Soft Computing (2019) 23:13433–13448
https://doi.org/10.1007/s00500-019-03882-y

METHODOLOGIES AND APPLICATION

A methodology to rank the design patterns on the base of text relevancy

Shahid Hussain[1] · Jacky Keung[2] · Mohammad Khalid Sohail[1] · Arif Ali Khan[3] · Manzoor Ilahi[1] · Ghufran Ahmad[1] · Muhammad Rafiq Mufti[4] · Muhammad Asim Noor[1]

Published online: 18 March 2019

Abstract
Several software design patterns have cataloged either with canonical or as variants to solve a recurring design problem. However, novice designers mostly adopt patterns without considering their ground reality and relevance to design problems, which causes to increase the development and maintenance efforts. The existing automated systems to select the design patterns need either high computing effort and time for the formal specification or precise learning through the training of several classifiers with large sample size to select the right design patterns realized on the base of their ground reality. In order to discuss this issue, we propose a method on the base of a supervised learning technique named 'Learning to Rank', to rank the design patterns via the text relevancy with the description of the given design problems. Subsequently, we also propose an evaluation model to assess the effectiveness of the proposed method. We evaluate the efficacy of the proposed method in the context of several design pattern collections and relevant design problems. The promising experimental results indicate the applicability of the proposed method as a recommendation system to select the right design pattern(s).

Keywords Software design patterns · Text mining · Learning to rank · Performance · Classification

Communicated by V. Loia.

✉ Shahid Hussain
shussain@comsats.edu.pk

✉ Arif Ali Khan
aakhan@comsats.edu.pk

Jacky Keung
jacky.keung2-c@my.cityu.edu.hk

Mohammad Khalid Sohail
Khaled_sohail7@comsats.edu.pk

Manzoor Ilahi
mitamimy@comsats.edu.pk

Ghufran Ahmad
ghufranahmad23@comsats.edu.pk

Muhammad Rafiq Mufti
Rafiq_muftee16@ciitvehari.edu.pk

Muhammad Asim Noor
asimnoor11@comsats.edu.pk

[1] CU, Islamabad, Pakistan

[2] City University of Hong Kong, Hong Kong, China

[3] Nanjing University, Nanjing, China

[4] CU, Vehari, Pakistan

1 Introduction

In software development life cycle, software design is considered as a most challenging task as compared to other activities. An architecture-based design method is one of the main groups which can be realized during the evolution process. In this category, different software architectural styles are applied and can be considered as coordination tools among the activities of the software development life cycle. An architectural style describes the components and their relationship and provides a bridge between satisfactions of system's critical requirements to implementation (Bass et al. 2012). The most common architecture-based design method is Attribute Driven Design (ADD) which incorporates the different architectural strategies and patterns to fulfill the quality attributes. In the working procedure of ADD, a software developer selects an architectural style and chooses the desired quality attributes included in the software system. Subsequently, software developers use the description of design requirements related to selected architectural component and employed software design patterns (Wood 2007). Over many years, based on their experiences, software developers have encapsulated and suggested the

proven solutions to satisfy the recurring design problems. The experience-based solutions are organized and realized as a standardized form for design patterns. For example, the canonical solution of Composite GoF (Gang-of-Four) design pattern is the result of formulating the recurring structural and compound design problems related to the composition of objects from its nesting and building tree structures (Gamma et al. 1995).

The employment of design patterns in software development is considered as an alternative approach to software refactoring to address the declined quality of a software system. Besides, other advantages to employing design patterns are: (1) to maintain the consistency between design and implementation, (2) to enable the relationship between developers, (3) to increase the reusability and (4) software modularization (Gamma et al. 1995). However, due to the existence of an alternative solution of spoiled patterns (Bouhours et al. 2015), challengeable design patterns (Booch 2006), and interdisciplinary design patterns (Baraki 2015), it is so difficult for a novice designer to find the right design pattern(s) and determine its applicability in order to solve a design problem. Since design patterns are formulated and cataloged on the base of software developer's experience, consequently, a designer with no or little experience with design patterns have concerns how to find the best one. In order to address this issue, the current efforts have been made to automate the design pattern selection process, which can be divided into three approaches named UML-based (Kim and Khawand 2007; Hsueh et al. 2007; Kim and Shen 2008), ontology-based (Hasso and Carlson 2005; Hotho et al. 2005; Khoury et al. 2008), and text categorization-based approach (Hasheminejad and Jalili 2012; Hussain 2016; Hussain et al. 2017). The effectiveness of existing text categorization-based automated systems for design pattern selection is limited to the need of an ample training sample size and formulating of most representative features set to overcome multiclass problem (Uysal 2016) for precise learning. However, it is not easy, especially with the existence of complex and interdisciplinary design pattern collection, where numerous classifiers will be required to make effective learning for each pattern class (Booch 2006; Kim and Khawand 2007).

In order to address this issue, we propose a method to rank the design patterns on the base of their text relevancy with given design problem(s). The proposed method is exploited through a supervised learning technique named 'Learning to Rank' which can help to rank the design patterns on their ground truth. Like other studies (Hasheminejad and Jalili 2012; Hussain 2016; Hussain et al. 2017), in this study, we also consider the design pattern(s) selection as Information Retrieval (IR) problem and proposed a methodology to rank the design pattern(s) for the given design problem(s) using a supervised learning technique named 'Learning to Rank'. There are several ranking algorithms which have been applied in order to solve information retrieval problems in the different domain such as in web search. These algorithms are grouped into point-wise, pair-wise, and list-wise categories. The point-wise and pair-wise ranking algorithms cannot model the ranking problem in a straightforward fashion such as RankNet, MART, and RankBoost, while list-wise ranking algorithms such as Coordinate Ascent, AdaRank and LambdaMART aid to rank the list of objects as instances and train a ranking function by minimizing the list-wise loss function applied on the ground truth and predicted list. The Lemur project is the best source for the implementation of several ranking functions.

However, in this paper, we leverage the three widely used list-wise ranking algorithms Coordinate Ascent, AdaRank, and LambdaMART in order to evaluate the effectiveness of the proposed methodology. The ranking algorithms used in the proposed study are described in Sect. 3. The framework of the proposed methodology is shown in Fig. 1. Though we have presented a short version of the proposed method (Hussain et al. 2017); however, in this paper, our main contributions are.

1. We propose a prototype of an automated recommendation system for the classification and selection of design patterns based on the text categorization approach and exploit through supervised learning technique 'Learning to Rank'.
2. We propose an evaluation model to investigate the effectiveness of the proposed method.
3. We leveraged and customized the three well-known ranking algorithms and assess the effectiveness of the proposed method.
4. We evaluated the effectiveness of the proposed method with four widely used design pattern collections and set of real design problems.

The rest of the paper is organized into eight sections. In Sect. 2, we summarized the related work regarding the automated systems used in the domain of classification and selection of design pattern. In Sect. 3, we present the brief introduction of design patterns and its components. In Sect. 4, we describe the framework of the proposed method. In Sect. 5, we describe the ranking algorithms used in the implementation of the proposed method. In Sect. 6, we proposed an evaluation model to investigate the effectiveness of the proposed method. In Sect. 7, we discussed the experimental results of case studies which are performed in this paper. In Sect. 8, we discuss the consequences retrieved from the experimental results. Finally, in Sects. 9 and 10, we describe threats to the validity and the conclusion of our work.

2 Related work

In 1977, Alexander, the father of patterns describe a roadmap to capture the design knowledge and present a collection of patterns to help the architects and engineers.

