (2013), 1427–1443.

[32] Voorhees, E.M. 1999. The TREC-8 Question Answering Track Report. *Natural Language Engineering*. E. Voorhees and De. Harman, eds. NIST. 77–82.

[33] Wang, S. and Lo, D. 2014. Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization. *Proceedings of the 22Nd International Conference on Program Comprehension* (Hyderabad, India, 2014), 53–63.

[34] Wang, S., Lo, D., Xing, Z. and Jiang, L. 2011. Concern Localization using Information Retrieval: An Empirical Study on Linux Kernel. *Reverse Engineering (WCRE), 2011 18th Working Conference on* (2011), 92–96.

[35] Yang, C.-Z., Du, H.-H., Wu, S.-S. and Chen, I.-X. 2012. Duplication Detection for Software Bug Reports Based on BM25 Term Weighting. *Technologies and Applications of Artificial Intelligence (TAAI), 2012 Conference on* (Nov. 2012), 33–38.

[36] Zaragoza, H., Zaragoza, H., Craswell, N., Craswell, N., Taylor, M., Taylor, M., Saria, S., Saria, S., Robertson, S. and Robertson, S. 2004. Microsoft Cambridge at TREC-13: Web and HARD tracks. *Proceedings of TREC 2004*. (2004).

[37] Zhou, J., Zhang, H. and Lo, D. 2012. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. *Software Engineering (ICSE), 2012 34th International Conference on* (2012), 14–24.