Impact of Parameter Tuning and Ensemble Methods on Fault Prediction Models

Guru Darshan Pollepalli Manohara
North Carolina State University,
Raleigh, NC
USA
gpollep@ncsu.edu

Abstract

Fault prediction models provide indispensable information to organizations and help them to redirect their efforts and resources to solving the issues in fault prone modules. This paper briefly surveys the state of the art algorithms for Fault prediction and provides suggestions to bridge the gaps in the algorithms to improve the reliability of the results. Based on results published by Gohtra et al [17], this paper claims that the predictive performance can be further improved by the application of ensemble methods and parameter tuning.

CCS Concepts •**Machine Learning** → Ensemble methods; Parameter tuning; Classification •**Software Engineering** → Fault prediction

Keywords Fault Prediction, Ensemble Methods, Parameter Tuning

1 Introduction

Ability for an organization to produce quality software products is highly correlated with their Quality Assurance methodologies. An important aspect of Software Quality team is to predict the chances of existence of a bug based on heuristics and past knowledge. Software giants have shown increasingly more interest in creating models capable of encapsulating the previously known knowledge about the software in terms of predefined metrics and creating data models which can be used to predict the occurrence of faults in their future releases. Identifying faulty modules helps the QA team to redirect their efforts and resources. On the research front, majority of the studies performed in the field of Software Engineering encompass fault prediction and project planning with more number of papers published in the field of fault prediction. Menzies et al [9] suggest that the probability associated with the identification of a bug using fault prediction models is relatively high (71%) when compared to traditional code review methodologies which result in a probability of around 60%. Code reviews increase the chances of human error rates as the code will have to be examined by developers line by line for errors. Hence using fault prediction models have significant advantage in terms of speed and accuracy when compared to traditional methods of code review.

Metrics associated with the software must be computed before the classification algorithms are executed. Chidamber and Kemerer [5] proposed a suite of metrics for object-oriented paradigms which can be used to evaluate the fault proneness of a class. Computing the CK metrics would involve examining the class structures in the code and inspecting bug repositories like Bugzilla or GitHub to extract the classes responsible for the bugs. Tibor et al [6] explored Bugzilla repository to obtain bug reports of projects associated with Mozilla and manually examined nearly 3000 bug reports to extract the classes responsible for the faults. Many well-known open-source projects have published the CK metrics associated with their code online to be used by data scientists to train their models. This project uses PROMISE data repository and Seacraft repository to obtain CK metrics for many open-source projects like log4j, ant, velocity, etc. Evaluation of CK metrics for a class can be used to proactively suggest the developer to avoid designs capable of causing faults in the code [8]. Danijel et al [12] analyzed different software metrics used in 106 papers published between 1991 and 2011. They concluded that nearly 49% of the metrics used were Object-oriented metrics. CK metrics were predominantly seen in most of the research papers. But Cagatay et al [13] suggest that the use of class level metrics for early detection of faults during design phase. Koru et al [15] showed that use of class level data yielded better prediction performance than method-level data.

A plethora of modelling techniques are available in a data scientist's tool box to capture, analyze and predict different faults in a software system. Lately, Random forests, logistic regressions and their variants are garnering attention for their simplicity of implementation and their predictive power. Papers published prior to Chidamber and Kemerer [5] used method based metrics to estimate the faults in a system. Porter et al [11] used an empirical approach involving classification trees to identify high risk classes in NASA and Hughes project data. An average of 79.3% accuracy was obtained using classification trees. They mention fine-tuning of parameters as a refinement in their paper. Tibor et al [6] used statistical methods (logistic and linear regression) and machine learning (decision tree and neural networks) to predict the number of bugs associated with a class.

Menzeis et al [4] suggest that tuning is an under-explored optimization problem in the field of data learning algorithms. This project aims to extend the ideas presented in the paper to other learning algorithms and combine them with ensemble methods to perform a comparative analysis on the results obtained.

Paper Organization

The remainder of the paper is organized as follows. Section 2 describes the importance of individual criteria. Section 3 describes the key criteria in detail by examining excerpts of published papers in the related fields. Section 4 surveys and critically evaluates research paper. Section 5 explains the methodologies used to bridge the gaps. Section 6 provides details about the project and poses research questions about test scenarios to be examined.

