Information Sciences 179 (2009) 1040–1058

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem

Cagatay Catal *, Banu Diri

TUBITAK-Marmara Research Center, Information Technologies Institute, Gebze, Kocaeli 41470, Turkey

ARTICLE INFO

Article history:
Received 30 April 2008
Received in revised form 27 November 2008
Accepted 1 December 2008

Keywords:
Machine learning
Artificial Immune Systems
Software fault prediction
J48
Random Forests
Naive Bayes

ABSTRACT

Software quality engineering comprises of several quality assurance activities such as testing, formal verification, inspection, fault tolerance, and software fault prediction. Until now, many researchers developed and validated several fault prediction models by using machine learning and statistical techniques. There have been used different kinds of software metrics and diverse feature reduction techniques in order to improve the models' performance. However, these studies did not investigate the effect of dataset size, metrics set, and feature selection techniques for software fault prediction. This study is focused on the high-performance fault predictors based on machine learning such as Random Forests and the algorithms based on a new computational intelligence approach called Artificial Immune Systems. We used public NASA datasets from the PROMISE repository to make our predictive models repeatable, refutable, and verifiable. The research questions were based on the effects of dataset size, metrics set, and feature selection techniques. In order to answer these questions, there were defined seven test groups. Additionally, nine classifiers were examined for each of the five public NASA datasets. According to this study, Random Forests provides the best prediction performance for large datasets and Naive Bayes is the best prediction algorithm for small datasets in terms of the Area Under Receiver Operating Characteristics Curve (AUC) evaluation parameter. The parallel implementation of Artificial Immune Recognition Systems (AIRS2Parallel) algorithm is the best Artificial Immune Systems paradigm-based algorithm when the method-level metrics are used.

1. Introduction

Fault-prone modules can be automatically identified before testing phase by using the software fault prediction models. Generally, these models mostly use software metrics of earlier software releases and previous fault data collected during the testing phase. The benefits of these prediction models for software development practices are shown as follows [6]:

- Refactoring candidates can be identified from the fault prediction models, which require refactoring.
- Software testing process and software quality can be improved by allocating more resources on fault-prone modules.
- The best design approach can be selected from design alternatives after identifying the fault-prone classes by using class-level metrics.
- As fault prediction is an approach to reach dependable systems, we can have a highly assured system by using fault prediction models during development.

* Corresponding author. Tel.: +90 2626772634; fax: +90 2626463187.
E-mail address: cagatay.catal@bte.mam.gov.tr (C. Catal).

0020-0255/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2008.12.001

Since 1990s, researchers have applied different kinds of algorithms based on machine learning and statistical methods to build high-performance fault predictors. However, most of them used non-public datasets to validate their models. Even though some of the researchers claimed that their models provided the highest performance in fault prediction, some of these models revealed to be unsuccessful when evaluated on public datasets. Therefore, it is extremely crucial to benchmark available fault prediction models under different conditions on public datasets. In this study, we aimed to investigate the effects of dataset size, metrics set, and the feature selection techniques for fault prediction problem. PROMISE repository, created in 2005 [37], includes several public NASA datasets. In this study, we used five of these datasets called KC1, KC2, PC1, CM1, and JM1 which are widely used in fault prediction studies.

We implemented high-performance fault prediction algorithms based on machine learning for our benchmarking. We used Random Forests (RF) [29], J48 [26], and Naive Bayes [31] algorithms in our experiments. RF, J48, Naive Bayes are high-performance fault predictors. Ma et al. [29] stated that Random Forests algorithm provides the best performance for NASA datasets in terms of the G-mean1, G-mean2, and F-measure evaluation parameters. Menzies et al. [31] reported that Naive Bayes with logNums filter achieves the best performance in terms of the probability of detection (pd) and the probability of false alarm (pf) values. RF algorithm implements tens or hundreds of trees, to use later the results of these trees for classification. J48 is a decision tree implementation based on Quinlan's C4.5 algorithm. Naive Bayes is a probabilistic classification algorithm with strong independence assumptions among features.

In addition to the above mentioned algorithms, there were analyzed several classifiers based on Artificial Immune Systems (AIS) paradigm. We used Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, and AIRS2Parallel which are AIS-based algorithms. AIRS (Artificial Immune Recognition Systems) is an immune-inspired high-performance classification algorithm, which has five steps: initialization, antigen training, competition for limited resource, memory cell selection, and classification. Immunos81 is the first classification algorithm which is inspired from vertebrate immune system. Brownlee has developed Immunos1 and Immunos2, which are implementations of the generalized Immunos-81 system, in order to reproduce the results given by Carter and evaluate the algorithm [4]. CLONALG uses the clonal selection theory of immunity. CLONALG has a smaller number of user-defined parameters and a lower complexity than AIRS. It has been used for pattern recognition, function optimization and combinatorial optimization.

Public NASA datasets include 21 method-level metrics proposed by Halstead [17] and McCabe [30]. However, some researchers generally use only 13 metrics from these datasets [38], stating that derived Halstead metrics do not contain any extra information for software fault prediction. In his software measurement book, Munson [32] explains that the four primitive metrics of Halstead describe the variation of all of the rest of Halstead metrics. That is why there is nothing new to learn from the remaining derived Halstead metrics.

This study evaluated and benchmarked algorithms first using 21 metrics and then, by reducing the number of metrics to 13. After this operation, we compared the algorithms' performances and checked whether the best algorithm changed or not. These two different experiments were carried out to respond to the first and the second research questions.

Some researchers used feature reduction techniques such as Principal Component Analysis (PCA) to improve the performance of the models and to reduce multicollinearity [23]. This research area, developing feature selection techniques, is a popular area and there are recent studies to develop such techniques, even for heterogenerous data. Hu et al. [19] developed a neighborhood rough set model for feature selection of heterogeneous features because most of the approaches can be used only with numerical or categorical features. We applied correlation-based feature selection (cfs) technique to get relevant metrics before applying the algorithm. We achieved a higher performance with cfs method [7] and, therefore, we examined only this feature reduction technique to inspect the performance variation of algorithms in our experiment. This experiment was performed to respond to the third research question.

Software fault prediction process uses the method-level metrics (Halstead and McCabe metrics), but additionally it can use class-level metrics, too. Method-level metrics are suitable for both procedural and object-oriented programming paradigms, whereas, class-level metrics can be only calculated for programs developed with object-oriented programming languages. We evaluated the performance of the models in the KC1 dataset, because it is the only one that provides class-level metrics. In our previous study, we showed that Coupling Between Object Classes (CBO) metric is the most significant metric for fault prediction, whereas, and Depth of Inheritance Tree (DIT) is the least significant one [8]. Janes et al. [20] used Chidamber–Kemerer metrics and statistical models such as Poisson regression, negative binomial regression, and zero-inflated negative binomial regression to identify the fault-prone classes for real-time, telecommunication systems. The zero-inflated negative binomial regression model is based on the concept that Response for a Class (RFC) metric is the best feature to describe the variability of the number of defects in classes.

Three types of the experiments of this study, which used class-level metrics, were implemented to respond to the fourth, fifth, and sixth research questions. The last experiment did not use the cross-validation method and all the models used 20% of datasets for training and the rest as test set. This type of validation can be called as "split sample validation". In this experiment, we aimed to compare the 10-fold cross-validation with a validation of the model that uses an independent testing data. The split sample validation uses an independent test set, whereas the cross-validation uses the training data as a test set. The difference is that the independent test set represents a new random sample for the validation. Thus, the performance can change according to these two validation types. The last experiment was performed to respond to the seventh research question.

Research questions are shown as follows:

RQ1: Which of the machine learning algorithms performs best on large datasets when all the method-level metrics (21 metrics) are used?
RQ2: Which of the machine learning algorithms performs best on large datasets when only 13 method-level metrics used, instead of the derived Halstead metrics?
RQ3: Which of the machine learning algorithms performs best on large datasets when correlation-based feature selection technique is applied to 21 method-level metrics?
RQ4: Which of the machine learning algorithms performs best on public datasets (in this case, only KC1 dataset) when six Chidamber–Kemerer and lines of code metrics are used?
RQ5: Which of machine learning algorithms performs best on public datasets (in this case, only KC1 dataset) when correlation-based feature selection technique is applied to 94 class-level metrics?
RQ6: Which of the machine learning algorithms performs best on public datasets (in this case, only KC1 dataset) when 94 class-level metrics are used?
RQ7: Which of the machine learning algorithms performs best on large datasets when only 20% of datasets are used as training and the 10-fold cross-validation technique is not used?

To the best of our knowledge this is the first study to investigate the effects of dataset size, metrics set, and the feature selection techniques for the software fault prediction problem. In addition, there are implemented several high-performance predictors and algorithms based on Artificial Immune Systems paradigm.

This paper is organized as follows: the following section presents the related work. Section 3 explains the algorithms used during in our experiments. Section 4 describes the experiments, contexts, hypotheses, subjects, design of the experiments, instrumentation, validity evaluation, and experiment operations, whereas Section 5 shows the results of the analysis. Finally, Section 6 presents the conclusions and future works.