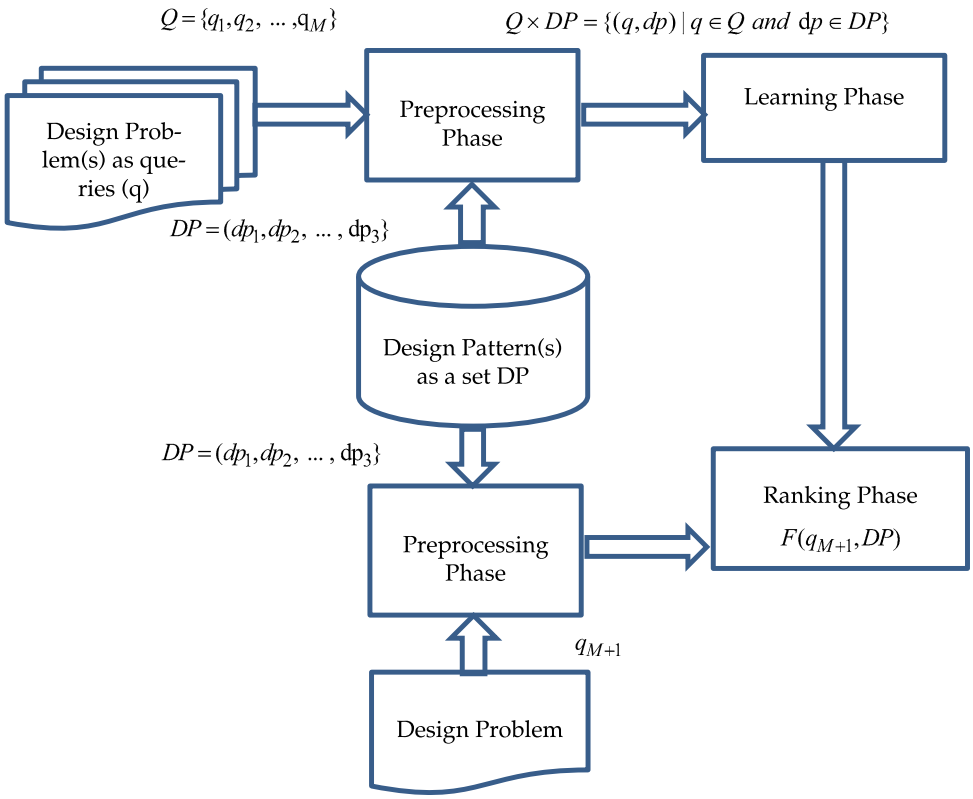


Fig. 1 Overview of the proposed methodology

According to Alexander, a pattern can be described in three sections that are context, a problem description, and a corresponding solution. However, the building components of each section varied depending on the developer's experience and knowledge in the response of handling design problems. For example, in the domain of object-oriented development, Gamma et al. (1995) present 23 design patterns and classified them into three groups named Creational, Structural, and Behavioral. Subsequently, in the domain of real-time relevant application, Douglass (2002) present 34 patterns which are divided into five categories named Concurrency, Resource, and Reliability, Memory, Distribution, and Safety. Subsequently, the Duyne's HCI pattern (Duyne et al. 2003), volumes of Pattern-Oriented Software Architecture (POSA) series (Schmidt et al. 2000; Kircher and Jain 2004; Buschmann et al. 2007; Buschmann et al. 2007), security patterns (Schumacher et al. 2006), and van Welie usability collection (van Welie 2006) are the well-known design pattern collection. In studies by Birukou (2010), Henninger and Correa (2007), the authors have conducted a detailed survey on the design pattern collection, pattern types, and the issues to discover and select the more appropriate patterns. In this paper, we summarized the researcher's efforts made to automate the selection of design patterns using different approaches such as UML-Based, ontology-based and text categorization approach.

Kim and Khawand depict an approach to select the design pattern by specifying its problem domain. In their study, they reported that the problem domain of design patterns can help to describe the known problems in the context. The authors used role-based modeling language (RBML) as UML-based pattern specification language to describe the meta-model to formalize the design patterns. The structure of a meta-model is formulated through a class and collaboration diagrams to describe the structural and behavioral aspects of software and detect a design pattern. The structural similarity and scalability in terms of a number of design patterns and its variation are the main constraints in implementation of UML-based approaches. Due to the structural similarity between design patterns, a meta-model cannot specify the problem domain of design patterns. For example, in the case of Gamma et al. design pattern catalog, the meta-model for Strategy pattern looks closer to meta-model of State design pattern; however, their goals are different. Subsequently, there are some design patterns such as Prototype, whose specification through meta-model cannot be presented (Kim and Khawand 2007). In another study, Hsueh et al. (2007) also follow the UML-based approach, depict a case study with a limited number of design patterns, and consider the four perspectives such as activity-view, requirement-view, problem-view, and tradeoff-view to find a systematic way to select a design pattern. They used UML notations to

describe the design problems by incorporating the non-functional requirements besides functional requirements.

In their study, Khoury et al. (2008) introduce an ontological interface depicted in Ontology Web Language (OWL) to aid developers to discover an eligible set of security patterns. The proposed ontological interface implements a mapping between security requirements on one side and security bugs, security errors and threat model on the other side of the context. Subsequently, Blomqvist (2008) proposed a semi-automatic ontology construction approach named OntoCase, rely on the use of ontology pattern to rank and select the right design pattern. In this work, the author defines criteria which include, relation matching, class matching, density, centrality and semantic similarity to rank the design patterns. The lack of existence of formal description in all software design domain and lack of evaluation reports is the common constraint to fulfill the aims of these studies. In a recent study, Velasco-Elizondo et al. (2016) proposed an approach which based on knowledge representation and information extraction to analyze the architectural pattern description with respect to the specific quality attributes. In order to facilitate the inexperienced architects, the proposed approach is automated using a computable model which can be referred to a prototype tool. However, the proposed approach claims on the quality attribute-oriented pattern selection and lack of limitation on the use of a predefined and closed pattern repository. Besides UML and ontology approach-based automated systems, researchers have made their efforts to make the recommendation systems based on the context and semantics of program entities such class and method invocations (Issaoui et al. 2015; Palma et al. 2012; Medhat et al. 2014).

In three recent studies, Hasheminejad and Jalili (2012) and Hussain et al. (2016, 2017) follow the text categorization approach and present the automated method to select the design pattern with respect to the degree of similarity between the description of design problems and problem definitions of the retrieved design patterns. In both studies, the authors consider the classification and selection of design patterns with respect to a given design problem as an information retrieval problem and used the text categorization approach. Subsequently, the common aim of both studies is to obtain the promising results with respect to the ground truth. However, for the precise learning, either there is a need of large sample size or several classifiers to overcome the multiclass problem (Uysal 2016). For the last decade, a new machine learning technique 'Learning to Rank' has been applied in different information retrieval related applications such as web search. The 'Learning to Rank' is a supervised learning technique which is used for the creation of a ranking model $f(q,d)$ to retrieve a document d with respect to the query q. In our study, we consider the design pattern selection as information retrieval problem and proposed a method based on 'Learning to Rank' to select the right design pattern(s) for the given design problem(s).

3 Illustration of design patterns

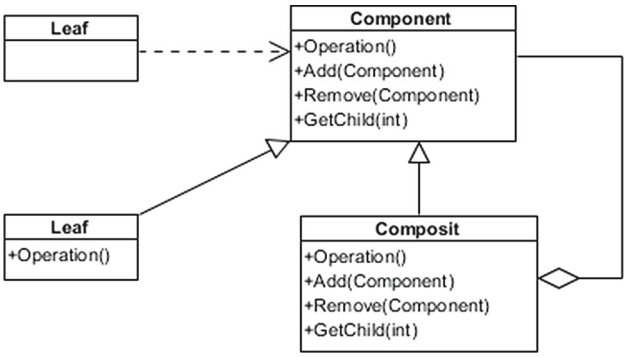
Usually, a design pattern template is divided into two sections named Problem Domain and Solution Domain shown in Table 1. The former section defined the context of a problem, while the latter section defined the structure applied to a problem and collaboration with other patterns. The sub-sections of Problem Domain describe the description, scenario illustration of a problem, and a condition required to apply a design pattern. Subsequently, the sub-sections of Solution Domain describe the structure of the participants, its participating components (classes and objects), a collaboration among components and consequences of the applied pattern (Gamma et al. 1995). The illustration of a GoF design pattern named Composite is shown in Table 1.

4 Proposed methodology

The rapid development of software design patterns leads to an increase in the amount of its electronic documentation worldwide. Therefore, there is a need to organize these patterns to reduce the time and efforts required to a novice designer to select more appropriate design patterns. This situation motivates us to consider the issue of design pattern organization and selection problem as an information retrieval problem, and use the text categorization approach to rank the design patterns with respect to text relevant to the given design problem. Though this issue is addressed by Hasheminejad and Jalili (2012) and Hussain et al. (2016, 2017) in their study, however, in the case of complex and interdisciplinary design pattern collections (Booch 2006; Baraki 2015), several classifiers are required to make supervised learning of each pattern class or need of a global filter-based feature method to construct most representative features set to overcome the multiclass problem (Uysal 2016).