2 Criteria

2.1 Readability

Readability of a data model is highly instrumental in many disciplines of Software Engineering and its related focuses. Project management tasks heavily rely on the organization and transparency of the model generated by the algorithms to effectively allocate resources and make minor adjustments to the model to provide the necessary flexibility for accomplishing the organizational goals. Many project management softwares are equipped with capturing microscopic details about products which are then translated into usable data models where managers can transform the data inputs into strategic goals and perform informed decisions. Project managers are responsible for overseeing the project milestones and deliverables. They are also responsible to dynamically fine-tune the resources based on availability without having to request the complete recompilation of the data model. Data models which provide little to none readability can slow down this process significantly. In many open-source software development cycles, developers are available on a voluntary basis and collaboration from developers becomes an essential component of project management. The need to have a readable and interpretable model becomes a necessity than a cosmetic. In cases where a developer is unavailable, the manager can quickly consult the data model to reallocate the resources on demand. Human readable models like Decision Trees provide better insights into the tasks than models like Artificial Neural Networks. Graphical illustration provided by Decision Trees increase their ease of use and help mangers to analyze the models better. Readable models can also be used to methodologically scrutinize the various inter dependencies between projects and help them create varied levels of focus for individual projects.

2.2 Actionable Conclusions

It is imperative to know the differences between the classifications than the classification themselves. That is, we can benefit much more by knowing the difference (in attributes) which caused a data point to be classified into one of the categories (say X) and not another category (say Y). In bug prediction algorithms, contrasts between Fault and Fault free categories can be examined by building a contrast learner. For example, in a bug prediction algorithm using Chidamber & Kemerer metrics [5], one can look at the classes which have bugs and say that a high value in CBO (Coupling Between Object classes - which can be qualitatively defined as how much a method in a class is dependent on methods or variables in another class) tends to increase the probability for a class to be bug prone [6]. In case of project management, one can assess if the number of hours spent by a developer on a task has a direct impact on his code being bug free or bug prone. Contrast learners can be programmed further to recognize the importance of the classification categories and generate contrasts with respect to only those categories. This can highly reduce computational costs associated with the learner as less data is examined to provide a conclusion.

2.3 Learnability and Repeatability of the Results

Algorithms like K-Means perform well in a plethora of situations but have the disadvantage that the entire dataset must be stored in the memory for every iteration of the algorithm (to compute centroids) [24]. [24] suggests the existence of many strategies which can be used to reduce the computational complexity while computing the centroids using aggregation of the training data into a statistic. Mini-batch K-Means uses an alternate approach which uses random set of fixed batches of samples to compute the centroid and a new set of samples is used every time. This way the space complexity is reduced and the clusters begin to converge at a rate which is inversely proportional to the number of new data added to a cluster [24]. Such optimizations to algorithms prove to be highly important despite their tradeoffs and are highly effective for executing computationally complex algorithms with minimal compromises. The small memory foot print of Mini Batch K-Means and Naïve Bayes provide very attractive solutions where the computational resources are minimum (like embedded systems). Xbox Kinect uses a highly modified version of random forests to compute the location of objects using its IR camera. Only 10% of the overall computational bandwidth is available for visual recognition, so the system compromises on accuracy for faster, real-time processing of tracking data.

2.4 Multi-goal reasoning

Virtually every real-world problem is associated with multiple goals. Many decision-making problems in the business world are formulated as multi-goal reasoning problems. In Economics and Finance world, the determination of policies and portfolios involve attaining a delicate balance between multiple (often conflicting) goals and the variant of multi-goal reasoning algorithms used play a very important role in determining the outcomes associated with the problem. Multiple iterations of the model are simulated to rigorously test and evaluate the performance of the model. Given that more than one solution is possible in multi-goal reasoning problems, the user is required to find tradeoff between the solutions and pick the solution best suited for the use case. Repeated application of the algorithm is required to obtain the multiple solutions. One of the methods of obtaining solutions for multiple goal reasoning is by converting multiple goal measures into single goal [3] (such as denomination scores) and then use the algorithms designed for single goal reasoning to obtain solutions for the multi-goal problem at hand. Evolutionary algorithms can be used in place which provide many unique advantages including efficient computation and better convergence of solutions [3]. These algorithms use Selection (retaining parents from one generation to next), Cross-over and Mutation to create new candidates in the population which can replace their parents (if they have higher scores). Multiple iterations of the algorithm are performed creating new generations and (possibly) new candidates in the frontier. Eventually the algorithm converges when the best fitness score of the frontier does not change or the maximum number of generations have reached.

