A Meta-Learning Framework for Algorithm Recommendation in Software Fault Prediction

Silvia N. das Dôres, Luciano Alves, Duncan D. Ruiz, Rodrigo C. Barros
Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Av. Ipiranga, 6681, 90619-900, Porto Alegre-RS, Brazil
rodrigo.barros@pucrs.br

ABSTRACT

Software fault prediction is a significant part of software quality assurance and it is commonly used to detect faulty software modules based on software measurement data. Several machine learning based approaches have been proposed for generating predictive models from collected data, although none has become standard given the specificities of each software project. Hence, we believe that recommending the best algorithm for each project is much more important and useful than developing a single algorithm for being used in any project. For achieving that goal, we propose in this paper a novel framework for recommending machine learning algorithms that is capable of automatically identifying the most suitable algorithm according to the software project that is being considered. Our solution, namely SFP-MLF, makes use of the meta-learning paradigm in order to learn the best learner for a particular project. Results show that the SFP-MLF framework provides both the best single algorithm recommendation and also the best ranking recommendation for the software fault prediction problem.

CCS Concepts

•**Software and its engineering** → **Software defect analysis;**

Keywords

Algorithm Recommendation; Machine Learning; Meta-Learning; Software Quality; Software Fault Prediction

1. INTRODUCTION

A fault is a structural imperfection in a software system that may lead to the overall failure of the system. Within the software development process, fault prediction is a very important activity in order to improve software quality and reduce the maintenance effort prior to the deployment of the software [15].

Several techniques for software fault prediction have been proposed over the past 30 years. Machine learning approaches, in particular, have been increasingly used considered the results they provide with little or no extra cost for the companies. Among the most commonly used machine learning algorithms for software fault prediction are the Artificial Neural Networks, Decision Trees, Bayesian Classifiers, Ensemble Learners, and SVMs [15].

Even though there exist extensive experimental analyses over several distinct machine learning algorithms, there is no consensus in the literature regarding the superiority of one method over another, since the results are not consistent across different studies [14, 18, 19]. Indeed, the well-known no-free-lunch theorem states that no machine learning algorithm is best for all problems, and it seems to be precisely the case of the existing software fault prediction studies. Hence, none of the existing techniques seem to be capable of handling all different existing project data, which may vary according to the specificities of the software development organizations and their processes.

Since no algorithm is best for all projects, we propose a change of paradigm and instead of developing more robust methods that may not generalize for every software project, we introduce in this paper a novel framework that is capable of recommending the most suitable algorithm according to the project data at hand. Our novel framework, namely SFP-MLF (*Software Fault Prediction – A Meta-Learning Framework*), makes use of the meta-learning approach for the recommendation of machine learning algorithms that takes into account the characteristics of the particular organizational data. Our proposed solution can be seen as a context-aware algorithmic recommendation platform, which may recommend a single algorithm to be used or a list (ranking) of algorithms.

The remainder of the paper is organized as follows. Section 2 provides a background on the current studies that relate machine learning and fault prediction. Section 3 describes our proposed framework for algorithm recommendation in software fault prediction. Section 4 presents the empirical analysis that was conducted in order to evaluate the performance of the proposed framework. We present our

SAC 2016,April 04-08, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2851613.2851788

conclusions and future work directions in Section 5.

2. MACHINE LEARNING FOR SOFTWARE FAULT PREDICTION

Studies that report the use of machine learning techniques for improving software fault prediction usually focus on classifying software components (modules/classes) according to their level of defect-proneness. Typically, such a problem is abstracted into a two-class learning task, which identifies each module as *defect-prone* or *not-defect-prone*, according to the current project's metrics and also to fault data from similar projects. The model that is built by a machine learning algorithm can be subsequently applied to classify novel software components in a real-time fashion. The knowledge of possible weaker components will help testers in focusing their available resources on fixing the modules that are more prone to faults [15].

In this context, several machine learning algorithms have been used to inductively find patterns within the data to classify the software modules, including statistical procedures [13], tree-based methods [12], analogy-based strategies [6], and artificial neural networks [11], just to name a few.

Due to the variety of machine learning algorithms that have been applied to address the fault prediction problem, a novel class of studies emerged proposing frameworks for properly comparing the performance of these algorithms in order to find the most suitable solution for the problem. Their results are our main motivation, since they reported the nonexistence of a global solution that is suitable for distinct projects. Next, we present the most important studies that addressed the problem of software fault prediction through machine learning approaches.