We look out the capacity of automated text categorization in different domains (Hasheminejad and Jalili 2012; Medhat et al. 2014; Zhang et al. 2014; Hussain et al. 2016, 2017) and use in the proposed method for two purposes: (1) to rank the design patterns with respect to the text relevancy with design problem(s) description and (2) automatically retrieve a more appropriate design pattern(s) for a given real design problem. Usually, the ranking tasks are performed by using local $f(q,d)$ and global $F(q,D)$

Table 1 Illustration of the GoF composite design pattern

Design pattern name	Composite
Problem domain	
Intent	Compose objects into tree structures to represent part-whole hierarchies. Composite lets client treat individual objects and composition of objects uniformly
Motivation	Graphics applications like drawing editors and schematic capture system let users build complex diagrams out of simple components. The user can group components to form larger components, which in turn can be grouped to form still larger components. A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives. But there's a problem with this approach: Code that uses these classes must treat primitives and container objects differently, even if most of the time they use treats them identically. Having to distinguish these objects makes the application more complex. The Composite pattern describes how to use recursive composition so that clients don't have to make this distinction". The rest of motivation section is available in Gamma et al. (1995)
Applicability	The composite pattern applies when there is a part-whole hierarchy of objects and a client needs to deal with objects uniformly regardless of the fact that an object might be a leaf or a branch
Solution domain	
Structure	
Participant Collaborations	Component, leaf, composite and client
Consequences	The composite pattern Defines class hierarchies consisting of primitive objects and composite objects. Primitive objects can be composed and so on recursively. Whatever client code expects a primitive object, it can also take a composite object Makes the client simple. The client can treat composite structure and individual object uniformly. The client normally doesn't know (and should not care) whether they are dealing with a leaf or composite component. This simplifies client code, because it avoids having to write tag-and-case-statement-style functions over the class that defines the composition Makes it easier to pass new kind of components. Newly defined Composite or leaf subclasses work automatically with the existing structure and client code. Clients don't have to be changed for new Component classes Can make your design overall general. The disadvantage of making it easy to add new components is that makes it harder to restrict the components of a composite. Sometimes you want a composite to have only certain components. With Composite, you can't rely on the type system to enforce those constraints for you. You will have to use run-time checks instead
Implementation	The implementation of Composite Design Pattern is available in Gamma et al. (1995) using C ++ constructs
Related Patterns	Decorator pattern: When decorators and composites are used together, they will usually have a common parent class

ranking models. These ranking functions are created to rank the set of documents D with respect to a user's query q. The existing approaches BM25 model and LMIR (Language Model for IR) are implemented to rank the documents by creating $f(q, d)$ without training, however, to achieve our objective, we use a machine learning technique named 'Learning to Rank' to construct a ranking model to select more appropriate design pattern(s) for a given design problem. Like other supervised learning techniques, 'Learning to Rank' also needs the training and testing data. In order to evaluate the effectiveness of proposed method, we leverage and incorporate three widely used list-wise ranking algorithms named LambdaMART (Wu et al. 2007), Coordinate Ascent (Metzler and Croft 2007), and

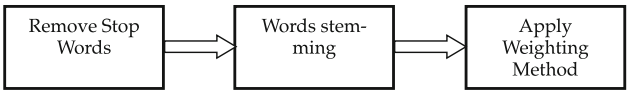


Fig. 2 Preprocessing activities

AdaRank (Xu and Li 2007). We formulate the proposed method in three phases Preprocessing, Learning (on training data) and Ranking (on testing data).

4.1 Preprocessing

The first phase of proposed method is preprocessing, which applied to problem description of all design patterns of a collection and description of given design problem(s). The sequence of preprocessing activities is depicted in Fig. 2.

The first two activities are performed to remove the stop words and words stemming consecutively. There are several text mining tools like JPreText (A simple Text Preprocessing Tool)[1] and their APIs[2] (Application Programming Interfaces) are available to perform these tasks. The problem description part of each design pattern and design problem description is presented in the form of a feature vector. Subsequently, a Vector Space Model (VSM) includes non-repeated terms of all documents are created. The structure of Vector Space Model (VSM) is described through N design patterns and design problem(s) with M non-repeated terms. Finally, in the last activity, one of weighting schemes (Sect. 3) is applied to the vector space model.

4.2 Learning phase

In the domain of information retrieval applications, three approaches to 'Learning to Rank' are followed in order to overcome the corresponding problem, such as point-wise, pair-wise, and list-wise related ranking algorithms. In the learning phase of proposed methodology, we leverage and analyzed the list-wise ranking algorithms named LambdaMART (Wu et al. 2007), Coordinate Ascent (Metzler and Croft 2007), and AdaRank (Xu and Li 2007) due to the existence of design patterns either as variant or a spoiled form (Bouhours et al. 2015; Hasheminejad and Jalili 2012; Hussain et al. 2016, 2017) of canonical solution and ranking them with respect to text relevancy with given design problem(s). Subsequently, leveraged list- wise ranking algorithms satisfied the properties (a) consistency, (b) soundness, (c) convexity and computational efficiency with respect to minimization of their loss function. We performed two tasks in the learning phase. The first task is performed to conduct learning on the pair-wise (Problem-Pattern) preference and construct a training set which includes design patterns, design problems, and their relevance levels. There are several ways to describe the relevance levels between design patterns and design problem, such as the use of similarity measures (Huang 2008) or use of labels to grade the patterns associated with the design problem (Li 2011). In this study, we assumed the latter approach in order to evaluate the pattern relevance with respect to a design problem. The formulization of the training set for learning is shown as follows.

We assume that Q is design problem set (that is $\{q_1, q_2, \ldots, q_M\}$, where q referred as a design problem),DP is the pattern set (that is $\{dp_1, dp_2, \ldots, dp_N\}$, where dp referred as a design pattern), and L is the label set (that is $\{1, 2, \ldots, l\}$) denotes different grades. The revised $L = \{l_{1,1}, l_{1,2}, \ldots l_{1,N}\}$ label set and $DP = \{dp_{1,1}, dp_{1,2}, \ldots, dp_{1,N}\}$ pattern set are created with respect to the association with a given design problem q_1, and are ordered $l \succ l-1 \succ l-2 \succ \cdots \succ 1$ accordingly. Where $\succ$ depicts the order relation. Subsequently, a feature vector $v_{i,j} = \phi(q_i, dp_{i,j})$, $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$ is created for each problem-pattern pair $(q_i, dp_{i,j})$, where ϕ referred as feature function for a problem-pattern pair. Finally, the training dataset is created according to Eq. 1.

$$\text{Training Set} = \{(q_i, \text{DP}_i), L_i\}_{i=1}^{M} \tag{1}$$

The second task is performed to construct a local ranking model $f(q, dp)$ to assign a score to a problem-pattern pair (q, dp) and global ranking model $F(q, DP)$ to assign the list of scores to problem-patterns pairs (q, DP). We assume that $F(.)$ is mapping function which maps a list of feature vectors $V = \{v_1, v_2, \ldots v_3\}$ to list of grades. The design problem(s) and its associated documents form a group which is represented by i.i.d. data, while members of each group have no i.i.d (Li 2011).

4.3 Ranking phase

In the ranking phase of proposed methodology, we formulate the ranking list that is permutation $\pi_i \in \Pi_i$ for a given design problem q and relevant patterns DP_i. Where Π_i referred to the list of permutations π_i on DP_i to present the position (or rank) of a pattern. For example, the position of the ^{j}th pattern can be represented as $\pi_i(j)$. The testing dataset consist of a new design problem q_{M+1} and the associated patterns DP_{M+1} shown in Eq. 2.

$$\text{Testing Dataset} = \{(q_{M+1,}\text{DP}_{M+1})\} \tag{2}$$

Subsequently, we create the feature vector v_{M+1} and used the ranking model (Sect. 4.2) to assign the grades to

[1] http://sites.labic.icmc.usp.br/torch/msd2011/jpretext/.

[2] http://www.opencalais.com/opencalais-demo/.

the patterns DP_{M+1} and output the ranking list π_{M+1} in the sorted form.