2.5 Anomaly Detection

Anomaly detection can be defined as a set of paradigms capable of detecting any uncommon data which do not conform to the expected behaviour of the data model (outliers). Anomaly detection has several applications in medical sciences like detection of cancer, credit card theft can be detected if there is a huge change the expenditures and very often it has been used to detect the health of component machine of a control system by continuously observing its internal parameters. Many safeguards have been designed in response to anomaly detection like blocking a credit card till the user can verify the exorbitant purchase or alerting the operator before a control system fails. Contextual anomaly detection can be used in case the anomaly detected must be examined in its local environment like a sudden increase in network bandwidth usage during sports broadcasts should not be cause any flags to be raised even though the bandwidth usage is above the normal limits.

2.6 Incremental Learning

In today's world, the database sizes increase at an alarming rate. Many companies are heavily investing in technologies capable of scaling their data learning algorithms to adapt to massive amounts of incoming training data [1]. Extensive research is done to find frameworks for existing algorithms to augment the ability to learn incrementally and provide continuous availability of data. Reconstruction of Data models upon receipt of new training data can be done either by building the model from scratch or appending the new data to the data model without sacrificing the existing knowledge base. The former method becomes computationally expensive when there are massive amounts of data involved (like Big Data). Hence the data model must be incrementally trained to update the knowledge base over time to cater to the real-time requirements of the task at hand. Alternatives include periodic retraining using batches of data. A crude comparison can be made to biological intelligent systems where the information is learnt over their lifetime to accumulate behaviors and develop associations to construct goal oriented behavior [2]. There are situations where the data used to train the data model may not be available for retraining the model during the next attempt. In such cases, incremental learning comes to the rescue.

2.7 Shareable

Data privacy is one of the very key factors to be considered while using data mining algorithms. The data used to train the data can be private to an organization but the data will be requested by a third party for creating data models. Essential protocols will have to be developed for sharing the data and must capture rules governing how the data is shared and how much of the data is shared. The protocol must attain a delicate balance between the reduction of dispensable data and concentrate on the vital data points which adequately describe the data model. Data obfuscation techniques will have to be used to mask the actual data but still maintaining the required parts to construct the data model. Increasing size of databases and data sets pose a further problem of how much data must be shared to create an efficient data model.

2.8 Context Aware

A gamut of today's state-of-the art data learning models visualize the data as an aggregated information and pay little to no heed to the sub-portions of the data which were obtained under different environments (contexts). Viewing the data set as being constituted of homogenous populations and deriving conclusions based on this assumption can often lead to critical errors in judgements causing depletion of accuracy of the data model. As an example, consider the following statement "Exercise and athletics can reduce the risk of diabetes". The statement generalizes by saying that exercise and athletics is always good. This might be true in case of a general population but the statement must be re-evaluated for sub-populations which contains patients recently treated for heart attack or people who had recently undergone surgery. The differences these sub-populations exhibit can be encoded into a context which specifies the type of population under observation. Even though these sub-populations form a smaller part of the overall populations, it is imperative that the data models are aware of the contexts and accordingly tune the control parameters of the data model or generate different data models based on the contexts. Since the exact number of context is unknown beforehand, it is hard to determine the partitions to capture all the contexts. Algorithms like Mini Batch K-Means can be used to cluster the data and then apply the algorithm on each individual cluster to create different data models aware of the local environments. Bettenburg et al [22] propose a new technique for bug prediction models using clustering algorithms where the training data is first clustered and a classification algorithm like Naïve Bayes is run on individual cluster so that the locality of a data point can be accounted while applying fault prediction algorithms.