2.1 Related Work

Menzies et al. [18] argued on the use of static code attributes to be used as input to learning algorithms. The authors proposed a baseline experiment using public-domain datasets and they compared the performance of different learning algorithms. Their experimental results showed that the best features to be used for defect prediction vary from dataset to dataset, and so does the algorithms' performance. Their main conclusion was that it is no longer correct to assess defect-learning methods using only one dataset and one learner, and that the choice of learning method is far more important than which subset of the available data is used for learning.

Lessmann et al. [14] presented a framework for performing an extensive comparison among machine learning algorithms for software defect prediction. They conducted a large-scale analysis of 22 different classification models over 10 public-domain datasets. Their experimental results indicated that no significant differences in performance could be detected regarding their experiments with the 22 distinct learning algorithms, since distinct algorithms were the best choice for distinct datasets.

Finally, Song et al. [19] proposed a general framework that combines distinct learning algorithms and evaluation schemes. In the evaluation stage, different learning schemes (a set that comprises a data preprocessor, a feature selection procedure and a learning algorithm) were evaluated in a part of the historical data to determine whether a certain learning scheme performs sufficiently well for prediction purposes. In the defect prediction stage, the best learning scheme was used to build a predictive model over all historical data, and the resulting model is then used to predict faults in new data. The authors concluded that no learning scheme dominates, and thus a solution would be to select different schemes according with the dataset characteristics. This is also commented by Malhotra [15] in her systematic review that compares the performance of several machine learning algorithms for 23 different datasets.

The fact that no algorithm outperforms all others for every software fault dataset is easily explained due to the no-free-lunch (NFL) theorem, according to which any advantage presented by an algorithm on a specific class of problems is mitigated when it is applied to another class of problems. Hence, if all the problems are equally probable, then the algorithms tend to present, on average, the same predictive behavior, preventing the existence of a universally optimal method [2]. Therefore, defining the algorithm to be used to obtain the most satisfactory results for a given application must be made in a context-aware application-driven manner.

To address the above-mentioned challenge, we propose a novel framework to predict faults within the software development scenario, focusing on recommending the most suitable machine learning algorithm for a given project. This approach is based on the meta-learning paradigm, which extracts a particular set of features for each given dataset and uses this so-called meta-data to point to the most suitable learning algorithm. We present our approach in detail in the next section.

3. META-LEARNING FRAMEWORK FOR SOFTWARE FAULT PREDICTION

Meta-learning is a relatively new area within machine learning that studies how learning systems can increase in efficiency through experience. Its main goal is to understand how learning itself can become flexible according to the domain or task under study. One can see meta-learning as a tool that helps determining under what conditions a given algorithm is more suitable to be applied instead of others, naturally allowing for the development of an approach for algorithm recommendation [7].

With that in mind, in this paper we propose *Software Fault Prediction – A Meta-Learning Framework* (SFP-MLF), a meta-learning based framework for recommending machine learning algorithms for software fault prediction. Figure 1 presents the rationale of the framework, whose process is based on the general meta-learning approach proposed by Kalousis [10] and widely used in algorithm recommendation approaches [7, 16]. In the following sections, each step of Figure 1 is explained in greater detail.