5 Ranking algorithms used in the proposed method

The ranking algorithms are categorized into two basic groups: (1) point-wise and pair-wise, and (2) list-wise ranking algorithms. The ranking algorithms related to the former group use neural nets to learn the pair-wise preference function, while the latter group algorithms use the relevance regression or classification. Subsequently, the ranking algorithms, which based on pair-wise approach, transform the ranking into the classification on object pairs. The list-wise-based ranking algorithms take a rank list of objects as instances and trains a ranking function through the minimization of a list-wise loss function described in the predicted and ground truth lists. In this study, we leverage the three well-known list-wise ranking algorithms Coordinate Ascent, AdaRank, and LambdaMART to exploit the proposed method, which has been implemented[3] to perform certain ranking tasks.

5.1 LambdaMART

The LambdaMART ranking algorithm is built for boosting tree algorithm name MART (Multiple Additive Regression Trees) that perform gradient descent $\delta F \propto -\partial C/\partial F$ in the function space. Where the cost C can be considered as a functional value F, and the functional gradient $\partial C/\partial F$ is evaluated at the training points. The boosted tree aid to estimate the smooth regression to the gradients and tree can be viewed as a small step in the function speech. Subsequently, the LambdaRank used the approximation for the gradient of cost called $\lambda-$ gradients $\equiv S_{ij}\left|\Delta \mathrm{NDCG} * \partial C_{ij}/\partial o_{ij}\right|$ for any given document (i.e., Pattern in our study), where $o_{ij} = s_i - s_j$ referred as difference in the ranking score for a pair of documents in a query,s_i is represented as $F(x_i)$, and the cross-entropy cost (C_{ij}) is represented as $C_{ij} \equiv C(o_{ij}) = s_j - s_i + \log(1 + e^{s_i - s_j})$. The authors combine LambdaRank and MART to form a new algorithm named LambdaMART and score of each document for a query is represented as $s_i = (1-\alpha)s_i^R + \alpha s_i^{R'}$ where s_i^R and $s_i^{R'}$ present the scores for rankers R and R', respectively, and α curves from 0 to 1 that is $\alpha \in [0,1]$. The objective of LambdaRank function gradient is to optimize the NDCG. In our proposed method, we leveraged the LambdaMART (Wu et al. 2007) and customized according to our design pattern collections and given design problems. We will refer Wu et al. (2007) to the readers for the detail understanding of MART and LambdaMART. The customize LambdaMART algorithm is shown as follows.

LambdaSMART algorithm		

Step-1	for i= 1 to N do
	$F_0(x_i) = BaseModel(x_i)$
	End for
		\\ Base Model may be empty or set to a sub-model.

Step-2	For m=1 to M do	\\ Repeat Step-3 to Step-6 for M round boosting. After each iteration a regression tree is constructed on all design patterns and for all design problems.

Step-3	for i= 1 to N do
	$y_i = \lambda_i$
	$w_i = \partial y_i / \partial F(x_i)$
	End for
		\\ Step-3 is performed to calculate the λ - gradient and derivative of λ - gradient for each design pattern.

Step-4	$\{R_{tm}\}_{l=1}^{L}$	\\ Step-4 is used to create a regression tree with L-terminal nodes on $\{y_i, x_i\}_{i=1}^{N}$, where L is the number of leaf nodes of each constructed tree.

Step-5	For l= 0 to L do
	$\gamma_{lm} = \sum_{x_i \in R_{lm}} y_i / \sum_{x_i \in R_{lm}} w_i$
	End for
		\\ Step-5 is used to find the leaf values based on approximate newton steps.

Step-6	For i=0 to N do
	$F_m(x_i) = F_{m-1}(x_i) + v\sum_t \gamma lm\, I(x_i \in R_{lm})$
	End for
	End for
		\\ Step-6 is used to update model based on the approximate newton step and shrinking size v.

[3] https://www.lemurproject.org/.

5.2 Coordinate ascent

This ranking algorithm based on an optimization technique named Coordinate Ascent, which is applied to unconstrained optimization problems. The aim of Coordinate Ascent is to converge the objective functions gradually with long ridges. The Coordinate Ascent optimizes the problem in Euclidean parameter space $\mathbb{R}^d$ and update the λ_i parameter is updated through equation $\lambda_i^{'} = \arg\max_{\lambda_i} E(R_{\wedge}; T)$ gradually with training data T (Metzler and Croft 2007).

5.3 AdaRank

The AdaRank ranking algorithm aid to improve and optimize the performance measures used in the document retrieval. The learning process of AdaRank algorithm is based on the linear combination of 'weak rankers', which is repeated the re-weight of training sample, creating the weak ranker and then compute the weight for the ranker. The formulation of AdaRank is as follows. Suppose, there are sets of m queries $Q = \{q_1, q_2, \ldots, q_m\}$ and l label ranks $Y = \{r_1, r_2, \ldots, r_l\}$. Each query q_i is associated with a list of documents $d_i = \{d_{i1}, d_{i2}, \ldots, d_{i,n(q_i)}\}$, where $n(q_i)$ present the sizes of the list d_i and $y_{ij} \in Y$. A feature vector $x_{ij} = \psi(q_i, d_{ij}) \in X$ is created for each query-document pair (q_i, d_{ij}), $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n(q_i)$ and $S = \{(q_i, d_i, y_i)\}_{i=1}^{m}$ is created for ranking model $f(x_{ij}) = R$. The AdaRank is the combination of T weak learners that is $f(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$, where $h_t(\vec{x_{ij}}) \in R$ referred as a tth weak ranker (Xu and Li 2007).

6 Evaluation model for the proposed methodology

The evaluation model for the proposed method is performed in two steps. The first step formulates the evaluation of ranking algorithms Coordinate Ascent, AdaRank, and LambdaMART to determine the appropriate pattern and a candidate pattern list for a particular design problem. Subsequently, the second step formulates the use of resources include four design pattern collections and 47 real design problems to evaluate the proposed method.

6.1 Evaluation of ranking algorithms

In the text categorization approach, Precision and Recall for the learners can be estimated using micro-averaging or macro-averaging equations; however, the precision of micro-averaging (used in this study) is better than macro-averaging. In order to select the best weighting method for ranking algorithms used in the proposed method, the effect of each method such as Binary, TFIDF, TFC, LTC and Entropy on the ranking algorithm (relevancy between ranked list and ground reality patterns categories) is computed in terms of Precision, Recall and F-value (Hotho et al. 2005; Hasheminejad and Jalili 2012).

$$P = \sum_{c=1}^{N} \mathrm{TP}_c / \sum_{c=1}^{N} (\mathrm{TP}_c + \mathrm{FP}_c), \quad \text{Micro - Averaging} \tag{3}$$

$$R = \sum_{c=1}^{N} \mathrm{TP}_c / \sum_{c=1}^{N} (\mathrm{TP}_c + \mathrm{FN}_c), \quad \text{Micro - Averaging} \tag{4}$$

$$F = (2 \times P \times R)/(P + R) \tag{5}$$

In Eqs. (3–5), N is the number of design pattern classes decided to evaluate the performance of learning techniques used in the proposed method. For example, in the case of GoF design pattern collection, N is 3 (Sect. 4.3). The term TP is the number of design problems which are correctly identified for each design pattern class, FP is the number of design problems which are incorrectly identified for each design pattern class, and FN is the number of design problems which are missed from each corresponding design pattern class. The values of P, R, and F are computed using Eq. 3–5 respectively. Subsequently, weighting method with highest F-value values is chosen for the ranking algorithm.