2.9 Self-tuning

Attributes of a model like its accuracy, performance, etc. are highly reliant on the values of the control parameters chosen for the algorithm at the beginning of the data modelling. To obtain the right combination of parameters which provide the best score, the algorithm will be required to run with different parameter settings or run all possible combinations of the parameters which becomes computationally intensive [4] (but provides an exhaustive search of all the possible combinations which can be used to optimize the performance of the model). Evolutionary algorithms like Differential Evolution decrease the convergence time by orders of magnitude using mutation and cross-overs of candidate solutions. Tuning has a direct impact on the conclusion obtained using the learners. Well-tuned learners tend to provide better accuracy and precision compared to other learners and the results of many research papers can be directly challenged by testing with tuned data models [4].

3 Key Criteria

Fault prediction models tend to underperform when the learners used to generate the data models are configured with sub-optimal settings for the control parameters. Chakkrit et al [7] argue that the parameter settings can have a significant impact on the performance of the classification techniques used in fault prediction models. The generated model is highly reliant on the control parameters used at the time of model training. Number of decision trees in a random forest or the value of K in K nearest neighbor algorithm can yield significantly different models for different values of the parameter. In case of KNN, the accuracy of the data model is determined by the value of K chosen before the algorithm is started. For very low values of K, the algorithm tries to overfit to the training data and the effect of noise is predominantly amplified. For large values of K, the algorithm tries to underfit the model but irons out the noisy data points. Jiang et al [14] show that the default control parameter values of Random Forests and Naïve Bayes are often suboptimal.

The control parameters must be carefully chosen to obtain the best performance for the model. Koru et al [15] suggest that the parameters chosen can have huge impact on the accuracy of the model. At the same time, it is impossible to test for all the possible parameter settings. Kocaguneli et al [16] show that there are nearly 17,000 possible settings for training KNN algorithm [7]. Algorithms like Grid Search try to obtain the best control parameter setting by testing all possible combinations. As explained before, the process is computationally intensive and outweigh the benefits obtained from tuning the parameters. On the other hand, Menzies et al [4] show that evolutionary algorithms like Differential evolution converge to an optimal solution in roughly 100 iterations. The paper also demonstrates an increase in precision by 60% in some cases. It warns about the fallacies of using off-the shelf tuned

models by showing that the tuned parameters are highly dependent on the data sets used to compute the model. The best part is, it challenges many of the previously published papers by recreating their test environment and producing highly tuned models that can be used to contradict some of the results published in other papers. Hence tuning is a key criterion for learners as they can result in completely different outcomes when used correctly.
Very often data models fail to produce learners that can be used for actionable outcomes. In case of fault prediction algorithms, if the data model can tell the user what differentiates a piece of code as bug prone from being bug free, the developer can receive feedback from a static code analyzer that can proactively warn him/her to take the necessary steps to remove the cause of the bug (like decreasing the coupling or cohesion in a class). This can only be achieved by a contrasts learner. The simplicity of the algorithm is often ignored and the readability is traded-off for performance.

4 Critique

The paper published by Ghotra et al [17] has been considered as a reference model for comparison of several state-of-the-art fault prediction algorithms. This paper compares different sets of data learners to examine the performance by evaluating the Area Under the Curve (AUC). The authors used a varied set of statistical (Naïve Bayes, Simple Logistic) and machine learning models (like KNN, Logistic Model Tree, etc.) to examine the faults in NASA and PROMISE corpus. A comprehensive list of all the learners examined by the authors has been listed in Table 1.

Table 1. Classification Techniques examined by Ghotra et al [17]

Family	Technique	Abbreviation
Statistical Techniques	Naive Bayes	NB
	Simple Logistic	SL
Clustering Techniques	K-means	K-means
	Expectation Maximization	EM
Rule-Based Techniques	Repeated Incremental Pruning to Produce Error Reduction	Ripper
	Ripple Down Rules	Ridor
Neural Networks	Radial Basis Functions	RBFs
Nearest Neighbour	K-Nearest Neighbour	KNN
Support Vector Machines	Sequential Minimal Optimization	SMO
Decision Trees	J48	J48
	Logistic Model Tree using Logistic Regression	LMT

The paper begins by examining an earlier well-known paper published by Lessmann et al [10] where the experiment involved evaluating different learners with NASA corpus. The experiment investigated 22 different classification techniques. Lessmann et al [10] concluded with their experiment that 17 out of the 22 classification techniques demonstrated little to no statistical difference. Even the precursors of Gohtra et al [17] like Sheppard et al [19] showed that the classification technique used for fault prediction algorithms had no effect on the fault prediction problem.