3.1 Defining the Input Datasets

For a meta-learning system to be operational, we need a

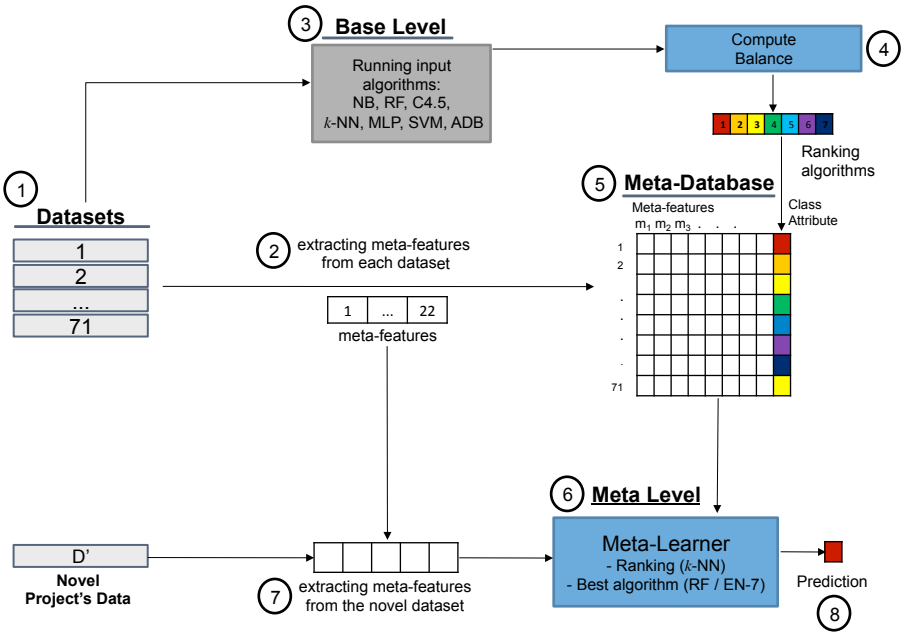

Figure 1: Overview of SFP-MLF.

representative set of datasets (Figure 1 - Item 1) from a given domain or set of domains. For the particular case of software fault prediction, several public datasets have been used by researchers in the past few years [15], such as the well-known NASA datasets.

A set of 71 datasets was used as input to SFP-MLF, all of them from the PROMISE public software engineering repository [17]. The PROMISE repository contains the NASA datasets, which are by far the most employed data in software fault prediction studies. Notwithstanding, there are only 13 datasets from NASA, and for a meta-learning system to be successful one needs to collect a much larger amount of datasets (as a thumb rule, more than 50 datasets are desired for a significant meta-learning analysis).

The set of input datasets is the original source of meta-knowledge of a meta-learning system. By learning the characteristics of these datasets, SFP-MLF will be capable of recommending suitable learning algorithms for unseen data (data from other projects).

3.2 Extracting Meta-Features

After collecting a representative set of datasets, we need to define which *meta-features* will be extracted from these data (Figure 1 - Item 2). The meta-features are abstractions from the set of input datasets, indicating in a meta-level the characteristics of each dataset and how hard it is to solve the classification problem that each one represents.

We decided to use two sets of meta-features that are well-known in the meta-learning area:

1) Meta-features from the STATLOG project: originally proposed in the STATLOG project [3], this set consists of 8 meta-features that are based on simple statistics collected from the datasets, such as the number of classes, number of features, correlation among features, etc.

2) Meta-features based on data complexity measures: originally proposed in [1, 9], these meta-features represent the complexity of a classification problem considering aspects such as the overlap in the feature values, the separability of the classes, and geometric or topological properties.

Table 1 presents a summary of all meta-features that are collected by SFP-MLF.

Table 1: Meta-features used in SFP-MLF.

Type	Name	Meta-feature
[3]	#Sam	Number of samples (instances)
[3]	#Clas	Number of classes
[3]	#Att	Number of attributes (features)
[3]	#BinAtt	Number of binary attributes
[3]	MeanSk	Mean skewness of attributes
[3]	MeanKur	Mean kurtosis of attributes
[3]	MeanCor	Mean multiple attribute correlation
[3]	SDRatio	Standard deviation ratio
[1]	F1	Max Fisher's discriminant ratio
[1]	F1v	Max Fisher's discriminant ratio (dir. vector)
[1]	F2	Overlapping of the per-class bounding boxes
[1]	F3	Maximum individual feature efficiency
[1]	F4	Collective feature efficiency
[1]	L1	Minimum sum of the error of a linear classifier
[1]	L2	Training error of a linear classifier
[1]	L3	Non-linearity of a linear classifier
[1]	N1	Fraction of points lying on the class boundary
[1]	N2	Average intra/inter class NN distances
[1]	N3	Leave-one-out error rate of the 1-NN
[1]	N4	Non-linearity of the 1-NN
[1]	T1	Fraction of max. covering spheres on data
[1]	T2	Collective feature efficiency

3.3 Defining the Input Algorithms

After defining the input set of datasets, one should also define a set of algorithms as input for the meta-learning system (Figure 1 - Item 3). The recommendation will be made regarding this set of algorithms previously defined as input to the system. Afterwards, each algorithm is executed over each dataset from the input set, and several metrics regarding its performance are recorded, such as predictive performance information (e.g., accuracy) and time complexity measures (e.g., time spent for running the algorithm over the given dataset).

