Empirical Evaluation of The Use of DV and LDA for Feature Location on Java and Python Programs

**Hongxin Chen**

**Matriculation Number: 16250132**

Dissertation 2018

Erasmus Mundus MSc in Dependable Software Systems

Department of Computer Science

Maynooth Univerisy

A dissertation submitted in partial fulfilment

of the requirements for the

Erasmus Mundus MSc Dependable Software Systems

Supervisor: Dr. James Power

19th June 2018

ii

I, Hongxin Chen, declare that this dissertation titled, “Empirical Evaluation of The Use of DV and LDA for Feature Location on Java and Python Programs” and the work presented in it are my own. I confirm that:

* This work was done wholly or mainly while in candidature for a research degree at this University.
* Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
* Where I have consulted the published work of others, this is always clearly attributed.
* Where I have quoted from the work of others, the source is always given. With the exceptions of such quotations, this dissertation is entirely my own work.
* I have acknowledged all main sources of help.
* Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

iii

Abstract

Feature location is the activity of locating artifacts in the source code that related to the implementation of specific software functionalities. Typically, the maintenance and development of dependable software systems will include eliminating undesired functionalities and fixing bugs, as well as adding new features and enhancing existing functionalities. Identifying location of where the concreate features are implemented in the source code is one of the most frequent, general and essential activities undertaken by software engineers. This makes feature location techniques (FLTs) significant to maintenance of dependable software, and thus beneficial to practitioners and researchers. In this thesis, a new methodology of change set generation for source files through mining git commit logs is introduced. Using the generated change set, we evaluate the effectiveness of Document Vector (DV) and Latent Dirichlet Allocation (LDA) techniques for feature location at the level of file documents. DV, an unsupervised framework based on deep learning, appears to be suited for source code as it retains not only the order but also the semantics of words in source code. LDA, a generative probabilistic model, is capable of extracting a set of topics from the code corpora, which could be regarded as representation of features. Results of experiments over 50 Java projects and 50 Python projects collected from github show that LDA outperforms DV in terms of file level feature location.

iv

Acknowledgements

Firstly, I would like to thank my supervisor Dr. James Power of the Department of Computer Science at Maynooth University for his dedication and patient guidance. He is always responsive, patient and never complain about my puzzles even for simple and silly ones. His generous and selfish share on the domain knowledge as well as constructive advices helped me a lot specially when I was stuck.

Then I would like to thank my parents who are far in China. They provide the whole tuition fee and living expenses over my undergraduate career and master programme, which makes me capable of focusing on my study without being disturbed with financial issues. They always encourage me to chase my dream and cheer me up whenever I felt depressed and frustrated. I was not able to complete my study and dissertation without their supports.

At last, I sincerely appreciate all the brilliant professors, lecturers and other staffs in the Maynooth University who make the school such an amazing place for me.

5

Table of Contents

1 Introduction 10

1.2 Contribution 10

1.3 Objectives 10

1.3.1 Primary Objective 10

1.3.2 Secondary Objective 10

1.4 Dissertation Structure 10

2 Background and Related Work 12

2.1 Feature Location Techniques 12

2.2 Latent Dirichlet Allocation 12

2.3 Document Vector 14

2.4 Related Work 15

3 Methodology 18

3.1 Research Questions 18

3.1.1 Automated Goldset Generation 18

Reference 19

Figure 1 Graphical Model Representation of LDA 13

Figure 2 Example of Learning Paragraph Vector 15

Table 1 Dataset from Change History 16

# 1 Introduction

In software systems, a feature stands for a functionality that is described and defined by requirements from system users and stakeholders such as software developers. Feature location is defined as an activity of locating corresponding implementation of functionalities in the source code of software system (Rajlich & Wilde, 2002). Software maintenance is one of the most general but necessary activities performed by software engineers in building system which has strict requirements on dependability. In general, behaviors during software maintenance normally consist of adding new functionalities, upgrade of existing features, bugs fixing and removing undesired functionalities. However, software developers are not able to start maintenance tasks without first pinpointing the relevant code.

For instance, Bob is a developer who has just taken over a software project and the first task given out by his leader is to fix a bug that has been recently reported. Due to unfamiliarity with the code base, he does not know where and how to begin especially. It is even worse if only few documentations on the system are provided and the help of the predecessors are not available. The only option that is left to Bob seems to be manual searching around the code base, which could be trivial, frustrating and time-consuming even with the help of conventional tools like integrated development environments (IDE). Unfortunately, this scenario is quite common among practitioners in real world. This makes feature location techniques significant to programmers like Bob in the context of developing and maintaining dependable software project.