The performance of ranking models is evaluated through a comparison between their computed ranking lists (computed permutations with their best weighting methods) and the ranking list given on the ground truth. There are several widely used performance measures in the domain of Information Retrieval (IR) to measure the effectiveness of ranking functions. For example, Mean Average Precision (MAP), Kendall's Tau, Discounted Cumulative Gain (DCG), and Normalized Discounted Cumulative Gain (NDCG) (Wu et al. 2007; Metzler and Croft 2007; Xu and Li 2007). We evaluate the performance of ranking algorithms to select the right design pattern or the list of candidate design patterns for a given design problem, and we used Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG) measures, respectively. The Mean Average Precision (MAP) performance measure is used when we assumed two levels of grades of relevance between Problem and Patterns. For example, the Average Precision (AP) for a given design problem q_i, with its associated patterns DP_i, labels L_i, and the permutation on DP_i is defined in Eq. 6.

$$\mathrm{AP} = \sum_{j=1}^{N} P(j).L_{i,j} / \sum_{j=1}^{N} L_{i,j} \tag{6}$$

From Eq. 6, $L_{i,j}$ referred as pattern label of $DP_{i,j}$ and yield either 0 (for irrelevancy) or 1 (for relevancy) with a design problem q_i. The precision $P(j)$ is computed through Eq. 7 with position $\pi_i(j)$ of design pattern $DP_{i,j}$ for design problem q_i.

$$P(j) = \sum_{k:\pi_i(k) \le \pi_i(j)} L_{i,k}/\pi_i(j) \tag{7}$$

Subsequently, Mean Average Precision (MAP) is computed with respect to the number of design patterns and given design problem.

The Discounted Cumulative Gain (DCG) performance measure is used whenever we assumed more than two levels of grades of relevance between Problem and Patterns. For example, the ranking list π_i for a give design problem q_i with its associated patterns DP_i, and labels L_i is defined in Eq. 8.

$$\mathrm{DCG}(k) = \sum_{j:\pi_i(j) \le k} G(j)D(\pi_i(j)) \tag{8}$$

where $G(.)$ and $D(.)$ referred as gain function and position discount function of pattern $DP_{i,j}$ in π_i. The summation is computed over the top k positions in the corresponding ranking list. The Normalized Discounted Cumulative Gain (NDCG) is an improved form of DCG, which can be computed through Eq. 9.

$$\mathrm{NDCG}(k) = G^{-1}_{\max,\mathrm{i}}(k) \sum_{j:\pi_i(j) \le k} G(j)D(\pi_i(j)) \tag{9}$$

The multiple ranking lists might be created for a design problem with respect its association with design patterns. The position of the pattern $DP_{i,j}$ in each ranking list π_i is represented as $\pi_i(j)$.

6.2 Design pattern collections

In the domain of software engineering, some groups of experts have categorized the design patterns on the base of their intent, motivation, and applicability. In this study, we consider four well-known design pattern collections to evaluate the proposed method.

6.2.1 Gang-of-Four (GoF) design pattern collection

The GoF collection includes 23 object-oriented design patterns which are divided into three groups named Creational, Structural, and Behavioral. This case study includes 1465 non-repeated words of all 23 design patterns after removing the stop words and words stemming (Gamma et al. 1995).

6.2.2 Duyne's HCI design pattern collection

The Duyne's HCI collection includes 90 patterns which are divided into 12 groups named Site Genres, Navigation Framework, Powerful Homepage, Writing and Managing Content, Building Trust and Credibility, Basic e-Commerce, Advance e-commerce, Helping Customers Complete Tasks, Effective Page layout, Fast Site Research, Making Easy Navigation and Speeding up site. This case study includes 1575 non-repeated words of all 90 design patterns after removing the stop words and words stemming (Duyne et al. 2003).

6.2.3 Douglass design pattern collection

The Douglass collection includes 34 real-time system relevant design patterns which are divided into five categories named Concurrency, Safety, and Reliability, Distribution, Memory, and Resource. This case study includes 1271 non-repeated words of all 34 design patterns after removing the stop words and words stemming (Douglass 2002).

6.2.4 Security design pattern collection

The Security collection includes 46 security systems relevant design patterns which are divided into eight categories named SACA (System Access and Control Architecture), ACM (Access Control Model), IA (Identification and Authentication), OSAC (Operating System Access Control), SIA (Security Internet Application), FA (Firewall Architecture), ESRM (Enterprise Security and Risk Management) and Accounting (Schumacher et al. 2006). This case study includes 1230 non-repeated words of all 46 design patterns after removing the stop words and words stemming.

6.3 Design problems

Subsequently, we consider 47 real design problems from different resources (Bouhours et al. 2015; Silberschatz and Galvin 2002; Shalloway and Trott 2001) in order to evaluate the effectiveness of proposed methodology; however, we mention the description of only six design problems due to space limitation and rest of 41 design problem[4] are publically available.

Design Problem-1 (DP-1) "An arithmetic expression consists of an operand, an operator (+ − */), and another operand. The operand can be a number or another arithmetic expression. Thus, 2 + 3 and (2 + 3) + (4 * 6) are both valid expressions."

[4] https://docs.google.com/document/d/1sWzdLaPdvaFR7grmi3vmyhg4ksq3y1TqXt-Jm0zQgFM/edit?usp=sharing.

Design Problem-2 (DP-2) "The pilots of the planes approaching or departing the terminal area communicate with the tower rather than explicitly communicating with one another. The constraints on who can take off or land is enforced by the tower. It is important to note that the tower does not control the whole flight. It exists only to enforce constraints in the terminal area."

Design Problem-3 (DP-3) "When the browser loads a web page, it traverses through all images on that page. The browser loads all new images from Internet and places them the internal cache. For already loaded images, a fly-weight object is created, which has some unique data like position within the page, but everything else is referenced to the cached one."

Design Problem 4 (DP-4) "How to protect the programs from one another and the kernel from them all and how to handle relocation in case, when more than one program reside in main memory, waiting either for I/O (Input/Output) or CPU resource, for better CPU utilization."

Design Problem 5 (DP-5) "In the case of limited resources such as I/O devices, how the process should be a model to access these resources without any deadlock and starvation. For example, how the process (philosophers) should be a model of the dining philosophers problem, who try to access a limited number of resources (i.e., Fork)."

Design Problem 6 (DP-6) "A distributed system using the mailboxes has two IPC primitives, send and receive. The latter primitive specifics a process to receive from and blocks if no message from that process is available, even though the message may be waiting for other processes. Is deadlock possible, if there are no shared resources, but the process needs to communicate frequently"

7 Results and discussion on evaluation of proposed method

The proposed method is evaluated according to the evaluation model described in Sect. 6. In order to improve the homogeneity and completeness of features, the most appropriate weighting method (in terms of *F*-value) has been used for each respective ranking algorithm of the proposed method. However, the weighting method can be explicitly revealed. A ranking algorithm (with its best weighting method) of the proposed method is suggested as a better ranking algorithm in terms of MAP and NDCG measure.

7.1 Evaluation of ranking for design pattern collections

In order to determine the right number of design pattern classes with respect to ground reality, the effectiveness of ranking algorithms used in the proposed method is evaluated using the four design pattern collections and evaluation model described in Sects. 6.2 and 6.1 respectively. In each evaluation process, Binary, TFIDF, TFC, LTC, and Entropy are applied; however, the effectiveness of ranking algorithms is evaluated with the best weighting method.

7.1.1 Gang-of-Four (GoF) design patterns group evaluation

The patterns in the GoF collection are divided into three categories. According to *F*-value criterion, the evaluation result in the selection of the best weighting method for each ranking algorithm is shown in Fig. 3. In case of AdaRank, TDIDF with highest *F*-value (i.e., *F*-measure = 0.80) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.66), TFC(*F*-measure = 0.64), LTC (*F*-measure = 0.65) and Entropy (*F*-measure = 0.70). Subsequently, in case of Coordinate Ascent, TDIDF with highest *F*-value (i.e., *F*-measure = 0.80) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.69), TFC(*F*-measure = 0.70), LTC (*F*-measure = 0.67) and Entropy (*F*-measure = 0.72). Finally, in case of LambdaMART, LTC with highest *F*-value (i.e., *F*-measure = 0.72) is suggested with minor difference as a best weighting method as compared to Binary (*F*-measure = 0.60), TFC(*F*-measure = 0.68), TFIDF (*F*-measure = 0.66) and Entropy (*F*-measure = 0.62).