For defining this input set of algorithms, we turned to the most referenced machine learning algorithms in the software fault prediction context. We based our choice on a recent systematic review performed by Malhotra [15], so we ended up selecting the 7 most prominent learning algorithms in the area, namely: 1) Naïve Bayes (NB); 2) Random Forests (RF); 3) C4.5; 4) k-Nearest Neighbors (k-NN); 5) Support Vector Machines (SVMs); 6) Multi-layer Perceptron (MLP); and 7) Ada Boost (ADB).

We executed each of the 7 algorithms over each of the 71 input datasets, resulting in a list of collected metrics regarding the performance of each algorithm in each dataset. We made use of the implementations available at the WEKA machine learning toolkit [8] for these 7 algorithms. Each of them has a large set of input parameters, so we decided to use the parameters suggested by WEKA (the default parameters).

3.4 Defining the Performance Measure

After executing the 7 algorithms over the 71 input datasets, we need to define an evaluation measure of choice in order to assess the performance of the algorithms in a straightforward fashion (Figure 1 - Item 4).

There are several predictive performance measures for classification, and a list of pros and cons of each measure is out of the scope of this paper. For avoiding issues on the selection of a proper evaluation measure, we decided to employ the *balance* criterion, which was first presented by Menzies et al. [18] and is commonly found in many software fault prediction studies. Balance is the Euclidean distance between the generated prediction and the ideal balance between true positives rate (TPR) and false positives rate (FPR). This distance is then normalized by the maximum distance across the ROC square ($\sqrt{2}$), so its value falls between $[0, 1]$. Balance also subtracts the normalized distance by 1, indicating that higher balance values (closer to 1) are preferred. Hence, for each of the 7 input algorithms we collect its corresponding balance values for each of the input datasets.

3.5 Building the Meta-Database

The next step of SFP-MLF is to build a meta-database (Figure 1 - Item 5), which embodies the knowledge of the relationship between meta-features and algorithmic predictive performance. In this meta-database, each instance (table row) corresponds to one of the 71 input datasets, and each predictive feature (attribute, table column) corresponds to the computation of the meta-features described in Section 3.2. Finally, there is a final piece of information in this meta-database, which indicates which algorithm obtained the best performance (as measured by the balance criterion) in the corresponding dataset. Another option is to include the complete ranking of the performance achieved by the input algorithms as this final piece of information in the meta-database. This choice should be made according to the final goal of the system: if one wants do predict which single algorithm is more suitable for a given dataset, then we need to store only which was the best algorithm for each input dataset; otherwise, if one wants a list of algorithms ranked according to their performance, then we need to store the ranking of the algorithms in the meta-database.

3.6 Choosing a Meta-Learner to Build a Recommendation Model

When we have data from a novel project where we would like to apply a machine learning algorithm to predict fault-prone modules, we need to extract the same meta-features that we extracted from the input datasets. Afterwards, we need to choose a meta-learner (machine learning algorithm) that will be applied to the meta-database in order to generate a predictive model that is capable of recommending algorithms for the software fault prediction problem (Figure 1 - Item 6). Once the model is built, it is applied over the meta-features that were extracted from the novel project's dataset (Figure 1 - Item 7), generating as output the most suitable algorithm to be used for that particular dataset (Figure 1 - Item 8).

The model generated by the meta-learner can be used with two different purposes [10]: i) ranking the list of available machine learning algorithms for the novel project's dataset; ii) recommending the best algorithm for the novel project's dataset. The meta-learner that SFP-MLF employs depends on the purpose desired by the final user. For the task of ranking algorithms, SFP-MLF makes use of the k-NN (k-nearest neighbors) algorithm, which naturally allows the prediction of a ranking list. For the task of recommending the most suitable algorithm, SFP-MLF allows the use of either the Random Forests (RF) algorithm [5] or of an ensemble of the 7 input algorithms (EN-7). RF was chosen considering it is said to be one of the most effective algorithms for software fault prediction [15]. The second option, namely EN-7, concerns the use of a majority vote scheme of the predictions made by the 7 algorithms that were chosen as input to SFP-MLF.

4. EXPERIMENTAL ANALYSIS

We employ a typical leave-one-out cross-validation procedure (LOO-CV) to evaluate the quality of the recommendation made by SFP-MLF. LOO-CV iteratively separates one dataset to be used as a the test set (novel project's dataset whose data is *a priori* unknown to SFP-MLF), and the remaining 70 to be used as the training set (datasets to be used as input to build the meta-database). This process is repeated until every dataset has been used once as test set.