Using the proof provided by Sheppard et al [19], the authors argue that the data sets used by Lessmann et al [10] from NASA corpus contained high bias and noise. And postulate that the results would have been different with filtered data from the NASA corpus. The data used by Lessmann et al [10] uses proprietary data which might have reduced the diversity in the type of data and thereby preventing the paper to make any generalizations about its results. Open-source project (especially community owned) paradigms significantly differ from proprietary paradigms. Proprietary models use a closed approach whereas Community based projects embrace higher flexibility. All these factors should be taken into account for fault prediction. Kaszycki et al [20] account for programmer's experience in addition to the software metrics to obtain better prediction model. Hence the source of the project data will require diversity to provide tangible generalizations for an experiment.

The authors revisited Lessmann et al [10] experiment using noisy data to verify if any substantial differences were seen compared to Lessmann et al's [10] results. The experiment resulted in similar findings where almost no difference was observed. The authors used AUC values coupled with Double Scott-Knott tests to rank their findings. The results reaffirmed the conclusions obtained by Lessmann et al [10]. 24 out of the 31 learned were grouped into the same rank, that is, there was no significant statistical difference between the 24 learners. The complete set of ranks is shown in Table 2. The bias and the noise present in NASA corpus was larger than anticipated. Wang et al [21] showed that presence of noise significantly alters the outcome of an experiment. In any experiment, noise is highly unavoidable problem. The conclusions and inferences gained from an experiment are heavily influenced by the level of noise in the training data and stability of the learner against noise.

Table 2. Result of Double Scott-Knott test for noisy NASA corpus

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Ad+NB, EM, RBFs, Ad+SL, RF+NB, Rsub+NB, Ad+SMO, K-means, KNN, NB, SL, Bag+NB, Rsub+SL, Ad+J48, Rsub+J48, Ad+LMT, LMT, RF+SL, Bag+SL, RF+J48, Rsub+LMT,Bag+J48, RF+LMT and Bag+LMT	3.2	3.03	0.97
2	Rsub+SMO, SMO, RF+SMO, Ridor, Bag+SMO, Ripper and J48	8	7.91	1.04

The authors further expanded their experiment to use the filtered data as suggested by Sheppard et al [19]. Upon repeating the experiment, four distinct classes (ranks) of classification techniques were obtained. The results reiterate the facts stated previously about the effects of noise and bias in the training data. The experiment was replicated again with PROMISE data sets. The result showed 4 consistent ranks into which the classification techniques can be grouped. Table 3 illustrates the results obtained after applying double Scott-Knott test on the PROMISE data sets.

The results concluded that classification techniques like Logistic Model Trees (LMT), Simple Logistic with ensemble methods like Bagging, Rotation Forest and Random Subspace outperformed all other classification techniques and hence can be considered the state-of-the-art algorithm to be examined for this essay. Logistic Model tree combines Logistic regression with Decision trees. The leaves of the decision tree contain logistic regression functions.

Table 3. Results of Double Scott-Knott test for PROMISE corpus

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Rsub+J48, SL, Rsub+SL, Bag+SL, LMT, RF+SL, RF+J48, Bag+LMT, Rsub+LMT, and RF+LMT	1.7	1.63	0.33
2	RBFs, Bag+J48, Ad+SL, KNN, RF+NB, Ad+LMT, NB, Rsub+NB, and Bag+NB	2.8	2.84	0.41
3	Ripper, EM, J48, Ad+NB, Bag+SMO, Ad+J48, Ad+SMO, and K-means	5.1	5.13	0.46
4	RF+SMO, Ridor, SMO,and Rsub+SMO	6.5	6.45	0.25

The authors used 10 iterations of 10-fold validation to obtain their results. The models were trained differently for clustering techniques where methods like K-Means cluster the training data and then Naïve Bayes is used to train models on individual clusters. Test data points are first classified as one of the clusters then the Naïve Bayes algorithm returns if the test data is bug prone or bug free. This approach can be surmised as context-aware as the individual clusters' local environment is considered before classifying the test data as fault prone.