The design problems (related to the GoF pattern collection) are considered in the evaluation of the proposed method, and the evaluation results of ranking algorithms are shown in Table 2. The experimental results indicate that LambdaMART outperform than other rankers in terms of MAP and NDCG performance metrics. The average MAP value (i.e., 0.80) of LambdaMART indicates its effectiveness to select the right pattern for the given design problem(s). Subsequently, the average NDCG value (i.e., 0.85) of LambdaMART indicate its effectiveness to create

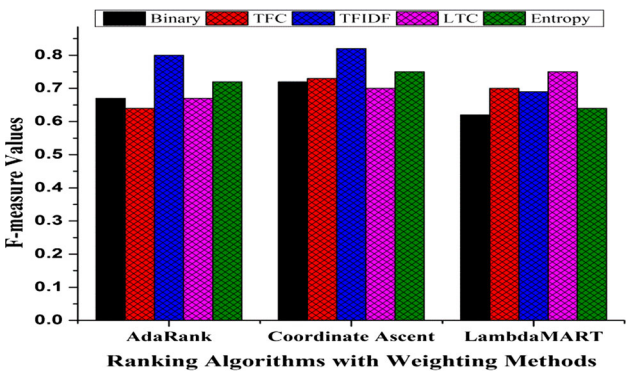


Fig. 3 Performance evaluation results (in terms of *F*-measure) for ranking functions on GoF collection

Table 2 Evaluation results of rankers (in terms of average *F*-measure) on design pattern collections

Ranker	Sample size	Performance metric	GoF collection	Duyne's collection	Douglass's collection	Security pattern collection
AdaRank	6 Problems	MAP	0.75	0.81	0.74	0.81
		NDCG	0.80	0.85	0.76	0.84
	47 Problems	MAP	0.73	0.77	0.70	0.78
		NDCG	0.79	0.81	0.72	0.77
Coordinate ascent	6 Problems	MAP	0.77	0.71	0.68	0.80
		NDCG	0.80	0.82	0.72	0.82
	47 Problems	MAP	0.75	0.67	0.63	0.73
		NDCG	0.78	0.80	0.70	0.77
LambdaMART	6 Problems	MAP	0.80	0.82	0.75	0.83
		NDCG	0.85	0.87	0.78	0.86
	47 Problems	MAP	0.78	0.79	0.71	0.75
		NDCG	0.81	0.82	0.72	0.85

the more appropriate pattern lists for the given design problem(s). The significant performance of LambdaMART describes the effective tuning of gradients of ranked positions of a given design problem (i.e., Query) and appropriate GoF design patterns (i.e., Documents), which rely on its ensemble nature, integration of performance measure during training and weight optimization via updating gradient. This thing depicts the right direction of each document (i.e., design pattern) with respect to a query (i.e., design problem).

7.1.2 Duyne's patterns group evaluation

The patterns in the Duyne's collection are divided into twelve categories. According to *F*-value criterion, the evaluation result in the selection of the best weighting method for each ranking algorithm is shown in Fig. 4.

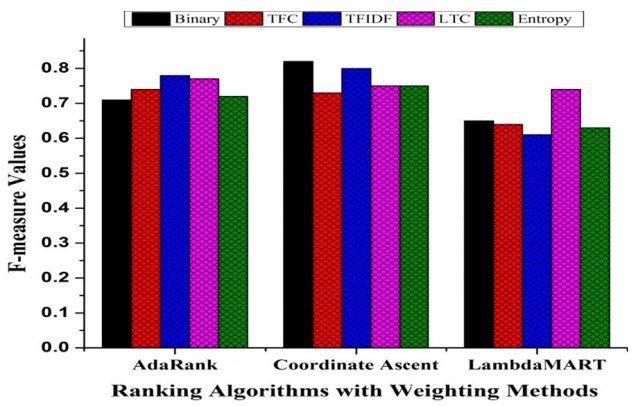


Fig. 4 Performance evaluation results (in terms of *F*-measure) for ranking functions on Dunye's collection

In case of AdaRank, TDIDF with highest *F*-value (i.e., *F*-measure = 0.77) is suggested with minor difference as a best weighting method as compared to Binary (*F*-measure = 0.70), TFC(*F*-measure = 0.74), LTC (*F*-measure = 0.76) and Entropy (*F*-measure = 0.72). Subsequently, in case of Coordinate Ascent, Binary with highest *F*-value (i.e., *F*-measure = 0.81) is suggested with minor difference as a best weighting method as compared to TFIDF (*F*-measure = 0.68), TFC(*F*-measure = 0.71), LTC (*F*-measure = 0.73) and Entropy (*F*-measure = 0.73). Finally, in case of LambdaMART, LTC with highest *F*-value (i.e., *F*-measure = 0.72) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.63), TFC(*F*-measure = 0.62), TFIDF (*F*-measure = 0.59) and Entropy (*F*-measure = 0.60).

The design problems (related to the Duyne's pattern collection) are considered in the evaluation of the proposed method, and the evaluation results of ranking algorithms are shown in Table 2. The experimental results indicate that LambdaMART and AdaRank outperform with minor difference than Coordinate Ascent; however, the average performance of LambdaMART remains better in terms of MAP and NDCG performance metrics. The average MAP values 0.82 and 0.81 of LambdaMART and AdaRank, respectively, indicate their effectiveness in order to select the right pattern. Subsequently, the average NDCG values that are 0.87 and 0.85 for LambdaMART and AdaRank, respectively, indicate their effectiveness to create a more appropriate pattern lists for the given design problem(s). Like LambdaMART, AdaRank also focuses on the weak learners that can work on the ranking process of poor queries. However, due to lack of ensemble nature and

performance measure integration in each step, AdaRank cannot outperform than LambdaMART. We observed significant performance of LambdaMART and AdaRank with minor difference.

7.1.3 Douglass patterns group evaluation

The patterns in the Douglass collection are divided into five categories. According to *F*-value criterion, the evaluation result in the selection of the best weighting method for each ranking algorithm is shown in Fig. 5.

In case of AdaRank, TDIDF with highest *F*-value (i.e., *F*-measure = 0.80) is suggested with minor difference as a best weighting method as compared to Binary (*F*-measure = 0.70), TFC(*F*-measure = 0.76), LTC (*F*-measure = 0.66) and Entropy (*F*-measure = 0.71). Subsequently, in case of Coordinate Ascent, TDIDF with highest *F*-value (i.e., *F*-measure = 0.78) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.70), TFC(*F*-measure = 0.74), LTC (*F*-measure = 0.69) and Entropy (*F*-measure = 0.74). Finally, in case of LambdaMART, LTC with highest *F*-value (i.e., *F*-measure = 0.74) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.64), TFC(*F*-measure = 0.65), TFIDF (*F*-measure = 0.66) and Entropy (*F*-measure = 0.68).

The design problems (related to the Douglass pattern collection) are considered in the evaluation of the proposed method, and the evaluation results of ranking algorithms are shown in Table 2. The experimental results indicate that LambdaMART and AdaRank outperform with minor difference than Coordinate Ascent; however, the average performance of LambdaMART remains better in terms of MAP and NDCG performance metrics. The average MAP values 0.74 and 0.75 of LambdaMART and AdaRank, respectively, indicate their effectiveness in order to select the right pattern. Subsequently, the average NDCG values that are 0.76 and 0.78 for LambdaMART and AdaRank indicate their effectiveness to create the more appropriate pattern lists for the given design problem(s).

Like Duynee's case study results, we observe significant performance of LambdaMART and AdaRank with minor difference. However, we observed poor performance of Coordinate Ascent which indicate non-smoothness and massive parallelism of function to rank.

7.1.4 Security patterns group evaluation

The patterns in the security collection are divided into eight categories. According to *F*-value criterion, the evaluation result for the selection of the best weighting method for each ranking algorithm is shown in Fig. 6.

In case of AdaRank, Binary with highest *F*-value (i.e., *F*-measure = 0.77) is suggested with minor difference as a best weighting method as compared to TFIDF (*F*-measure = 0.75), TFC(*F*-measure = 0.65), LTC (*F*-measure = 0.68) and Entropy (*F*-measure = 0.71). Subsequently, in case of Coordinate Ascent, TDIDF with highest *F*-value (i.e., *F*-measure = 0.84) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.78), TFC(*F*-measure = 0.74), LTC (*F*-measure = 0.75) and Entropy (*F*-measure = 0.73). Finally, in case of LambdaMART, TFIDF with highest *F*-value (i.e., *F*-measure = 0.78) is suggested with minor difference as a best weighting method as compared to Binary (*F*-measure = 0.72), TFC(*F*-measure = 0.72), LTC (*F*-measure = 0.75) and Entropy (*F*-measure = 0.73).