2. Related work

Researchers used different methods such as Genetic Programming [14], Decision Trees [24], Neural Networks [41], Naive Bayes [31], Dempster–Shafer Networks [16], Case-based Reasoning [12], Fuzzy Logic [48] and Logistic Regression [33] for software fault prediction. We developed and validated several Artificial Immune Systems-based models for software fault prediction problem [6–8]. Elish et al. [13] stated that the performance of Support Vector Machines (SVMs) is generally better than, or at least is competitive against the other statistical and machine learning models in the context of four NASA datasets. They compared the performance of SVMs with the performance of Logistic Regression, Multi-layer Perceptrons, Bayesian Belief Network, Naive Bayes, Random Forests, and Decision Trees. Gondra's [15] experimental results showed that Support Vector Machines provided higher performance than the Artificial Neural Networks for software fault prediction. Kanmani et al. [22] validated Probabilistic Neural Network (PNN) and Backpropagation Neural Network (BPN) using a dataset collected from projects of graduate students, in order to compare their results with results of statistical techniques. According to Kanmani et al.'s [22] study, PNN provided better performance. Quah [36] used a neural network model with genetic training strategy to predict the number of faults, the number of code changes required to correct a fault, the amount of time needed to make the changes and he proposed new sets of metrics for the presentation logic tier and data access tier.

Menzies et al. [31] showed that Naive Bayes with a logNum filter is the best software prediction model, even though it is a very simple algorithm. They also stated that there is no need to find the best software metrics group for software fault prediction because the performance variation of models with different metrics group is not significant [31]. Almost all the software fault prediction studies use metrics and fault data of previous software release to build fault prediction models, which are called "supervised learning" approaches in machine learning community. However, in some cases, there are very few or no previous fault data. Consequently, we need new models and techniques for these two challenging prediction problems.

Unsupervised learning approaches such as clustering methods can be used when there are not any previous fault data, whereas semi-supervised learning approaches can be used when there are very few fault data [9]. Zhong et al. [49] used Neural-Gas and K-means clustering algorithms to create the clusters and then an expert examined representative modules of clusters to assign the fault-proneness label. The performance of their model was comparable with classification algorithms [49]. In the cases when a company does not have previously collected fault data, or when a new project type is initiated, there can be used models based on unsupervised learning approaches. Seliya et al. [39] used Expectation–Maximization (E-M) algorithm for semi-supervised software fault prediction problem showing also that their model had comparable results with classification algorithms. So, the semi-supervised fault prediction models are required especially in the de-centralized software projects, where most of the companies find difficulties to collect fault data.

3. Algorithms

This section explains the details of each algorithm used during the experiments.

3.1. Artificial Immune Recognition Systems (AIRS) algorithms

Artificial Immune Systems (AIS) embody the principles and advantages of vertebrate immune system. They are used for intrusion detection, classification, optimization, clustering and search problems. Recently, there is an increasing interest in the use of AIS paradigm to solve complex problems. Powers et al. [35] used AIS paradigm to detect anomalous network connections and categorized them using a Kohonen Self Organising Map. The performance of the system they proposed showed favourable false positive and attack classification results on the KDD 1999 Cup. Lei et al. [27] proposed a fuzzy classification system using immune principles such as clonal selection and hypermutation. They applied this algorithm as a search strategy to mine fuzzy classification rule sets and compared to other approaches. Bereta et al. [1] developed immune K-means algorithm for data analysis and classification. They combined the clonal selection principle with negative selection for data clustering and their approach showed promising results. Ding et al. [11] developed an evolutionary framework for the distributed object computing (DOC) and their approach inspired by immune systems to improve the existing DOC infrastructures. Tavakkoli-Moghaddam et al. [40] proposed a hybrid multi-objective algorithm based on the features of an immune system and bacterial optimization to find Pareto optimal solutions for no-wait flow shop scheduling problem. They showed that their approach outperforms five algorithms for large-sized problems.

Artificial Immune Recognition System (AIRS) is a supervised learning algorithm, inspired by the vertebrate immune system. Until 2001, most of the studies in AIS were focused on unsupervised learning algorithms. Therefore, Watkins [44] decided to show that Artificial Immune Systems could be used for classification problems. The only exception was Carter's study [5] in 2000 who introduced a complex classification system based on AIS. Watkins [44] demonstrated that AIS implemented with supervised learning could be used for classification problems. This algorithm uses the resource-limited approach of Timmis et al. study [42] and the clonal selection principles of De Castro et al. [10] study. After the first AIRS algorithm, some researchers proposed a modified AIRS which had a higher complexity with a little reduction of accuracy [46]. Watkins [45] also developed a parallel AIRS. AIRS algorithm has powerful features which are listed as follows [2]:

- Generalization: The algorithm does not need all the dataset for generalization and it has data reduction capability.
- Parameter stability: Even though user-defined parameters are not optimized for the problem, the decline of its performance is very small.
- Performance: There has been demonstrated that its performance is excellent for some datasets and totally remarkable.
- Self-regulatory: There is no need to choose a topology before training.

AIRS algorithm has five steps: Initialization, antigen training, competition for limited resource, memory cell selection and classification [2]. The first step and the last step are applied only once, but steps 2, 3, 4 are used for each sample in the dataset. In the AIRS algorithm, the term sample is called antigen for AIRS algorithm. Brownlee [2] implemented this algorithm in Java language that can be currently accessed from http://sourceforge.net/projects/wekaclassalgos.net website. The activity diagram for this algorithm is shown in Fig. 1.

This algorithm is presented below, whereas further details can be attained from its source code distributed with GPL (General Public License) license. First, dataset is normalized for the training step and Affinity Threshold variable is calculated using the average Euclidean distance between antigens (each row in dataset). In the current implementation, 50 data points are taken into account. Furthermore, memory cell pool and Artificial Recognition Ball (ARB) pool are initialized with a random vector chosen from the dataset. After this step, antigen training step starts. This and following two steps (step 4 and 5) are repeated for all the data in dataset. In this second stage, antigens (training data) are presented to the memory pool one by one. Antibodies in the memory pool are stimulated with this antigen and a stimulation value is assigned to each single cell. The recognition cell which has a maximum stimulation value is selected as the best match memory cell and it is used for the affinity maturation process. This cell is cloned and mutated. Then, these clones are added to the Artificial Recognition Ball (ARB) pool. The number of clones are proportional to the stimulation value and it is calculated using

$$\text{NumClones} = (1 - \text{affinity}) * \text{clonalRate} * \text{hypermutationRate} \quad (1)$$

formula.

After the second step, competition for limited resource starts. Competition begins when mutated clones are added to the ARB pool. ARB pool is stimulated with antigens and the limited resources are assigned according to the stimulation values. ARBs which do not have enough resources are thrown away from the pool. If the stopping criteria are satisfied, the process stops. Otherwise, mutated clones of ARBs are generated and the recursion goes on until the stopping criteria are fulfilled. The stopping criterion is related to the stimulation level between ARBs and the antigen. If the stimulation level is still beyond the threshold level, the recursion goes on. After this third step, forth step starts. This is called as memoy cell selection phase.

In the previous step, when the stopping case is fulfilled, the ARB which has a maximum stimulation score is chosen as a candidate memory cell. If an ARB's stimulation value is higher than the original value of the best matching memory cell, ARB is copied to the memory cell pool. Step two, third and forth are done for each data point in dataset, but the first step (initialization) and the last step (classification) are done just one time. The last step for the algorithm is the classification step.

Learning process is completed before this step starts. Final memory cell pool is used for the cross-validation or to test new data. The test data is presented to the memory cell pool and the K most stimulated cells are identified. The majority label of these K cells determine the label of the test set.