The paper uses AUC as evaluation parameter for ranking the learners. Use of AUC can be highly contextual. In case of mission critical applications, where detecting Recall is more prominent, cost of false alarm rate can be ignored. Hence the authors should consider the context of the application under test to provide ranking based on other reliable parameters.

Ranking of Classification Techniques is performed by application of Double Scott-Knott test where for a given data set Scott-Knott test is applied to obtain the groups/ranks of the algorithms based on their statistical differences. Then the best algorithms are chosen for each data set and Scott-Knott test is applied again. The methodology is illustrated in Fig 1. Results obtained for the first level of Scott-Knott test is highly dependent on the data set chosen. The noise and bias in the training data can cause significant differences in the end results. As concluded by the experiments of Ghotra et al [17], the results published by Lessmann et al [10] was incapable of finding significant statistical differences between classification techniques due to noise and bias in the training data. Hence the ranks generated by first level Scott-Knott test is highly susceptible to errors. Executing second round of Scott-Knott test will further propagate these errors and result in wrong conclusions.

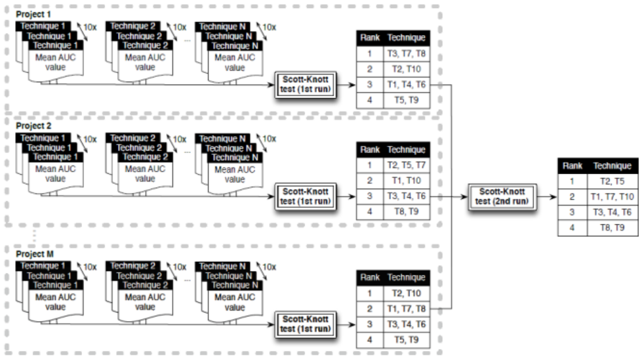

Figure 1. Double Scott-Knott approach used in Ghotra et al [17]

The authors acknowledge the need for parameter tuning for some classification techniques. But they propose tuning for only K Nearest Neighbors and K-Means. They conclude that there was no significant difference in performance for both the algorithms when tested with values of $k = \{2,3,4,5\}$ for K-Means and $K = \{2,4,6,8,16\}$ for KNN. The authors have tested a very limited parameter space and have failed to account tuning for other classification techniques. As explored in further sections of this essay, control parameters of learners in ensemble methods may be susceptible to coupling and hence should be taken into account. Learners like Random Forests can highly benefit from parameter tuning and might improve the results compared to Logistic Model Tree and Simple Logistic. Hence the LMT and Simple Logistic failed to use Self Tuning in their approach.

5 Review

Evolutionary algorithms can tune the parameters significantly faster when compared to other techniques. Menzies et al [4] have shown that Differential Evolution converges faster (~100 iterations) whereas grid search can take thousands of iterations to optimize the control parameters. It is also stated that differential evolution is comparable in performance to swarm optimization. But there might be cases where the number of iteration could exceed the grid search iterations. In such situations, we must observe if any trend in the input data set or the learner could be responsible for the increased number of iterations and provide evidence to confirm the hypothesis. If the number of iterations required for convergence is very high, then alternate variants of DE like DE/closest/1 can be examined for computation time reduction.

Other optimizations to the algorithm include Alternative Differential Evolution (ADE) (Ali et al [23]) where the mutation rule is modified based on the weighted difference between the best and worst candidates in a particular generation. Hence DE and its variants can be applied to the algorithm along with SMOTE for training data (for unbalanced data sets) to improve the overall predictive power of the algorithm.