## Contribution

## 1.3 Objectives

### Primary Objective

### 1.3.2 Secondary Objective

## 1.4 Dissertation Structure

The dissChapter 6 presents the conclusion of what has been done during dissertation and look forward to the future work.

# 2 Background and Related Work

In this chapter we introduce the background and previous work with respect to feature location, covering domain knowledge used in the thesis. It starts with introducing the classes of feature location techniques and a brief introduction to textual method in section 2.1. Section 2.2 gives an overview of definition and workflow of the LDA. In section 2.3 the principle of document vector is introduced. At last, Previous work related to the thesis is presented.

## 2.1 Feature Location Techniques

Feature location is an extensive area involving various research domains and indispensable part of software maintenance (Dit, Revelle, Gethers, & Poshyvanyk, 2013). In general, the goal of a typical feature location task is generating ranked list of program elements at different granularity (e.g., files, methods or classes) given a developer query as input, which could provide sufficient information to developers for tracking location of implementation of functionalities in source code. Different FLTs vary in input requirements, the specific algorithm of locating features and how they present the results. The dimensions of a FLT concsist of types of analysis (dynamic analysis, static analysis, textural analysis, etc.), user input (natural language query, execution scenario, source code artifacts), data sources (compliable element, non-executable element, execution trace, historical information, etc.), output (file/class, method/function and statements), programming language support and evaluation methods (Dit, Revelle, Gethers, & Poshyvanyk, 2013).

One of the mainstream solution in feature location is text-based approach. Using information such as source code or natural language query as input, textual methodologies of feature location analyzes program elements and their properties reflected in the code corpus. The method is based on assumption that the implementation of a feature in a software system may be relevant to the associated requirement. Information Retrieval (IR) techniques are one of the major direction among textual approaches to feature location. Instances of IR include Latent Dirichlet Allocation and other statistical learning methods used to find the associated code of the feature by learning and recognizing source code elements such as method declaration and comments that has high similarities to a query provided by a user. IR techniques are significant in the areas of Software Development, Maintenance and Evolution (Binkley & Lawrie, 2006). No matter what the type of textual analysis used, the quality of feature location is heavily tied to the quality of the source code naming conventions and/or the user-issued query (Dit, Revelle, Gethers, & Poshyvanyk, 2013).

## 2.2 Latent Dirichlet Allocation

LDA (Blei, Ng & Jordan, 2003) is a generative probabilistic model, a typical class of bag-of-words model, as well as an unsupervised learning approach designed for processing discrete data such as text corpora. Motivated by the problem of modelling text corpora, LDA extracts a set of topics associated to the artifacts and then gauges the distribution of finite topics over the corpus. It aims at finding shorter description of the entries of a large collection while holding statistical relation with respect to basic learning tasks (e.g., text classification, similarity and relevance judgements, novelty detection, etc.). This process can be considered as dimensionality reduction on traits with good interpretation for the data in the matter of probabilistic semantics.

The code concept of LDA is that documents can be represented as composition of random latent topics K and each topic could in turn expressed with distribution over words. This is quite easy to understand, because creating articles always begins with setting up a few topics of the article.

The procedure of training of LDA is separated into two phases. The first phase is finding out appropriate K topics and the second phase is determining the distribution of these topics on each document. The algorithm of LDA is interpreted as following.

Assuming there are two matrices A and B. A is a m × k matrix where the m means the number of documents and k represents the distribution of topics on single document. The ith document of A means a topic vector that could represent the word. B is a m × k matrix where k and v representing the number of topics and frequency of words related to the topic respectively. The Bi represents a vector on vocabulary vectors (the total number of words is v) The process of determining matrix A and matrix B is equivalent to generation of a LDA model.

Using a three-level hierarchical Bayesian model, LDA models every single entry of a document collection into a combination of various topics where each document is characterized by a distribution of topics over unstructured document. The Figure 1 shows the three levels of LDA structure in graphical representation. The boxes are plates representing replicates, which are repeated entities. The outer plate represents documents and the inner plate means the repeated choice of words positions in a given document.