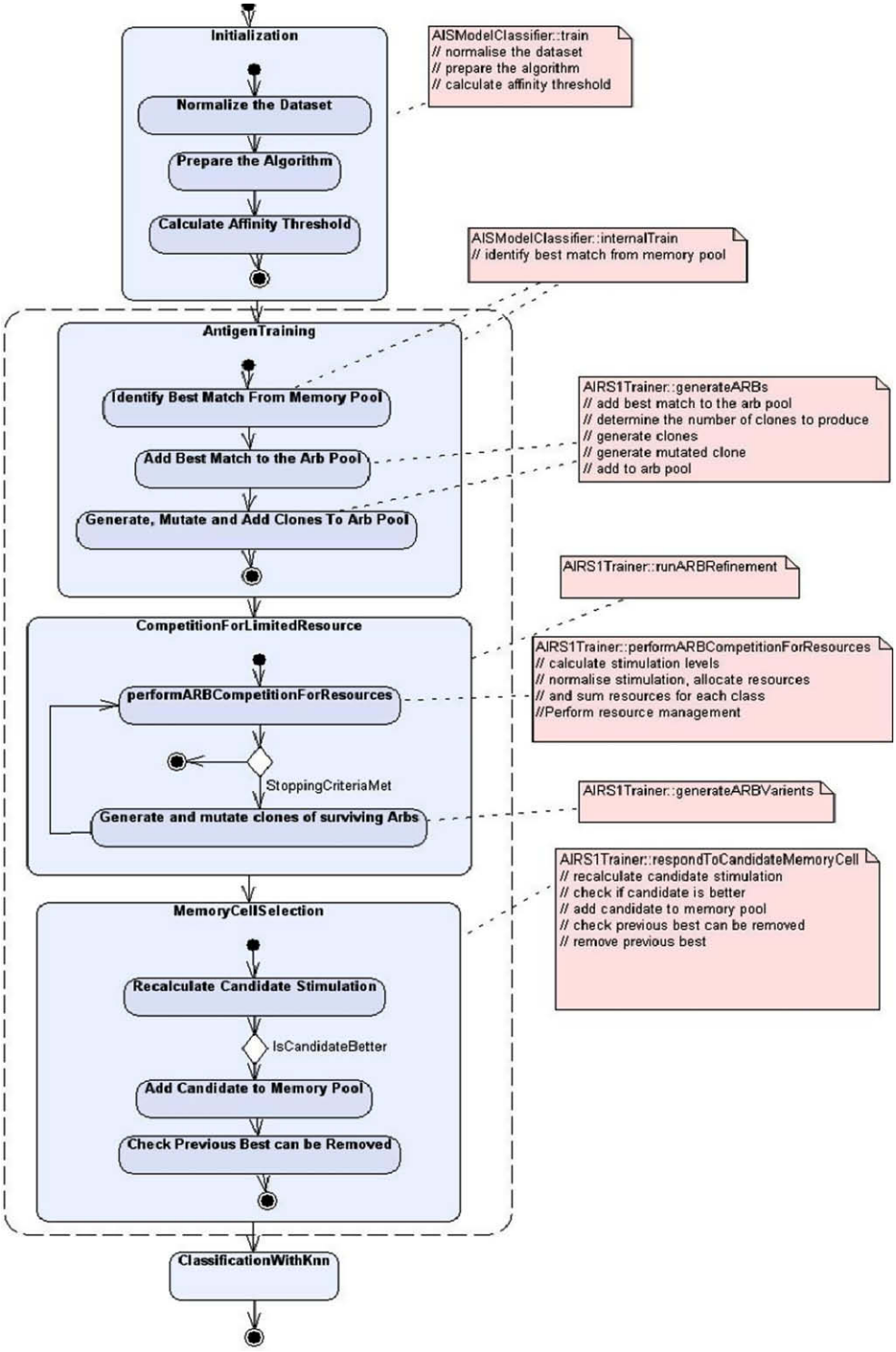

Fig. 1. This diagram shows the activity diagram of AIRS algorithm [8].

AIRSv1 is the first version of the algorithm, which is extremely complex. That's why, a second version was introduced. AIRSv2 has a lower complexity, a higher data reduction percentage and a few decreases in accuracy. Watkins [45] developed a parallel implementation of AIRS algorithm as part of his Ph.D. thesis and it is the third form of AIRS algorithms which was investigated in this study.

The main steps of AIRSv1 and AIRSv2 are generally similar but there are some minor differences. AIRSv2 is simpler than AIRSv1 and has a better data reduction percentage. AIRSv2 has a less performance degradation and a lower complexity than AIRSv1.

AIRSv1 uses the ARB pool as a permanent resource, whereas AIRSv2 uses it as a temporary resource. AIRSv2 approach is more accurate, because of the higher similarity between the antigens which have same class label and the ARBS. Previous ARBs which come from other ARB improvement transitions do not need to be evaluated continuously as in AIRSv1. Classes of clones can be changed after mutation process in AIRSv1, but this is not allowed in AIRSv2. AIRSv1 uses user-defined mutation parameter but AIRSv2 uses somatic hyper mutation concept. Mutation amount of clone is proportional to its affinity.

AIRS algorithm was implemented with parallel programming approach by Watkins [45]. The approach consists of the following steps [2]:

- Divide the training dataset into np partitions.
- Allocate training partitions to processes.
- Combine np number of memory pools.
- Use a merging approach to create the combined memory pool.

If the dataset is not divided into very small subsets, this approach provides a faster solution while the performance is preserved. Further researches should be done to solve the problem of the merging techniques, which lead to worse data reduction percentage [45].

3.2. CLONALG

Clonal selection algorithm (CLONALG) is inspired from the following elements [3]:

- Maintenance of a specific memory set.
- Selection and cloning of most stimulated antibodies.
- Death of non-stimulated antibodies.
- Affinity maturation.
- Re-selection of clones proportional to affinity with antigen.
- Generation and maintenance of diversity.

An antibody is a single solution to the problem and an antigen is a feature of the problem space. The aim here is to develop a memory pool of antibodies. The algorithm has two mechanisms to find the final pool. The first mechanism is a local search via affinity maturation of cloned antibodies, and the second one is a global search, which includes the insertion of randomly generated antibodies to the population to avoid the local optima problem [3]. An overview of the CLONALG algorithm is shown in Fig. 2.

Each step of this algorithm is explained below [3]:

- **Initialization**: The first step for CLONALG is to prepare a fixed size antibody pool. Later, this pool is separated into two sections. The first section represents the solution and the remaining pool is used for introducing diversity into the system.
- **Loop**: After the initialization phase, the algorithm executes G number of iterations, which can be customized by the user. The system is exposed to all of the known antigens in this phase.
  a. **Select Antigen**: An antigen is chosen at random from the antigen pool.
  b. **Exposure**: The selected antigen is shown to the antibodies and the affinity values showing the similarity between antibody and antigen are calculated. Generally Hamming or Euclid distance is used to calculate the affinity value.
  c. **Selection**: The highest affinity values are identified and a set of n antibodies, which have these highest values, is selected.
  d. **Cloning**: To introduce the diversity to the system, selected antibodies are cloned proportionally to their affinity values.
  e. **Affinity maturation (mutation)**: Duplicate antigens are mutated inversely proportionally to their parent's affinity values.
  f. **Clone exposure**: The clone is presented to the antigen and again affinity values are calculated.
  g. **Candidature**: Clones having the highest affinities are marked as candidate memory antibodies. The candidate memory antibody is replaced with the highest stimulated antigen from the memory pool if the affinity of this candidate memory cell is higher than that antigen.

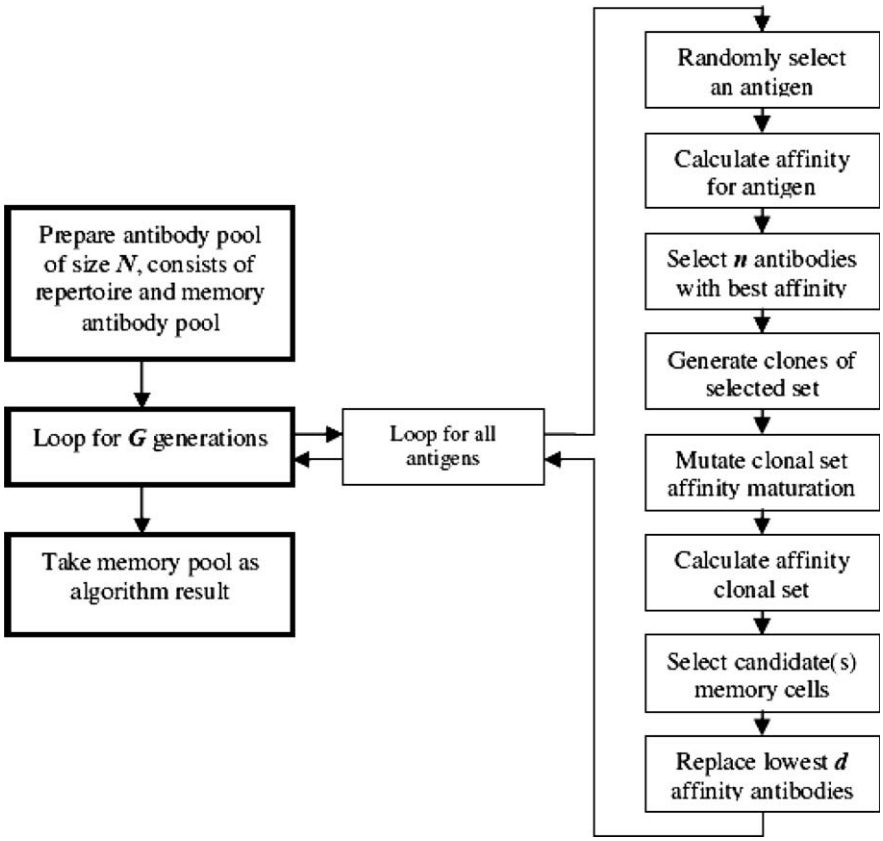

Fig. 2. Overview of the CLONALG algorithm [3].

h. **Replacement**: *d* antigens inside the remaining antigen pool ***r*** having the lowest affinity values are replaced with new random antibodies.