In Section 4.1, we describe how SFP-MLF was applied

to predict the ranking of input algorithms for each novel project's dataset. In Section 4.2, we present the results achieved by SFP-MLF when recommending a single most suitable algorithm to be applied for each novel project's dataset that is being considered.

4.1 Ranking Recommendation

A meta-learning system can be used to recommend a ranking of algorithms, i.e., a list of algorithms whose order is dictated according to their predicted performance for the dataset of interest. For predicting rankings, we employ the k-NN algorithm with $k = 1$, which means the ranking to be predicted for a given project's dataset will match with the ideal ranking of its nearest neighbor (i.e., the ranking of the dataset it is most similar to).

An ideal ranking should represent the exact ranking order of the machine learning algorithms for the unseen data [4], but this scenario is unlikely to happen in the real world. Therefore, with the purpose of validating whether SFP-MLF is capable of recommending a "good" ranking, we compare real and predicted rankings according to the well-known Spearman's correlation coefficient.

For assessing the quality of SFP-MLF's ranking prediction (henceforth called $SFP\text{-}MLF\text{-}R$), we compare it to two distinct baselines: i) random ranking; and ii) majority ranking. In random ranking (RR), we generated 30 random rankings for each dataset: each input algorithm was assigned a random number without replacement from the integer interval $[1, 7]$, since there is a total of 7 input algorithms. We computed the Spearman's coefficient for the 30 random rankings per test dataset, and then averaged them to produce a single Spearman's coefficient value per test dataset. In majority ranking ($MAJ\text{-}R$), we simply select the most frequent ranking from the 70 datasets in the training set. The most frequent ranking (majority ranking) is used as the predicted ranking for all test datasets.

4.1.1 Ranking Results

Table 2 presents the results of the ranking evaluation analysis. Results show the average Spearman's coefficient value for the 71 datasets (each computed within the LOO-CV procedure). Note that the average value of Spearman's coefficient for $SFP\text{-}MLF\text{-}R$ (0.327) is considerably larger than those of the baseline strategies RR (0.016) and $MAJ\text{-}R$ (0.205). As expected, the random ranking strategy generates an average result close to 0. Results provided by $MAJ\text{-}R$, in turn, show that employing always the same algorithm for every particular dataset does not provide results as accurate as recommending different algorithms for different datasets.

Table 2: Average μ and standard-deviation σ values of the Spearman's coefficient for $SFP\text{-}MLF\text{-}R$, $MAJ\text{-}R$, and RR.

	$SFP\text{-}MLF\text{-}R$	$MAJ\text{-}R$	RR
μ	**0.327**	0.205	0.016
σ	0.478	0.512	0.086

4.2 Single-Algorithm Recommendation

The other strategy employed by SFP-MLF is to recommend a single algorithm as the most suitable choice for the project's data at hand. In this strategy, we employ either RF or EN-7 as the meta-learner to generate a predictive model from the meta-database. From here on, we refer to these strategies as $SFP\text{-}MLF\text{-}RF$ and $SFP\text{-}MLF\text{-}EN$-7.

In order to assess the performance of $SFP\text{-}MLF\text{-}RF$ and $SFP\text{-}MLF\text{-}EN$-7, we compare the predictive performance of the algorithm recommended by them with the predictive performance of each one of the 7 algorithms that are used as input to SFP-MLF.

Table 3 shows the results obtained with $SFP\text{-}MLF\text{-}RF$. The values presented in Table 3 refer to the average rank achieved by each algorithm in the 71 datasets. For instance, suppose that $SFP\text{-}MLF\text{-}RF$ recommends the Naïve Bayes algorithm to be used for the PC1 NASA dataset. Hence, we verify the real performance of Naïve Bayes in PC1 in terms of the balance criterion (recall that we have executed all input algorithms over all input datasets), and next we check how the corresponding balance value stands when compared with the balance achieved by the 7 algorithms (including Naïve Bayes itself). After that analysis is made, we rank the performance of each one of the 8 algorithms (the algorithm recommended by $SFP\text{-}MLF\text{-}RF$ alongside the 7 input algorithms). For each input dataset, there is a ranking of the 8 algorithms, and hence we compute the average rank of the 8 algorithms for the 71 datasets. The lower the value of the average rank, the better the algorithm. The best balance average rank that SFP-MLF could possibly achieve is 1.5, which would mean that SFP-MLF always recommends the best-performing algorithm for all 71 datasets.

Table 3: Average rank (μ) and standard-deviation (σ) of the balance criterion for the 71 input datasets. Here, SFP-MLF's version is the one that selects the best algorithm with Random Forests.