The parameters α and β are corpus level parameters, assumed to be sampled once in the process of generating a corpus. The variables θ are document-level variables, sampled once per document. The variables z and w stand for word-level variables and are sampled once for each word in each document. The parameters of LDA that used to generate models during sampling consist of the number of topics, the iterations need for coverage and the hyper parameters α and β. α affects the distribution of topics per document and β has influence on distribution of term per topics. In practical application, appropriate parameters tuning for LDA could lead to precise reflection of underlying topics representing each document and thus improve the learning accuracy. More specifically, LDA assumes the following generative process for each document w in corpus D:

1. Sampling from Dirichlet distribution α and generate Ai  for document i
2. Sampling from Dirichlet distribution β and generate Bk for topic k
3. for the jth word of document i in D,
   1. sampling from multinomial distribution Ai  and generate topics for Di,j
   2. sampling from multinomial distribution Bzi,j and finally the wi,jis generated

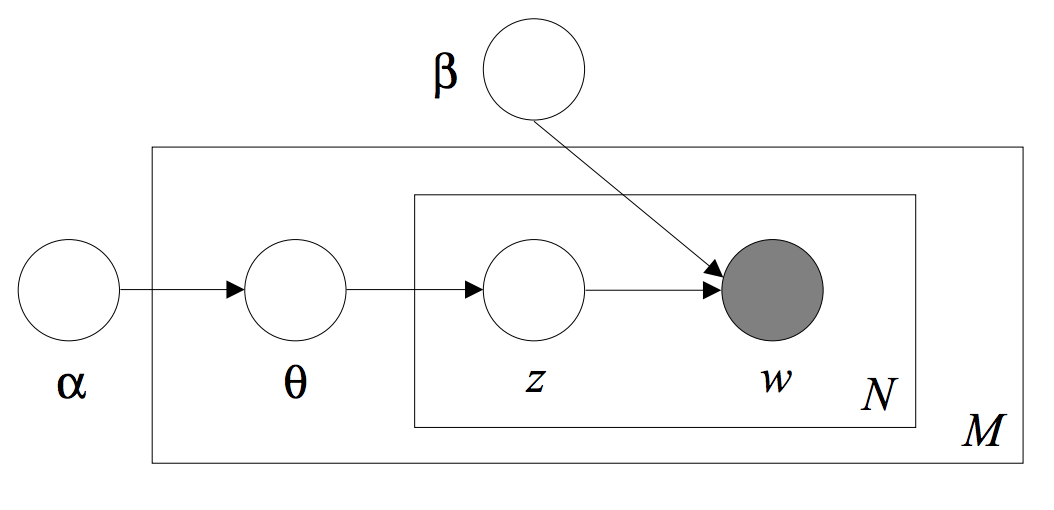


Figure Graphical Model Representation of LDA

Although LDA was originally used for learning natural language text, the assumption that features in programing language seems having similar characteristic as those in natural language makes it possible to apply LDA in software artifacts and could be leveraged to support software maintenance task such as feature location. In the context of using LDA for feature location, the collection of documents used for input could be program elements such as source code classes or methods, bug trace information or any other textual software artifact. The Eclipse extension tool called TopicXP has been successfully developed and published in 2010 (Savage, Dit, Gethers & Poshyvanyk, 2010). The plug-in aims at supporting programmers in daily tasks especially in software maintenance by extracting the unstructured embedded information from source code comments and identifiers using an advanced information retrieval technique, that is Latent Dirichlet Allocation. The topics generated from the trained models are mapped to the source code, and the underlying relationship among the topics is then determined by censoring the static dependencies of the code components. The developers are able to browse through these topics and access the source code related to the topics with a graphical user interface. The results of experiment that Respondents were required to perform functionality location over two software jEdit and muCommander) using Eclipse plus TopicXP or using just the pure Eclipse IDE in the task show it is competitive axillary tool comparing to Eclipse IDE and even more efficient in some cases regarding feature location.

## 2.3 Document Vector

Document Vector or Paragraph Vector (Le & Mikolov, 2014) is class of unsupervised learning algorithm based on deep learning, which enable learning continuous distributed and variable-length sentences or documents. DV takes advantage of not only the context of the document but also the semantics of the document and use them as features in prediction task. The algorithm of DV is inspired by and developed on top of the approach of learning distributed vector representation of words using neural network (Bengio et al, 2010) whose sprit is that the surrounding words in the context are contributed to predict a word in the sentence.

As shown at Figure 2, the fourth word “on” could be predicted via capturing the context the preceding three words “the”, “cat” and “sat”. During training phase, A sliding window that has fixed length is used for sampling. Each single document in corpus is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W. Afterward the document vector and the word vector are concatenated or averaged to infer the next word. The paragraph token could be considered as another word. The paragraph token is like a supplement for the missing information of the current context of the paragraph and could be used as memory of the paragraph topic, that is why the model called Distributed Memory Model of Paragraph Vectors. The paragraph vector (the column of D) is unique among other paragraph vectors (other columns of D) while the word matrix W is shared across documents (word itself has the same meaning even in different paragraphs). Being similar with the process of learning word vectors, both paragraph vectors and word vectors are trained using stochastic gradient descent and backpropagation (Rumelhart, David, Hinton, Geoffrey, Williams & Ronald, 1986). The parameters of the model (a matrix of weights) are updated in every iteration, which makes the model gradually approximate the underlying target function. During prediction phase, the new paragraph that needs to be inferred is vectorised and inputted into the trained model whose output is also a vector representing that document and could be fed into any conventional classifiers such as logistic regression.