- **Finish**: After the training phase, memory section of antigen pool is considered as the solution of the problem.

This algorithm has a lower complexity and a smaller number of user parameters than the AIRS algorithm. CLONALG has seven user-defined parameters, explained as follows [3]:

- **antibody pool size (N)**: the total number of antibodies in the system, this then divided into a memory pool and a remainder pool.
- **clonal factor** (β): The factor used to scale the number of clones created for each selected antibody.
- **number of generations (G)**: The total number of generations to run for.
- **remainder pool ratio**: The ratio of the total antibodies (N) to allocate to the remainder antibody pool.
- **random number generator seed**: The seed for random number generator.
- **selection pool size (n)**: The total number of antibodies to select from the entire antibody pool for each antigen exposure.
- **total replacements (d)**: The total number of new random antibodies to insert into the remainder pool each antigen exposure.

3.3. Immunos algorithms

Immunos algorithm is the first AIS-based classification algorithm and it has been developed by a medical doctor, Carter [5]. Brownlee [3] developed two different versions of this algorithm in Java language. Browlee tried to contact Carter to discuss his Immunos-81 technique to clarify some points, but Carter was not reached. Thus, there are still two different implementations of this algorithm in Java language. According to Brownlee [3], Carter omitted some specifics of Immnos in his single publication on this algorithm.

Fig. 3 shows the training scheme and Fig. 4 shows the classification scheme of Immunos-81 algorithm.

"His model consisted of, T cells, B cells, and antibodies and an amino-acid library. Immunos-81 used the artificial T cells to control the production of B-cells. The B cells would then in turn compete for the recognition of the "unknowns". The amino-

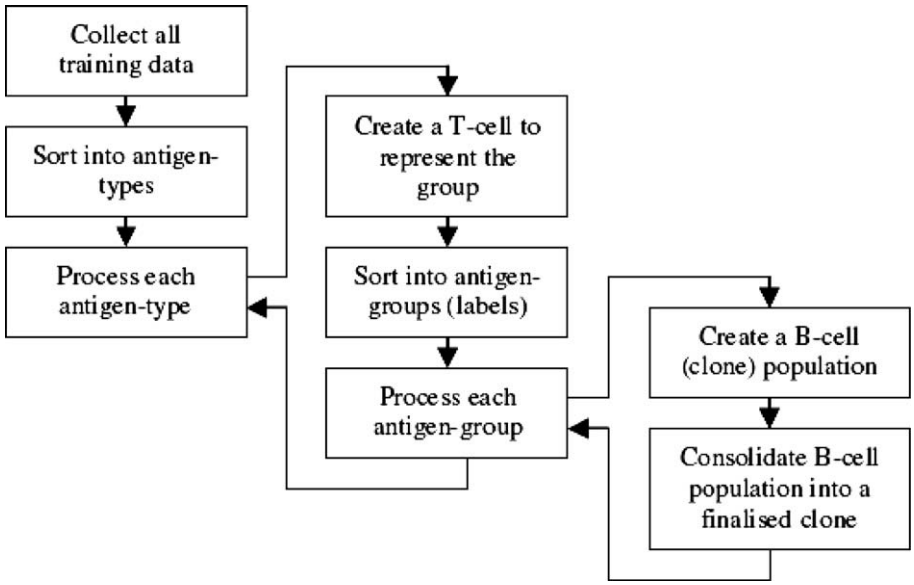

Fig. 3. Overview of the training scheme of Immunos-81 [4].

acid library acts as a library of epitopes (or variables) currently in the system. When a new antigen is introduced into the system, its variables are entered into this library. The T cells then use the library to create their receptors that are used to identify the new antigen. During the recognition stage of the algorithm T cell paratopes are matched against the epitopes of the antigen, and then a B cell is created that has paratopes that match the epitopes of the antigen" [43].

3.4. Random Forests

RF is a classification algorithm that includes tens or hundreds of trees. Results of these trees are used for majority voting, after which and RF chooses the class who has the highest votes. Random Forests provide an accurate classifier for many datasets. Koprinska et al. [25] showed that Random Forest is a good choice for filing e-mails into folders and spam e-mail filtering and outperforms algorithms such as decision trees and Naive Bayes. Because both the spam e-mail detection and software fault prediction datasets have unbalanced datasets, we used this algorithm. In addition, Ma et al. [29] stated that Random Forests (RF) algorithm provides the best performance for NASA datasets. Breiman's algorithm has been used in this study.

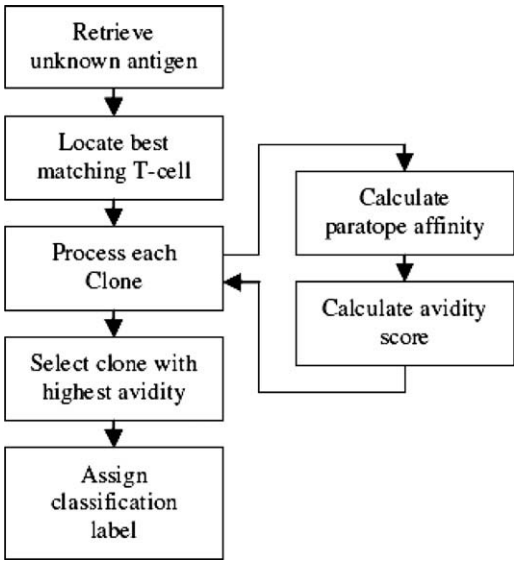

Fig. 4. Overview of the classification scheme of Immunos-81 [4].

"Each classification tree is built using a bootstrap sample of the data, and at each split the candidate set of variables is a random subset of the variables." [21]. "Let the number of training cases be *N*, and the number of variables in the classifier be *M*. We are told the number *m* of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*. Choose a training set for this tree by choosing *N* times with replacement from all *N* available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes. For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier)" [18].

3.5. J48 algorithm

J48 is Weka's (an open source machine learning tool) implementation of the C4.5 decision tree learning. Research on C4.5 algorithm was funded by the Australian Research Council for many years. Decision trees have internal nodes, branches, and terminal nodes. Internal nodes represent the attributes, terminal nodes show the classification result, and branches represent the possible values that the attributes can have. C4.5 is a well-known machine learning algorithm, whose details are skipped here.

3.6. Naive Bayes algorithm

Naive Bayes is a probabilistic classifier based on Bayes' theorem and it has strong independence assumptions. Naive Bayes assumes that the existence of a class's feature does not depend on the existence of the other features. When compared to the other algorithms, Naive Bayes requires a smaller data quantity to estimate the parameters. Naive Bayes is a well-known machine learning algorithm, whose details are skipped here.

4. Experiment description

This section describes in details the definition, context and variables selection, hypotheses formulation, selection of subjects, design of the experiment, instrumentation, validity evaluation, and the experiment operation.

4.1. Experiment definition

The objects of this study are the previous software metrics and the previous software fault data. The purpose is to identify the best software fault prediction algorithm when different groups of metrics are used. The quality focus is the reliability and the effectiveness of fault prediction models. The study is evaluated according to the researcher's point of view and this view is the perspective of this study. The experiment is performed by using the datasets of NASA projects and this is the context of this study [47].

4.2. Context selection

The experiment's context was the NASA projects. We used five public datasets called JM1, KC1, PC1, KC2, and CM1 belonging to several NASA projects. JM1 belongs to a real-time predictive ground system project and it consists of 10885 modules. This dataset has noisy measurements and 19% of modules are defective. JM1 is the largest dataset in our experiments implemented with C programming language. KC1 dataset, which is implemented in C++ programming language, belongs to a storage management project for receiving/processing ground data. It has 2109 software modules, 15% of which are defective. PC1 belongs to a flight software project for earth orbiting satellite. It has 1109 modules, 7% of which are defective and the implementation language is C. KC2 dataset belongs to a data processing project and it has 523 modules. C++ language is used for the KC2 implementation and 21% of modules have defects in KC2 dataset. CM1 belongs to a NASA spacecraft instrument project developed in C programming language. CM1 has 498 modules of which 10% are defective.

4.3. Variables selection

We used different independent variables to respond to the research questions. However, for all the experiments we used the fault-proneness label as the dependent variable. This label can be fault-prone or not fault-prone depending on the result of the prediction. Each experiment was conducted to respond to each research question. The independent metrics that were used during the experiments are explained as follows:

Experiment 1: 21 method-level metrics (McCabe and Halstead metrics).
Experiment 2: 13 method-level metrics.
Experiment 3: Metrics identified after the correlation-based feature selection technique on 21 metrics.
Experiment 4: 6 Chidamber–Kemerer metrics and lines of code metric.

Experiment 5: Identified metrics after the application of correlation-based feature selection technique on 94 class-level metrics.