Algorithm	μ	σ
$SFP\text{-}MLF\text{-}RF$	**2.472**	1.501
NB	3.972	2.650
RF	3.993	1.902
C4.5	4.225	1.855
k-NN	4.775	1.989
SVM	7.028	1.680
MLP	4.134	1.561
ADB	5.331	2.054

We can see that $SFP\text{-}MLF\text{-}RF$ outperforms all the remaining algorithms in terms of balance values, with an average rank of 2.472. These results clearly suggest that our proposed approach for recommending algorithms achieves best results than using the same algorithm for every software fault prediction dataset.

Table 4 shows the results obtained with $SFP\text{-}MLF\text{-}EN$-7. Note that changing the meta-learner in SFP-MLF has little impact on the final result, since $SFP\text{-}MLF\text{-}EN$-7 also outperforms all remaining algorithms with a balance average rank of 2.556. Whereas most papers in the software fault prediction context suggest using either Naïve Bayes or Random Forests for predicting software module fault-proneness, we can see that a customized approach

capable of recommending distinct algorithms according to the dataset at hand indeed achieves enhanced performance, proving our initial hypothesis that research in the area should concentrate efforts on recommending algorithms instead of developing more robust algorithms to use in all datasets.

Table 4: Average rank (μ) and standard-deviation (σ) of the balance criterion for the 71 input datasets. Here, SFP-MLF's version is the one that selects the best algorithm with the ensemble of the 7 input algorithms.

Algorithm	μ	σ
SFP-MLF-EN-7	**2.556**	1.519
NB	3.958	2.647
RF	3.944	1.920
C4.5	4.239	1.850
k-NN	4.775	2.007
SVM	7.021	1.704
MLP	4.127	1.574
ADB	5.310	2.068

5. CONCLUSIONS

In this paper, we proposed a framework called SFP-MLF (*Software Fault Prediction: A Meta-Learning Framework*). It analyzes the performance of machine learning algorithms on a collection of datasets and develops a meta-database that allows the recommendation of the most suited machine learning algorithm for novel data. Experiments were conducted taking into account 71 public software fault prediction datasets from the PROMISE repository [17], and results show that SFP-MLF is capable of recommending the best algorithm for each dataset used as test set, outperforming the selection of a single algorithm for all datasets. Our findings seem to be an important landmark in the area, since they suggest that research efforts should be directed towards improving algorithm recommendation instead of building more robust approaches, the latter being the case of all studies so far.

As future work, we plan to enlarge the list of input algorithms, performing recommendation with the list of 22 machine learning algorithms that were experimented in the work of Lessmann et al. [14].

Acknowledgments

We acknowledge the Brazilian research agency CAPES for funding this research.

6. REFERENCES

[1] M. Basu and T. K. Ho. *Data Complexity in Pattern Recognition*. Springer London, www.springer.com, 2006.

[2] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer, Berlin, 2009 edition edition, Nov. 2008.

[3] P. B. Brazdil and R. J. Henery. Machine learning, neural and statistical classification. pages 175–212. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[4] P. B. Brazdil, C. Soares, and J. P. d. Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, Mar. 2003.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai. Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software*, 55(3):301–320, 2001.

[7] B. Feres de Souza, C. Soares, and A. C. de Carvalho. Meta-learning approach to gene expression data classification. *International Journal of Intelligent Computing and Cybernetics*, 2(2):285–303, 2009.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[9] T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):356–370, Mrço 2002.

[10] A. Kalousis. *Algorithm Selection via Meta-Learning*. PhD thesis, Geneva, 2002.

[11] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai. Object-oriented software fault prediction using neural networks. *Information and software technology*, 49(5):483–492, 2007.

[12] T. A. M. Khoshgoftaar and N. Seliya. Tree-based software quality estimation models for fault prediction. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 203–214. IEEE, 2002.

[13] T. M. Khoshgoftaar and E. B. Allen. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(04):303–317, 1999.

[14] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, July 2008.

[15] R. Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27(Complete):504–518, 2015.

[16] R. G. Mantovani, A. L. Rossiy, J. Vanchorenz, B. Bischlx, and A. C. De Carvalho. To tune or not to tune: recommending when to adjust SVM hyper-parameters via Meta-learning. IEEE, 2015.

[17] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

[18] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, Jan. 2007.

[19] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, 37(3):356–370, Maio–Junho 2011.