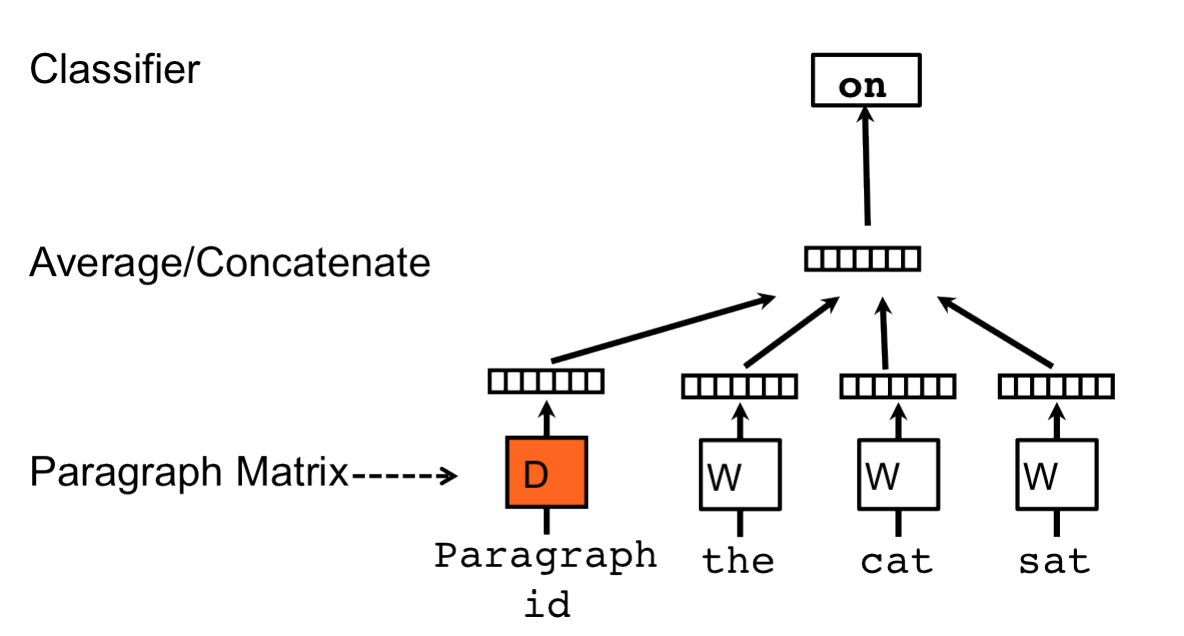


Figure Example of Learning Paragraph Vector

Comparing to other conventional methods like bag of words and n-gram for text learning task (e.g. text classification and sentiment analysis), one of the merits of DV is that it is capable of retaining adequate semantics of the corpus during training. Results on a couple of text classification tasks show that DV is competitive to state-of-art approaches in terms of learning representation for sequential (Le & Mikolov, 2014).

## 2.4 Related Work

Previous researches show that the DV-based feature location techniques have better performance than an analogous FLT based on LDA in terms of feature location at the granularity of method (Corley, Damevski & Kraft, 2015). The experimental results reveal that DV model has lower computational cost regarding training while maintaining accuracy on par with LDA, which makes it a potential solution of implementing intelligent programmer search tool in IDE.

The paper of Corley et al. introduces a series of methodology for corpora preprocessing, model comparison and evaluation with respect to method level feature location based on a dataset extracted from six public open source software system (Dit, Poshyvanyk & Kagdi, 2013). As shown in Table 1, the dataset includes 633 queries for method level goldsets and was automatically collected from changsets that relate to quires on issue tracking system. These open-source softwares are ArgoUML, jEdit, JabRef and muCommander.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data | Description | Release Interval | Files | Methods |
| ArgoUML 0.22 | UML diagram tool | 0.20-0.22 | 1439 | 11000 |
| ArgoUML 0.24 | UML diagram tool | 0.22-0.24 | 1480 | 11464 |
| ArgoUML 0.26.2 | UML diagram tool | 0.24-0.26.2 | 1752 | 14597 |
| JabRef 2.6 | BibTeX bibliography management tool3 | 2.0-2.6 | 579 | 4607 |
| JEdit 4.3 | text editor | 4.2-4.3 | 503 | 6413 |
| muCommander 0.8.5 | cross-platform file manager | 0.8.0-0.8.5 | 1069 | 8187 |