Koru et al. [26] converted method-level metrics into class-level ones using minimum, maximum, average and sum operations for KC1 dataset. 21 method-level metrics were converted into 84 class-level metrics. There were used 84 metrics deriving from transformation and 10 metrics from class-level metrics to create 94 metrics. Object-oriented metrics are *Percent_Pub_Data*, *Access_To_Pub_Data*, *Dep_On_Child*, *Fan_In*, *Coupling_Between_Objects* (*CBO*), *Depth_Of_Inheritance_Tree* (*DIT*), *Lack_Of_Cohesion_Of_Methods* (*LCOM*), *Num_Of_Children* (*NOC*), *Response_For_Class* (*RFC*), and *Weighted_Method_Per_Class* (*WMC*). The object-oriented metrics are described below [8]:

- WMC (Weighted Methods per Class) is the # of methods which located in each class.
- DIT (Depth of Inheritance Tree) is the distance of the longest path from a class to the root in the inheritance tree.
- RFC (Response for a Class) is the # of methods which can be executed to respond a message.
- NOC (Number of Children) is the # of classes which are direct descendants for each class.
- CBO (Coupling between Object Classes) is the # of different non-inheritance-related classes to which a class is coupled.
- LCOM (Lack of Cohesion in Methods) is related to the access ratio of attributes.
- Percent_Pub_Data is the percentage of public or protected data.
- Access_To_Pub_Data is the amount of public or protected data access for a class.
- Dep_On_Child is shows whether a class is depends on a descendant or not.
- Fan_In is the # of calls by higher modules.

Experiment 6: There were used 94 class-level metrics.
Experiment 7: **There were used** only 13 method-level metrics, without employing the derived Halstead metrics. Additionally, cross-validation was not used in this experiment. We used 20% of datasets as a training set and the rest as a test set.

4.4. Hypotheses formulation

Hypotheses formulation is a crucial component of software engineering experiments. Ma et al. [29] stated that Random Forests (RF) algorithm provides the best performance for NASA datasets in terms of the G-mean1, G-mean2, and F-measure evaluation parameters. Consequently, our hypotheses are oriented on Random Forests algorithm.

Our hypotheses are listed as follows:

- **HP1: "Prediction with 21 method-level metrics" hypothesis**. RF is the best prediction algorithm for large datasets when 21 method-level metrics are used. (Null hypothesis: RF algorithm is not the best prediction algorithm for large datasets when 21 method-level metrics are used).
- **HP2: "Prediction with 13 method-level metrics" hypothesis**. RF is the best prediction algorithm for large datasets when 13 method-level metrics are used. (Null hypothesis: RF algorithm is not the best prediction algorithm for large datasets when 13 method-level metrics are used).
- **HP3: "Prediction with correlation-based feature selection technique on method-level metrics" hypothesis**. RF is the best prediction algorithm for large datasets when correlation-based feature selection is used on 21 method-level metrics. (Null hypothesis: RF is not the best prediction algorithm for large datasets when correlation-based feature selection is used on 21 method-level metrics.)
- **HP4: "Prediction with CK metrics suite" hypothesis**. RF is the best prediction algorithm when CK metrics suite is used together with lines of code metric. (Null hypothesis: RF is not the best prediction algorithm when CK metrics suite is used together with lines of code metric.)
- **HP5: "Prediction with correlation-based feature selection technique on 94 class-level metrics" hypothesis**. RF is the best prediction algorithm when cfs technique is applied to 94 class-level metrics. (Null hypothesis: RF is not the best prediction algorithm when cfs technique is applied to 94 class-level metrics.)
- **HP6: "Prediction with 94 class-level metrics" hypothesis**. RF is the best prediction algorithm when 94 class-level metrics are used. (Null hypothesis: RF is not the best prediction algorithm when 94 class-level metrics are used.)
- **HP7: "Prediction with 20% training sets" hypothesis**. RF is the best prediction algorithm when 20% of datasets are used as training and the rest as test sets. (Null hypothesis: RF is not the best prediction algorithm when 20% of datasets are used as training and the rest as test sets.)

4.5. Selection of subjects

The subjects are experienced developers working on the NASA projects. We chose five projects from NASA because they are widely used by researchers in the software fault prediction area.

4.6. Design of the experiment

We conducted seven different experiments with respect to seven research questions and seven hypotheses. The design is one factor (machine learning algorithm), with nine treatments (RF, J48, Naive Bayes, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, and AIRS2Parallel). All algorithms were tested with their default parameters.

4.7. Instrumentation and measurement

We did not provide any material or instrument to subjects of experiments, because software metrics and fault data were already collected by NASA Metrics Data Program team. Datasets from PROMISE repository were used for the experiments. This repository includes datasets with ARFF (Attribute-Relation File Format) format, which can be easily used in an open source machine learning tool called WEKA (Waikato Environment for Knowledge Analysis).

4.8. Validity evaluation

External validity: Threats to the external validity limit the generalization of the results outside the experimental setting. The system which can be used in an empirical study must have the following features: "(1) developed by a group, rather than an individual; (2) developed by professionals, rather than students; (3) developed in an industrial environment, rather than an artificial setting; (4): large enough to be comparable to real industry projects" [23]. Our experiments satisfy all the above mentioned requirements and the results can be generalized outside the experimental setting. However, the characteristics of the data can affect the performance of the models, and therefore, the best prediction algorithm can change for different domains and different systems.

Conclusion validity: "Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. Threats to conclusion validity are issues that affect the ability to draw the correct inferences" [23]. In this study, tests were performed using 10-fold cross-validation. Experiments were repeated 10 times to produce statistically reliable results. The performance of the classifiers was compared by using the Area Under Receiver Operating Characteristics Curve (AUC) values. The ROC curve plots probability of false alarm (PF) on the *x*-axis and the probability of detection (PD) on the *y*-axis. ROC curve was first used in signal detection theory to evaluate how well a receiver distinguishes a signal from noise and it is still used in medical diagnostic tests [50]. Since the accuracy and precision parameters are deprecated for unbalanced datasets, we used AUC values for benchmarking.

Internal validity: "Internal validity is the ability to show that results obtained were due to the manipulation of the experimental treatment variables" [23]. Poor fault prediction can be caused by noisy modules (noise can be located in metrics or dependent variable). It was reported [38] that JM1 dataset contains some inconsistent modules (identical metrics but different labels), which are removed by some researchers before their experiments. However, NASA quality assurance group did not report anything about these modules. Therefore, we did not eliminate such modules for our experiments. If we had eliminated these inconsistent modules, we would have obtained better defect predictors and we would probably have suggested another algorithm for software fault prediction problem.

Construct validity: "This refers to the degree to which the dependent and independent variables in the study measure what they claim to measure" [34]. As we used well-known and previously validated software metrics, we can state that these variables are satisfactory and there is no threat for construct validity. Method-level metrics and CK metrics suite are widely used in software fault prediction studies.

4.9. Experiment operation

The experiments used public NASA datasets directly, without performing any extra experiment. If we had to collect software metrics and fault data from an online project, we could have divided this operation into preparation and execution phases.

5. Analysis of the experiment

This section describes the experiments which respond to the research questions. Each subsection or experiment is related to one research question and one hypothesis.

5.1. Experiment #1 (RQ1)

All the experiments investigated the following algorithms: J48, RF, Naive Bayes, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, and AIRS2Parallel. Additionally, they all were performed by using a 10-fold cross-validation. Experiments were repeated 10 times to produce statistically reliable results. The performance of the classifiers were compared by using the Area Under Receiver Operating Characteristics Curve (AUC) values, because the datasets used are highly unbalanced. Moreover, researchers showed that AUC is statistically more consistent than accuracy for balanced or unbalanced datasets. AUC is widely used for the problems having unbalanced datasets. For example, Liu et al. [28] proposed a weighted rough set-based

Table 1
21 method-level metrics.

Attributes	Information
loc	McCabe's line count of code
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead delivered bugs
t	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAndComment	Lines of comment and code
uniq_Op	Unique operators
uniq_Opnd	Unique operands
total_Op	Total operators
total_Opnd	Total operands
branchCount	Branch count of the flow graph

method for the class imbalance problem and showed that this approach is better than the re-sampling and filtering-based methods in terms of AUC. Consequently, we calculated the accuracy values of models to check if a model having high AUC value has really a high accuracy value. For example, Table 7 shows that Immunos2 algorithm on PC1 dataset provides 93.06% accuracy. However, its AUC value is 0.50 which is equal to the performance of random guessing as shown in Table 6. Therefore, we must not use this algorithm on this kind of dataset and problem. The higher the AUC value, the better is the performance.

This experiment used all the method-level metrics (21 metrics) located in the datasets. These 21 metrics are shown in Table 1. Table 2 presents the AUC values of the algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 3 shows accuracy values of algorithms for this experiment.

As mentioned before, JM1 and PC1 datasets are the largest datasets in our experiments. According to Table 2, RF algorithm provides the best prediction performance for JM1 and PC1. Therefore, our hypothesis #1 (HP1) is valid and the answer to the first research question is RF algorithm. In addition to RF algorithm, the performance of Naive Bayes algorithm is remarkable and its performance is better than RF for small datasets such as KC2 and CM1. It is a well-known fact that machine learning algorithms perform better when datasets are large and that is why RF works well in these datasets.