The design problems (related to security pattern collection) are considered in the evaluation of the proposed method, and the evaluation results of ranking algorithms are shown in Table 2. The experimental results indicate

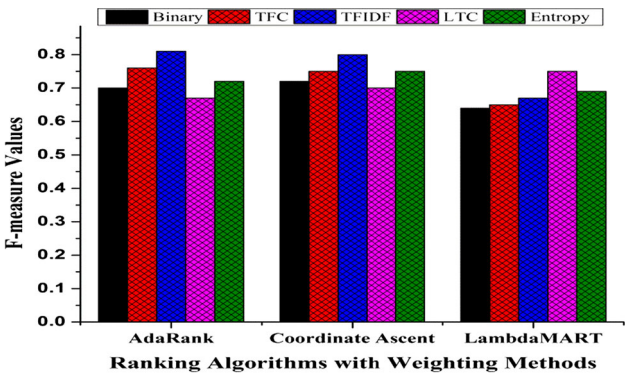


Fig. 5 Performance evaluation results (in terms of *F*-measure) for ranking functions on Douglass's collection

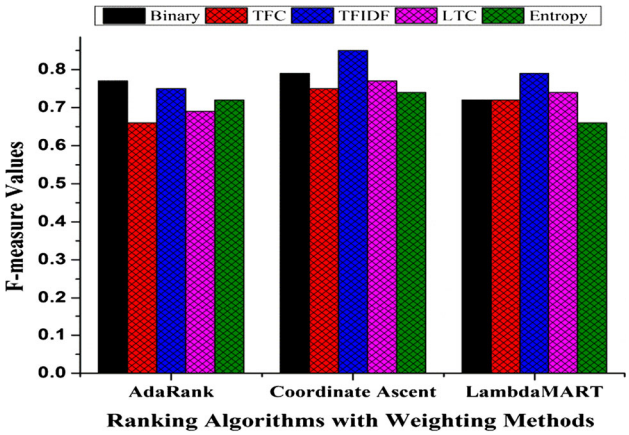


Fig. 6 Performance evaluation results (in terms of *F*-measure) for ranking functions on Security's collection

that LambdaMART outperforms with minor difference than other rankers in terms of MAP and NDCG performance metrics. The average MAP value (i.e., 0.83) of LambdaMART indicates its effectiveness to select the right pattern. Subsequently, the average NDCG value (i.e., 0.86) of LambdaMART indicates its effectiveness to create the more appropriate pattern lists for the given design problem(s).

7.1.5 Security patterns group evaluation

The patterns in the security collection are divided into eight categories. According to *F*-value criterion, the evaluation result for the selection of best weighting method for each ranking algorithm is shown in Fig. 6.

In case of AdaRank, Binary with highest *F*-value (i.e., *F*-measure = 0.77) is suggested with minor difference as a best weighting method as compared to TFIDF (*F*-measure = 0.75), TFC(*F*-measure = 0.65), LTC (*F*-measure = 0.68) and Entropy (*F*-measure = 0.71). Subsequently, in case of Coordinate Ascent, TDIDF with highest *F*-value (i.e., *F*-measure = 0.84) is suggested with significant difference as a best weighting method as compared to Binary (*F*-measure = 0.78), TFC(*F*-measure = 0.74), LTC (*F*-measure = 0.75) and Entropy (*F*-measure = 0.73). Finally, in case of LambdaMART, TFIDF with highest *F*-value (i.e., *F*-measure = 0.78) is suggested with minor difference as a best weighting method as compared to Binary (*F*-measure = 0.72), TFC(*F*-measure = 0.72), LTC (*F*-measure = 0.75) and Entropy (*F*-measure = 0.73).

The design problems (related to security pattern collection) are considered in the evaluation of the proposed method, and the evaluation results of ranking algorithms are shown in Table 2. The experimental results indicate that LambdaMART outperforms with minor difference than other rankers in terms of MAP and NDCG performance metrics. The average MAP value (i.e. 0.83) of LambdaMART indicates its effectiveness to select the right pattern. Subsequently, the average NDCG value (i.e., 0.86) of LambdaMART indicates its effectiveness to create the more appropriate pattern lists for the given design problem(s).

The ensemble nature, integration of performance measure during training and weight optimization via updating gradient of LmbdaMart help to rank the design patterns with respect to a given design problem effectively.

7.2 Evaluation of design problems

In this study, first, we evaluate the proposed method with 47 real design problems (6 out of 47 are given in Sect. 6.3) related to the design pattern collection (Sect. 6.2) and summarized the experimental results in Table 2. Consequently, we present the results with two sample of design problems. The first sample includes the six design problems mentioned in Sect. 6.3. The second sample includes the 47 design problems to evaluate the effectiveness of proposed methodology. Due to the meta-model similarity between patterns, the selection of an appropriate pattern for a given design problem becomes a challenging task for a novice designer. Consequently, the effectiveness of ranking algorithms used in the proposed method is evaluated in the creation of more appropriate candidate pattern list (in terms of NDCG) and recommendation of more appropriate patterns (in terms of MAP). For example, the effectiveness of ranking algorithms used in the proposed study can be evaluated on the basis of improvement in the NDCG to rank a pattern list for the given problem(s), more close to the list based on the ground truth with respect to expert's opinion. For example, the list of candidate patterns and most appropriate pattern for the given design problems (Sect. 6.3) ranked by outperformed ranker LambdaMART (from the results of Table 2) is shown in Table 3. The evaluation of ranking algorithms (in terms of NDCG) for the creation of more appropriate candidate pattern list for each design problem is shown in Fig. 7. The result of Fig. 7, indicates that for the first three and last design problems (i.e., Design_Problem_1, Design_Problem_2 and Design_Problem_3, Design_Problem_6), the Lambda_MART outperform than AdaRank and Coordinate Ascent, while for the

Table 3 Ranking list and ranked pattern of outperform LambdaMART ranker on design problems

Design problem	Patterns in catalog	Appropriate candidate list	Appropriate pattern
DP1	23	Composite, decorator, adaptor and strategy	Composite
DP2	23	Mediator, facade, adaptor and observer	Mediator
DP3	23	Flyweight, state, strategy and factory	Flyweight
DP4	34	Static allocation, fixed size buffer, pool allocation and smart pointer	Static allocation
DP5	34	Critical section, simultaneous locking, highest locker and order locking	Critical section
DP6	34	Shared memory, proxy, remote method call, observer	Shared memory

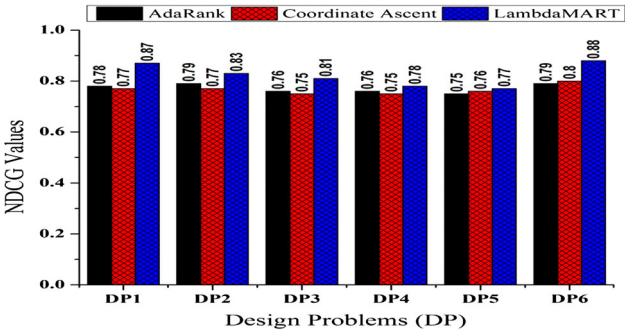


Fig. 7 Evaluation of ranking functions on the sample of six design problems

Design_Problem_4 and Design_Problem_5, we observe the LambdaMART and AdaRank outperform with the minor difference as compared to Coordinate Ascent.

8 Evaluation summary

1. We applied three ranking algorithms in the proposed method. In the evaluation results, we observe that the LambdaMART and AdaRank achieve better results as compared to Coordinate Ascent. However, LambdaMART presents better results with samples of 6 and 47 design problems, respectively, is analyzed.
2. We used different weighting methods (such as TFIDF, TFC, LTC, Binary, and Entropy) in order to increase the efficacy of learning on training data. Though in few cases we observe Binary and LTC as best weighting method, however, from the experimental results (Figs. 3, 4, 5, 6), we can conclude the suggestion of TFIDF as the best weighting method for the rankers.
3. We formulate the evaluation model of the proposed method using PAM and NDCG performance metrics. In terms of NDCG criterion, we can conclude that the proposed method is applicable to create more appropriate candidate pattern list for any given design problem.
4. Subsequently, in terms of PAM, the proposed method is also effective to select the right patterns on the base of ground truth described through expert's opinion.