Table Dataset from Change History

Their experiment workflow starts with preprocessing. Before training, series of preprocessing steps are employed on the corpora and are in following order:

* Splitting: split items of corpora into tokens according to the conventional coding style rules (e.g., the use of camel case or underscores) and on the presence of non-letters (e.g., punctuations or digits)
* Normalizing: replace any upper case letters with corresponding lower case letters.
* Filtering: remove stop words, programming language keywords, standard libraries entity names or short words

After extracting documents, tokenizing and preprocessing, the performance of LDA and DV on training time and accuracy respectively in terms of method level feature location are evaluated by applying the learning methods on subject systems then measuring the outcomes with the goldsets. In training phase, the approach is quite straight forward for both DV and LDA, that is training a model using all source codes as input data. As for prediction, the vector related to the query is firstly obtained from the generated model and a rank sorted by similarity to the query is calculated and inferred using that output. At last, the rank can be evaluated with the metric. In addition, they found out a method based on word vector are efficient for locating feature through taking query’s summed vectors and perform pair-wise similarity to them. The impact of one of the model parameters, that is number of topics (the analogous parameter is vector size in DV), was also investigated in their research. It turns out that a few trends could be observed from the results. The DV inference has the worst performance on all versions of ArgoUML, JabRef and jEdit except for muCommander where it beats the other two methods while LDA’s performance is in general a bit better comparing to that of DV inference. DV word vector summation method performs the best on most projects. The DV inference seems much faster than LDA in respect with training time.

# Methodology

## Research Questions

### 3.1.1 Github versus SVN

The dataset is derived from only 6 software projects and three of them are just different release versions of the exact same project (ArgoUML) besides all of them are written in Java.

Despite the fact that the work of Corley et al. seems to have proved that the FLT based on deep learning outperforms the FLT based on Latent Dirichlet Allocation with regard to method level feature location, the conclusions are less persuasive and limited due to the lack of statistical significance on the number of test subject systems

The dataset they used involves a few manual processes, which makes it difficult to extend and reproduce (Dit, Poshyvanyk & Kagdi, 2013). Therefore, we address a methodology consist of automated goldset generation

# Reference

Dit, B. , Revelle, M. , Gethers, M. and Poshyvanyk, D. (2013), Feature location in source code: a taxonomy and survey. J. Softw. Evol. and Proc., 25: 53-95.

V. Rajlich and N. Wilde, "The role of concepts in program comprehension," *Proceedings 10th International Workshop on Program Comprehension*, 2002, pp. 271-278.

Blei, D. M., Ng, A. Y., and Jordan, M. I., (2003), "Latent Dirichlet Allocation", Journal of Machine Learning Research, vol. 3, pp. 993-1022.

Roweis, S. T.; Saul, L. K. (2000). "Nonlinear Dimensionality Reduction by Locally Linear Embedding". Science. 290 (5500): 2323–2326.

Q. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” in Proceedings of the 31st International Conference on Machine Learning, T. Jebara and E. P. Xing, Eds., 2014, pp. 1188–1196.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. Nature, 323(6088):533–536, 1986.

Bengio, Yoshua, Schwenk, Holger, Senecal, Jean-Sebastien, Morin, Frederic, and Gauvain, Jean-Luc. ´ Neural probabilistic language models. In Innovations in Machine Learning, pp. 137–186. Springer, 2006.

Binkley, D. and Lawrie, D., (2010a), "Information Retrieval Applications in Software Development", in Encyclopedia of Software Engineering, P. Laplante, Ed.: Taylor & Francis LLC.

Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T., (1987), "The Vocabulary Problem in Human-System Communication", Communications of the ACM, vol. 30, no. 11, pp. 964-971.

C. S. Corley, K. Damevski, and N. A. Kraft, “Exploring the use of deep learning for feature location,” in Int’l Conf. Software Maintenance and Evolution, Sep. 2015, pp. 556–560.

Savage, T., Dit, B., Gethers, M., and Poshyvanyk, D., (2010a), "TopicXP: Exploring Topics in Source Code using Latent Dirichlet Allocation", in Proceedings of 26th IEEE International Conference on Software Maintenance (ICSM'10), Timişoara, Romania, September 12-18, pp. 1-6.

B. Dit, A. Holtzhauer, D. Poshyvanyk, and H. Kagdi, “A dataset from change history to support evaluation of software maintenance tasks,” in Proceedings of the 10th IEEE Working Conference on Mining Software Repositories, 2013, pp. 131–134.