The parallel implementation of Artificial Immune Recognition Systems (AIRS) algorithm is the best fault prediction algorithm based on AIS paradigm for large datasets in this experiment. The performance of Immunos1, Immunos2 and CLONALG algorithms for PC1 and JM1 datasets are very low and they cannot be used for large datasets. The accuracy values of Immunos1 for PC1 and JM1 are 15.84% and 56.23%, respectively. The performance of AIRS1 algorithm is better than AIRS2, but lower than AIRS2Parallel. We suggest using the RF algorithm for large datasets and Naive Bayes algorithm for small datasets.

5.2. Experiment #2 (RQ2)

This experiment used 13 method-level metrics, without employing the derived Halstead metrics. Table 4 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 5 shows accuracy values of algorithms for this experiment.

Table 2
AUC values of algorithms for the experiment #1.

Algorithms	KC1 (AUC)	KC2 (AUC)	PC1 (AUC)	CM1 (AUC)	JM1 (AUC)
J48	0.70	0.69	0.64	0.56	0.66
RandomForests	**0.79**	0.80	**0.81**	0.70	**0.72**
NaiveBayes	**0.79**	**0.84**	0.71	**0.74**	0.69
Immunos1	0.68	0.68	0.53	0.61	0.61
Immunos2	0.51	0.72	0.50	0.50	0.50
CLONALG	0.52	0.58	0.51	0.50	0.51
AIRS1	0.60	0.69	0.56	0.55	0.55
AIRS2	0.57	0.67	0.57	0.53	0.54
AIRS2Parallel	0.61	0.69	0.57	0.54	0.56

Table 3
Accuracy values of algorithms for the experiment #1.

Algorithms	KC1 (ACC)	KC2 (ACC)	PC1 (ACC)	CM1 (ACC)	JM1 (ACC)
J48	84.04	81.19	93.63	88.05	79.81
RandomForests	85.37	82.66	93.48	88.68	80.95
NaiveBayes	82.46	83.62	89.00	84.84	80.42
Immunos1	50.57	53.49	15.84	32.74	56.23
Immunos2	75.18	65.90	93.06	88.92	80.67
CLONALG	82.46	77.94	90.61	87.72	74.21
AIRS1	77.11	78.38	89.40	82.67	69.76
AIRS2	73.10	79.60	91.37	85.92	69.56
AIRS2Parallel	81.94	81.98	91.86	86.00	71.48

According to Table 4, RF algorithm again provides the best prediction performance for JM1 and PC1 when 13 metrics are used. AUC values of RF and Naive Bayes algorithms did not change significantly. However, AUC value of AIRS2Parallel algorithm decreased, but the performance did not vary in considerable quantities. Our hypothesis #2 (HP2) is valid and the answer to the second research question is RF algorithm. The parallel implementation of Artificial Immune Recognition Systems (AIRS) algorithm is again the best fault prediction algorithm based on AIS paradigm for large datasets in this experiment. The performance of Immunos1, Immunos2 and CLONALG algorithms for PC1 and JM1 datasets are again very low and they cannot be used for large datasets. Experiment 1 together with experiment 2 show that derived Halstead metrics do not change the performance considerably and therefore, there is no need to collect these derived metrics for software fault prediction problem. According to this experiment, we again suggest the implementation of the RF algorithm for large datasets and the Naive Bayes algorithm for small datasets.

5.3. Experiment #3 (RQ3)

This experiment used correlation-based feature selection (cfs) technique to reduce the multicollinearity. It was developed by using 21 method-level metrics for each dataset. The chosen metrics with this technique are listed as follows:

CM1:	loc, iv(g), i, lOComment, lOBlank, uniq_Op, uniq_Opnd	→7 metrics
JM1:	loc, v(g), ev(g), iv(g),i, lOComment, lOBlank, locCodeAndComment	→8 metrics
KC1:	v, d, i, lOCode, lOComment, lOBlank, uniq_Opnd, branchCount	→8 metrics
KC2:	ev(g), b, uniq_Opnd	→3 metrics
PC1:	v(g), I, lOComment, locCodeAndComment, lOBlank, uniq_Opnd	→6 metrics

Table 4
AUC values of algorithms for the experiment #2.

Algorithms	KC1 (AUC)	KC2 (AUC)	PC1 (AUC)	CM1 (AUC)	JM1 (AUC)
J48	0.70	0.70	0.65	0.53	0.69
RandomForests	**0.80**	0.81	**0.79**	0.72	**0.72**
NaiveBayes	0.79	**0.84**	0.70	**0.74**	0.66
Immunos1	0.71	0.73	0.64	0.63	0.63
Immunos2	0.50	0.51	0.50	0.50	0.50
CLONALG	0.53	0.61	0.50	0.50	0.52
AIRS1	0.59	0.69	0.56	0.55	0.53
AIRS2	0.59	0.65	0.56	0.54	0.56
AIRS2Parallel	0.60	0.67	0.58	0.53	0.56

Table 5
Accuracy values of algorithms for the experiment #2.

Algorithms	KC1 (ACC)	KC2 (ACC)	PC1 (ACC)	CM1 (ACC)	JM1 (ACC)
J48	84.63	81.36	93.48	88.60	80.22
RandomForests	85.77	83.55	93.84	88.45	80.66
NaiveBayes	82.54	83.68	88.80	86.22	80.43
Immunos1	56.16	61.37	37.00	40.25	62.58
Immunos2	84.54	79.47	93.06	90.16	80.67
CLONALG	81.55	78.08	92.19	88.38	77.30
AIRS1	75.69	76.88	87.16	82.07	63.10
AIRS2	78.11	76.01	88.83	84.91	72.49
AIRS2Parallel	80.73	80.88	91.31	85.59	71.48

Table 6 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 7 shows accuracy values of algorithms for this experiment.

According to Table 6, RF algorithm provides the best prediction performance for JM1 and PC1 when cfs technique is applied to 21 method-level metrics. AUC values of RF and Naive Bayes algorithms did not change significantly. However, AUC value of AIRS2Parallel algorithm decreased, but the performance varied in inconsiderable quantities. Our hypothesis #3 (HP3) is valid and the answer to the third research question is RF algorithm. The parallel implementation of Artificial Immune Recognition Systems (AIRS) algorithm is the best fault prediction algorithm based on AIS paradigm for large datasets in this experiment. Also, AIRS2 provides same performance with AIRS2Parallel. The performance of Immunos1, Immunos2 and CLONALG algorithms for PC1 and JM1 datasets are once more very low and this shows that they cannot be used for large datasets. The first three experiments show that the chosen metrics are not very important for software fault prediction. The most essential component is the chosen classification algorithm or the prediction model. According to this experiment, we once more suggest the RF algorithm for large datasets and the Naive Bayes algorithm for small datasets.

5.4. Experiment #4 (RQ4)

In this experiment, we used class-level and lines of code metrics. As only KC1 dataset includes class-level metrics, we applied nine algorithms on KC1 by using six Chidamber–Kemerer metrics and lines of code metric. KC1 is not a very large dataset and, therefore, it includes fewer metrics and fewer data points than large datasets such as JM1.

Table 8 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 9 shows accuracy values of algorithms for this experiment.

According to Table 8, RF algorithm yet again provides the best prediction performance when class-level metrics are used together with lines of code metric. Furthermore, Naive Bayes, J48, and Immunos2 provide remarkable results for this dataset. It is interesting to note that the performance of Immunos2 algorithm is better than AIRS2 algorithm in this experiment. The AUC value of Immunos2 algorithm on KC1 dataset is 0.51 as shown in Table 2 when 21 method-level metrics are used. According to Table 8, now its AUC value is 0.72. Therefore, we can conclude that the class-level metrics improved the performance of Immunos2 algorithm significantly. Similarly the performance of AIR2Parallel algorithm increased. However, the performance of RF and Naive Bayes did not record any improvement when using the class-level metrics. Our hypothesis #4 (HP4) is valid and the answer to the fourth research question is RF algorithm. Immunos2 algorithm is the best fault prediction algorithm based on AIS paradigm when the class-level metrics are used. However, we must apply this algorithm in other datasets before generalizing this result. In addition, we can conclude that AIS-based algorithms provide better results when class-level metrics are used.

Table 6
AUC values of algorithms for the experiment #3.

Algorithms	KC1 (AUC)	KC2 (AUC)	PC1 (AUC)	CM1 (AUC)	JM1 (AUC)
J48	0.70	0.80	0.69	0.51	0.66
RandomForests	0.79	0.78	**0.81**	0.65	**0.71**
NaiveBayes	**0.80**	**0.84**	0.75	**0.76**	0.67
Immunos1	0.68	0.67	0.70	0.70	0.60
Immunos2	0.49	0.52	0.50	0.50	0.50
CLONALG	0.52	0.62	0.50	0.50	0.53
AIRS1	0.60	0.65	0.58	0.54	0.54
AIRS2	0.60	0.68	0.58	0.52	0.56
AIRS2Parallel	0.60	0.66	0.58	0.52	0.56

Table 7
Accuracy values of algorithms for the experiment #3.