9 Threats to validity

External validity This type of validity is concerned to restrict the generalization of the experimental results outside of their setting. Our proposed study satisfies the following features: "(1) This study is conducted by a group rather than an individual and (2) Professional have conducted this study." However, we found few external threats. The first threat is related to the number of design pattern collections and design problem(s) which are considered during the evaluation process of our proposed method. In order to evaluate the significant results of the proposed method, we will consider more design pattern collections and relevant examples of design problem(s). Subsequently, the reliability of the proposed method can be accessed by mining queries and solution patterns from websites of the different domain using the StakeExchange question answering system. The second threat relates to the incorporation of ranking algorithms. We customized each ranking algorithm using the default parameters; however, their performance can be reported when parameters are tuned.

Conclusion validity This type of validity is concerned with the statistical relationship between treatment and the outcome. In order to respond the main threat to conclusion validity that is correct inferences, we performed 10-fold cross-validation for each experiment. Moreover, we used *F*-measure, MAP and NDCG performance measure to rank the patterns.

Internal validity This type of validity is concerned with the use of experimental treatment variables and its manipulation to achieve better results. The performance of ranking algorithms can be affected due to existence of noisy modules.

Construct validity This type of validity is concerned with the degree to which the dependent and independent variables satisfy its measurement claim. However, we recommend a set of preprocessing activities. However, the performance of ranking algorithms can be affected due to noise features in datasets. If we applied certain sort of feature selection methods, then better performance of rankers can be achieved.

10 Conclusion

The experimental results illustrate that the proposed method based on text categorization approach via a supervised learning technique 'Learning to Rank' aid to provide a promising basis to select the more appropriate design pattern(s) for a given design problem. While existing automatic techniques based on UML-based, ontology-based, and text categorization approaches aim to reduce preprocessing cost and computation time in the domain of design pattern(s) selection. However, the formal specification, the existence of the ground reality of class labels for design patterns, and the adequate sample size are required for these automatic techniques to make precise learning. To address this issue, the proposed method is evaluated with the use of four design pattern collections and descriptions of numerous real design problems.

The proposed method has three phases: preprocessing, learning (on training dataset) and ranking phase (on testing dataset). The LambdaMART, AdaRank and Coordinate Ascent ranking algorithms are leveraged and used in the proposed method. We also proposed an evaluation model to evaluate the effectiveness of the proposed method in terms of creation of candidate pattern lists and selection of an appropriate pattern.

From the experimental results, we conclude several important consequences. First, the best weighting method for each ranking algorithm varies across the design pattern collections, consequently, the same weighting method cannot be recommended in proposed method. Second, in terms of PAM criterion, LambdaMART outperform than AdaRank and Coordinate Ascent with a minor difference in order to select more appropriate pattern. Thirds, in terms of NDCG criterion, LambdaMART present better performance than AdaRank and Coordinate Ascent with minor difference in order to create the appropriate pattern lists for the given design problems. In a future work, we will go to use the semantics of two contiguous words instead of the single word in the feature vector to improve the effectiveness of the proposed method.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Since in the article there is no human involvement for data gathering, consequently informed consent is not required.

References

Baraki H et al (2015) Interdisciplinary design patterns for socially aware computing. In: Proceeding of 37th international conference on software engineering (ICSE)

Bass L, Clements P, Kazman R (2012) Software architecture in practice, 3rd edn. Addison-Wesley Professional, Boston

Birukou A (2010) A survey of existing approaches for pattern search and selection. In: Proceeding of PLoP

Blomqvist E (2008) Pattern ranking for semiautomatic ontology construction. In: Proceedings of SAC

Booch G (2006) Handbook of software architecture, 2004. IBM Corporation. http://handbookofsoftwarearchitecture.com/

Bouhours C, Leblance H, Percebois C (2015) Spoiled patterns: how to extend the GoF. Softw Qual J 23:661–694

Buschmann F, Henney K, Schmidt DC (2007a) Pattern-oriented software architecture, vol 4. A pattern language for distributed computing. Wiley, New York

Buschmann F, Henney K, Schmidt DC (2007b) Pattern-oriented software architecture, vol 5. On patterns and pattern languages. Wiley, New York

Douglass BP (2002) Real-time design patterns: robust scalable architecture for real-time systems. Addison-Wesley, Boston

Duyne V et al (2003) The design of sites: patterns, principles and processes for crafting a customer-centered web experience. Addison Wesley, Boston

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Boston

Hasheminejad SMH, Jalili S (2012) Design patterns selection: an automatic two-phase method. J Syst Softw 85:408–424

Hasso S, Carlson CR (2005) A theoretically-based process for organizing design patterns. In: Proceedings of 12th pattern language of patterns

Henninger S, Correa V (2007) Software pattern communities: current practices and challenges. In: Proceedings of the 14th conference on pattern languages of programs

Hotho A, Nurnberger A, Paab G (2005) A brief survey of text mining. J Comput Linguist Lang Technol 20:19–62

Hsueh NL, Kuo J-Y, Lin C-C (2007) Object-oriented design: a goal-driven and pattern-based approach. J Softw Syst Model 8(1):1–18

Huang A (2008) Similarity measures for text document clustering. In: Proceedings of NZCSRSC

Hussain S et al (2016) A methodology to automate the selection of design patterns. In: Proceeding of 40th annual computer software and applications conference (COMPSAC)

Hussain S, Keung J, Khan AA (2017a) Software design patterns classification and selection using text categorization approach. Appl Soft Comput 58:225–244

Hussain S, Keung J, Khan AA (2017) A framework for ranking of software design patterns. In: Proceedings of 11th international conference on complex, intelligent, and software intensive systems, Jul 10–12, 2017

Issaoui I, Bouassida N, Abdallah HB (2015) A new approach for interactive design pattern recommendation. Lect Notes Softw Eng 3(3):173

Khoury PE, Mokhtari A, Coquery E, Hacid MS (2008) An ontological interface for software developers to select security patterns. In: Proceedings of 19th international conference on database and expert systems application, (DEXA'08), pp 297–301

Kim DK, Khawand CE (2007) An approach to precisely specifying the problem domain of design patterns. J Vis Lang Comput 18:560–591

Kim DK, Shen W (2008) Evaluating pattern conformance of UML models: a divide and conquer approach and case studies. Softw Qual J 16(3):329–359

Kircher M, Jain O (2004) Pattern-oriented software architecture, vol 3. Patterns for resource management. Wiley, New York

Li H (2011) A short introduction to 'learning to rank'. IEICE Trans Inf Syst 94(10):1854–1862

Medhat W, Hassan A, Korashy H (2014) Sentiment analysis algorithms and applications: a survey. Ain Shams Eng J 5:1093–1113

Metzler D, Croft WB (2007) Linear feature-based models for information retrieval. Inf Retr 10(3):257–274

Palma F, Farzin H, Gueheneuc Y-G (2012) Recommendation system for design patterns in software development: an DRP Ovrview. In: Proceeding of RSSE

Schmidt DC, Stal M, Rohnert H, Buschmann F (2000) Pattern-oriented software architecture, vol 2. Patterns for concurrent and networked objects. Wiley, New York

Schumacher M, Fernandez E, Hybertson D, Buschmann F (2006) Security patterns: integrating security and systems engineering. Wiley, New York

Shalloway A, Trott R (2001) Design pattern explained: a new perspective on object oriented design. Addison Wesley, Boston

Silberschatz A, Galvin PB, Gagne G (2002) Operating System Concepts, 6th edn. Wiley, USA
Uysal AK (2016) An improved global feature selection scheme for text classification. Expert Syst Appl 43:82–92
van Welie M (2006) Patterns in interaction design. http://www.welie.com/. Updated: 27 June 2006
Velasco-Elizondo P, Marin-Pina R, Vazquez-Reyes S, Mora-Soto A, Mejia J (2016) Knowledge representation and information extraction for analyzing architectural patterns. Sci Comput Program 121:176–189
Wood WG (2007) A practical example of applying attribute-driven design (ADD), version 2.0, technical report, SE Institute
Wu Q, Burges CJC, Svore K, Gao J (2007) Adapting boosting for information retrieval measures. J Inf Retr 13(3):254–270
Xu J, Li H (2007) AdaRank: a boosting algorithm for information retrieval. In: Proceedings of SIGIR, pp 391–398
Zhang C, Wu X, Niu Z, Ding W (2014) Authorship identification from unstructured texts. Knowl Based Syst 66:99–111