Algorithms	KC1 (ACC)	KC2 (ACC)	PC1 (ACC)	CM1 (ACC)	JM1 (ACC)
J48	84.50	84.60	93.42	89.22	80.94
RandomForests	85.04	80.83	93.72	87.97	80.02
NaiveBayes	82.40	84.25	89.17	86.32	80.43
Immunos1	50.05	50.58	59.17	69.61	60.16
Immunos2	80.24	69.05	93.06	90.16	80.67
CLONALG	82.42	80.48	91.61	88.56	76.62
AIRS1	75.85	74.83	89.03	82.85	64.32
AIRS2	77.16	77.12	90.49	84.70	72.03
AIRS2Parallel	78.67	75.22	89.84	85.32	70.29

5.5. Experiment #5 (RQ5)

In this experiment, we applied cfs technique on 94 metrics and identified relevant metrics. Koru et al. [26] converted method-level metrics into class-level ones using minimum, maximum, average and sum operations for KC1 dataset. 21 method-level metrics were converted into 84 class-level metrics. There were used 84 metrics deriving from transformation and 10 metrics from class-level metrics to create 94 metrics. After the cfs technique was applied, the following metrics were identified: *CBO, maxLOC_COMMENTS, maxHALSTEAD_CONTENT, maxHALSTEAD_VOLUME, maxLOC_TOTAL, avgHALSTEAD_LEVEL, avgLOC_TOTAL.* We used nine algorithms on KC1 dataset by employing these metrics and calculated AUC values for each algorithm.

Table 10 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 11 shows accuracy values of algorithms for this experiment.

According to Table 10, Naive Bayes algorithm provides the best prediction performance when cfs is applied on 94 class-level metrics. Furthermore, J48, Random Forests, and Immunos2 provide remarkable results for this dataset. It is interesting to note that the performance of Immunos2 algorithm is better than AIRS2 algorithm in this experiment. Our hypothesis #5 (HP5) is not valid and the answer to the fifth research question is Naive Bayes algorithm. Immunos2 algorithm is the best fault prediction algorithm based on AIS paradigm when cfs is applied to 94 class-level metrics. However, we must apply this algorithm in other datasets before generalizing this result. The best performance is achieved when the above mentioned seven metrics and the Naive Bayes are used together. The AUC value of Naive Bayes is 0.82 for this case, being also the highest value in all the experiments. As a result, we can conclude that AIS-based algorithms provide better results when class-level metrics are used.

Table 8
AUC values of algorithms for the experiment #4.

Algorithms	KC1-classLevel (AUC)
J48	0.75
RandomForests	0.79
NaiveBayes	0.76
Immunos1	0.69
Immunos2	0.72
CLONALG	0.67
AIRS1	0.70
AIRS2	0.71
AIRS2Parallel	0.68

Table 9
Accuracy values of algorithms for the experiment #4.

Algorithms	KC1-classLevel (ACC)
J48	68.78
RandomForests	71.08
NaiveBayes	69.55
Immunos1	63.50
Immunos2	70.49
CLONALG	68.65
AIRS1	71.10
AIRS2	71.90
AIRS2Parallel	70.15

Table 10
AUC values of algorithms for the experiment #5.

Algorithms	KC1-classLevel (AUC)
J48	0.74
RandomForests	0.79
NaiveBayes	0.82
Immunos1	0.69
Immunos2	0.74
CLONALG	0.69
AIRS1	0.71
AIRS2	0.72
AIRS2Parallel	0.71

5.6. Experiment #6 (RQ6)

In this experiment, we used 94 class-level metrics. Table 12 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 13 shows accuracy values of algorithms for this experiment.

According to Table 12, Naive Bayes algorithm provides the best prediction performance when 94 class-level metrics are used. Furthermore, Random Forests and J48 provide remarkable results for this dataset. Our hypothesis #6 (HP6) is not valid and the answer to the sixth research question is Naive Bayes algorithm. AIS-based algorithms provide better performance when feature selection technique is applied, that is why the performance of Immunos2 improved when feature selection technique (cfs) was used. However, in this experiment, its performance is lower than the previous one. provide better performance when feature selection technique is applied. Immunos2 algorithm is the best fault prediction algorithm based on AIS paradigm when 94 class-level metrics are applied, but we must apply this algorithm in other datasets in order to generalize this result.

5.7. Experiment #7 (RQ7)

In this study, we did not apply the cross-validation technique. Instead of the cross-validation, we used 20% of datasets as training and the rest (80%) as test set. We checked whether the best algorithm would change or not when the cross-validation was not applied. 13 method-level metrics were employed in this experiment, without using the derived Halstead metrics. Table 14 presents AUC values of algorithms for this experiment. We also computed accuracy values of algorithms to check if the accuracy level is acceptable for the relevant algorithm. Table 15 shows accuracy values of algorithms for this experiment.

Table 11
Accuracy values of algorithms for the experiment #5.

Algorithms	KC1-classLevel (ACC)
J48	69.99
RandomForests	70.54
NaiveBayes	71.27
Immunos1	63.79
Immunos2	70.91
CLONALG	69.25
AIRS1	70.84
AIRS2	72.83
AIRS2Parallel	71.98

Table 12
AUC values of algorithms for the experiment #6.

Algorithms	KC1-classLevel (AUC)
J48	0.70
RandomForests	0.80
NaiveBayes	0.81
Immunos1	0.67
Immunos2	0.69
CLONALG	0.68
AIRS1	0.64
AIRS2	0.64
AIRS2Parallel	0.64

Table 13
Accuracy values of algorithms for the experiment #6.

Algorithms	KC1-classLevel (ACC)
J48	67.70
RandomForests	71.58
NaiveBayes	68.59
Immunos1	61.17
Immunos2	66.59
CLONALG	69.42
AIRS1	65.15
AIRS2	65.15
AIRS2Parallel	65.66

Table 14
AUC values of algorithms for the experiment #7.

Algorithms	KC1 (AUC)	KC2 (AUC)	PC1 (AUC)	CM1 (AUC)	JM1 (AUC)
J48	0.64	0.70	0.57	0.55	0.63
RandomForests	0.74	0.78	0.72	0.66	0.68
NaiveBayes	0.79	0.84	0.68	0.73	0.67
Immunos1	0.70	0.72	0.60	0.63	0.62
Immunos2	0.50	0.55	0.50	0.50	0.50
CLONALG	0.53	0.61	0.51	0.51	0.51
AIRS1	0.59	0.67	0.55	0.56	0.56
AIRS2	0.58	0.65	0.54	0.53	0.55
AIRS2Parallel	0.58	0.67	0.53	0.56	0.55

Table 15
Accuracy values of algorithms for the experiment #7.

Algorithms	KC1 (ACC)	KC2 (ACC)	PC1 (ACC)	CM1 (ACC)	JM1 (ACC)
J48	83.96	83.31	92.50	87.24	80.22
RandomForests	84.20	82.56	93.04	88.52	79.76
NaiveBayes	82.69	83.98	86.96	**83.22**	80.39
Immunos1	55.27	61.29	39.63	40.01	64.05
Immunos2	84.55	80.19	93.04	90.18	80.67
CLONALG	79.89	78.28	92.31	87.42	67.80
AIRS1	78.05	77.80	86.49	79.88	72.33
AIRS2	79.36	80.07	90.93	80.94	71.49
AIRS2Parallel	80.15	80.84	90.59	85.68	74.35

According to Table 14, RF algorithm provides the best prediction performance when the cross-validation technique is not applied. RF algorithm provides the best prediction performance for PC1 and JM1 datasets. Our hypothesis #7 (HP7) is valid and the answer to the seventh research question is RF algorithm. In this case, AIRS1 algorithm is the best fault prediction algorithm based on AIS paradigm when the cross-validation technique is not applied. The performance of algorithms did not change significantly. As a result, we can conclude that RF is the best prediction algorithm for large datasets and Naive Bayes is much more suitable for smaller datasets.

6. Summary and conclusions

In this study, we identified and used high-performance fault prediction algorithms based on machine learning for our benchmarking. We used public NASA datasets from PROMISE repository to make our predictive models repeatable, refutable, and verifiable. The seven test groups were identified to respond to the research questions and nine algorithms were evaluated in each of the five public datasets. However, three test groups were performed on only one dataset because the class-level metrics did not exist in the other datasets.

To the best of our knowledge this is the first study which investigates the effects of dataset size, metrics set, and the feature selection techniques for software fault prediction problem. Furthermore, we employed several algorithms belonging to a new computational intelligence paradigm called Artificial Immune Systems.

According to the experiments we conducted, hypotheses HP1, HP2, HP3, HP4, and HP7 are valid, whereas HP5 and HP6 are not valid. In most of the experiments, RF algorithm mostly provided the best performance for large datasets and Naive Bayes algorithm operated better than RF for small datasets. AIRS2Parallel algorithm is the best AIS-based prediction algorithm when method-level metrics are used. Immunos2 algorithm provides remarkable results when class-level metrics are applied. This study showed that the most crucial component in software fault prediction is the algorithm and not the metrics suite.

Acknowledgements

This research project is supported by The Scientific and Technological Research Council of TURKEY (TUBITAK) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor. We thank to Elda Dedja for her constructive suggestions.

References

[1] M. Bereta, T. Burczynski, Immune K-means and negative selection algorithms for data analysis, Information Sciences, Available online 14 November 2008.
[2] J. Brownlee, Artificial immune recognition system: a review and analysis, Technical Report 1-02, Swinburne University of Technology, 2005.
[3] J. Brownlee, Clonal selection theory & CLONALG. The clonal selection classification algorithm, Technical Report 2-02, Swinburne University of Technology, 2005.

[4] J. Brownlee, Immunos 81-the misunderstood artificial immune system, Technical Report 3-01, Swinburne University of Technology, 2005.
[5] J.H. Carter, The immune system as a model for pattern recognition and classification, Journal of American Medical Informatics Association 7 (1) (2000) 28–41.
[6] C. Catal, B. Diri, A fault prediction model with limited fault data to improve test process, in: Proceedings of the Ninth International Conference on Product Focused Software Process Improvement, Lecture Notes in Computer Science, Springer-Verlag, Rome, Italy, 2008, pp. 244–257.
[7] C. Catal, B. Diri, Software defect prediction using artificial immune recognition system, Proceedings of the Fourth IASTED International Conference on Software Engineering, IASTED, Innsbruck, Austria, 2007, pp. 285–290.
[8] C. Catal, B. Diri, Software fault prediction with object-oriented metrics based artificial immune recognition system, in: Proceedings of the 8th International Conference on Product Focused Software Process Improvement, Lecture Notes in Computer Science, Springer-Verlag, Riga, Latvia, 2007, pp. 300-314.
[9] C. Catal, B. Diri, A conceptual framework to integrate fault prediction sub-process for software product lines, in: Proceedings of the Second IEEE International Symposium on Theoretical Aspects of Software Engineering, IEEE Computer Society, Nanjing, China, 2008.
[10] L.N. De Castro, F.J. Von Zubben, The clonal selection algorithm with engineering applications, in: Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, 2000, pp. 36–37.
[11] Y.S. Ding, Z.H. Hu, H.B. Sun, An antibody network inspired evolutionary framework for distributed object computing, Information Sciences 178 (24) (2008) 4619–4631.
[12] K. El Emam, S. Benlarbi, N. Goel, S. Rai, Comparing case-based reasoning classifiers for predicting high risk software components, Journal of Systems and Software 55 (3) (2001) 301–320.
[13] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, Journal of Systems and Software 81 (5) (2008) 649–660.
[14] M. Evett, T. Khoshgoftaar, P. Chien, E. Allen, GP-based software quality prediction, in: Proceedings of the Third Annual Genetic Programming Conference, San Francisco, CA, 1998, pp. 60–65.
[15] I. Gondra, Applying machine learning to software fault-proneness prediction, Journal of Systems and Software 81 (2) (2008) 186–195.
[16] L. Guo, B. Cukic, H. Singh, Predicting fault prone modules by the Dempster–Shafer belief networks, in: Proceedings of the 18th IEEE International Conference on Automated Software Engineering, IEEE Computer Society, Montreal, Canada, 2003, pp. 249–252.
[17] M. Halstead, Elements of Software Science, Elsevier, New York, 1977.
[18] <http://en.wikipedia.org/wiki/Random_forest> (retrieved 20-09-08).
[19] Q. Hu, D. Yu, J. Liu, C. Wu, Neighborhood rough set based heterogeneous feature subset selection, Information Sciences 178 (18) (2008) 3577–3594.
[20] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, Identification of defect-prone classes in telecommunication software systems using design metrics, Information Sciences 176 (24) (2006) 3711–3734.
[21] X. Jin, R. Bie, Random forest and PCA for self-organizing maps based automatic music genre discrimination, in: Conference on Data Mining, Las Vegas, Nevada, 2006, pp. 414–417.
[22] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, Object-oriented software fault prediction using neural networks, Information and Software Technology 49 (5) (2007) 483–492.
[23] T.M. Khoshgoftaar, N. Seliya, N. Sundaresh, An empirical study of predicting software faults with case-based reasoning, Software Quality Journal 14 (2) (2006) 85–111.
[24] T.M. Khoshgoftaar, N. Seliya, Software quality classification modeling using the SPRINT decision tree algorithm, in: Proceedings of the Fourth IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, 2002, pp. 365–374.
[25] I. Koprinska, J. Poon, J. Clark, J. Chan, Learning to classify e-mail, Information Sciences 177 (10) (2007) 2167–2187.
[26] A.G. Koru, H. Liu, An investigation of the effect of module size on defect prediction using static measures, in: Workshop on Predictor Models in Software Engineering, St. Louis, Missouri, 2005, pp. 1–5.
[27] Z. Lei, L. Ren-hou, Designing of classifiers based on immune principles and fuzzy rules, Information Sciences 178 (7) (2008) 1836–1847.
[28] J. Liu, Q. Hu, D. Yu, A weighted rough set based method developed for class imbalance learning, Information Sciences 178 (4) (2008) 1235–1256.
[29] Y. Ma, L. Guo, B. Cukic, A Statistical framework for the prediction of fault-proneness, in: Advances in Machine Learning Application in Software Engineering, Idea Group Inc., 2006, pp. 237–265.
[30] T. McCabe, A complexity measure, IEEE Transactions on Software Engineering 2 (4) (1976) 308–320.
[31] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (1) (2007) 2–13.
[32] J.C. Munson, Software Engineering Measurement, Auerbach Publications, Boca Raton, FL, 2003.
[33] H.M. Olague, S. Gholston, S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, IEEE Transactions on Software Engineering 33 (6) (2007) 402–419.
[34] G.J. Pai, J.B. Dugan, Empirical analysis of software fault content and fault proneness using bayesian methods, IEEE Transactions on Software Engineering 33 (10) (2007) 675–686.
[35] S.T. Powers, J. He, A hybrid artificial immune system and self organising map for network intrusion detection, Information Sciences 178 (15) (2008) 3024–3042.
[36] T. Quah, Estimating software readiness using predictive models, Information Sciences, in press (Corrected Proof, Available online 14 October 2008).
[37] S.J. Sayyad, T.J. Menzies, The PROMISE Repository of Software Engineering Databases, University of Ottawa, Canada, 2005. <http://promise.site.uottawa.ca/SERepository>.
[38] N. Seliya, T.M. Khoshgoftaar, Software quality estimation with limited fault data: a semi-supervised learning perspective, Software Quality Journal 15 (3) (2007) 327–344.
[39] N. Seliya, T.M. Khoshgoftaar, S. Zhong, Semi-supervised learning for software quality estimation, in: Proceedings of the 16th International Conference on Tools with Artificial Intelligence, Boca Raton, FL, 2004, pp. 183-190.
[40] R. Tavakkoli-Moghaddam, A. Rahimi-Vahed, A.H. Mirzaei, A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness, Information Sciences 177 (22) (2007) 5072–5090.
[41] M.M. Thwin, T. Quah, Application of neural networks for software quality prediction using object-oriented metrics, in: Proceedings of the 19th International Conference on Software Maintenance, Amsterdam, The Netherlands, 2003, pp. 113–122.
[42] J. Timmis, M. Neal, Investigating the evolution and stability of a resource limited artificial immune systems, Genetic and Evolutionary Computation Conference, Nevada, 2000, pp. 40–41.
[43] J. Timmis, T. Knight, Artificial immune systems: using the immune system as inspiration for data mining, Data Mining: A Heuristic Approach, Idea Group, 2001.
[44] A. Watkins, AIRS: A resource limited artificial immune classifier, Master Thesis, Mississippi State University, 2001.
[45] A. Watkins, Exploiting immunological metaphors in the development of serial, parallel, and distributed learning algorithms, Ph.D. Thesis, Mississippi State University, 2005.
[46] A. Watkins, J. Timmis, Artificial Immune Recognition System: Revisions and Refinements, ICARIS 2002, University of Kent, Canterbury, 2002. pp. 173–181.
[47] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, 2000.

[48] X. Yuan, T.M. Khoshgoftaar, E.B. Allen, K. Ganesan, An application of fuzzy clustering to software quality prediction, in: Proceedings of the Third IEEE Symposium on Application-Specific Systems and Software Engineering Technology, IEEE Computer Society, Washington, DC, 2000, pp. 85.
[49] S. Zhong, T.M. Khoshgoftaar, N. Seliya, Unsupervised learning for expert-based software quality estimation, in: Proceedings of the Eighth International Symposium on High Assurance Systems Engineering, IEEE Computer Society, Tampa, FL, 2004, pp. 149–155.
[50] M.J. Zolghadri, E.G. Mansoori, Weighting fuzzy classification rules using receiver operating characteristics (ROC) analysis, Information Sciences 177 (1) (2007) 2296–2307.