6 Planning

The project is aimed at creating fault prediction models with highly tuned parameters. The work of Menzies et al [4] will be extended to support more data learning algorithms which can greatly benefit from tuning. Tuning algorithms like Grid search and Differential Evolution will be applied to the models to find the right set of control parameters to create efficient algorithms. Additionally, the impact of ensemble methods like Boosting and Bagging will be tested on a data model with well-tuned parameters.

Based on Menzies et al [4] argument that off- shelf tuned models do not perform very well, this project aims to test, compare and contrast 4 different scenarios where the models are tuned separately based on the context of the test.

The fault prediction models will be built using the data sets available from PROMISE and Seacraft repositories. Chidamber and Kemerer [5] metrics suite will be used to obtain the required metrics for various open-source projects. SMOTE will be used for data sets with imbalanced training data. Table 4 shows the list of data sets that will be considered for model training. Table 5 lists the Learners which will be used for evaluation. Table 6 and Table 7 list the ensemble methods and Tuning algorithms in the project respectively.

Table 4. List of datasets

Datasets (D_i)
Ant
Arc
Ivy
Jedit
Log4j
Lucene
Tomcat
Velocity
Xalan

Table 5. List of Learners

Learners (L_j)
Logistic Regression
Naive Bayes
Decision Trees
Random Forest
SVM
K Nearest Neighbors
K-Means
Ridor
Logistic Model Trees
Simple Logistic

Table 6. List of Ensemble Methods

Ensemble Methods (E_k)
Bagging
AdaBoost
Gradient Tree Boosting

Table 7. List of Tuning Algorithms

Tuning algorithms (T_q)
Grid Search
Gradient Optimization
Differential Evolution
Alternate Differential Evolution

Now the Test Scenarios (**TSx**) can be defined using the attributes defined above:

TS1: Evaluate the Learners' performance from $\{L_j\}$ with Datasets in $\{D_i\}$ using only their default settings and without using Ensemble methods or Tuning algorithms

TS2: Evaluate Learners' performance from $\{L_j\}$ with Datasets in $\{D_i\}$ using tuning algorithms $\{T_q\}$.

TS3: Evaluate Learners' performance from $\{L_j\}$ with Datasets in $\{D_i\}$ using ensemble methods $\{E_k\}$.

TS4: Evaluate the performance of Learners from $\{L_j\}$ and Datasets $\{D_i\}$ with the application of Tuning algorithms of $\{T_q\}$ for parameter optimization and Ensemble Methods $\{E_k\}$.

Performance parameters can be classified using Area under the curve (AUC), Recall, precision, F-score, and accuracy. In case of **TS4**, the ensemble method can contain multiple learners each with their own individual set of control parameters. Now the problem is modified to find the optimal performance using control parameters of all the learners that are part of the ensemble (**TS4a)**. This is a computationally intensive optimization problem because multiple models should be executed for every iteration of control parameters values. Alternatively, the problem can be simplified by first obtaining the tuned parameters for the models from **TS2** and then using these off-shelf tuned models to perform further tests (**TS4b)**. But it restricts the optimization space which can be obtained for the given ensemble. Also, Menzies et al [4] suggest that off-shelf models are not always optimized. Hence a preliminary test will be performed for **TS4a** and **TS4b** to check if **TS4a** is indeed computationally intensive than **TS4b**. If true, then **TS4a** will only be examined for a limited combination of models and **TS4b** will used to examine all the combinations.

Finally, to explore the solutions to the problems above the following research questions can be posed:

RQ1: *Does the application of ensemble methods (**TS3**) on the data learning algorithms always result in better performance than defaults (**TS1**)?*

RQ2: *Does tuning the parameters for learners (**TS2**) always result in better performance compared to defaults (**TS1**)?*

RQ3: *Does the combination of ensemble methods and parameter tuning (**TS4**) always yield better results compared to direct application of ensemble methods or parameter tuning? If not, which models perform the same or worse with the combination of ensemble methods and parameter tuning?*

These research question will be evaluated in the project to obtain tangible conclusions. For analyzing the time complexity of the tuning algorithms, the time taken for tuning will be reported as a function of model and data set size. To rank the results obtained from watch learner, stats.py will be used.

Python scikit-learn package will be used to code the algorithms for most of the above processes. Matplotlib will be used for visualizations. The plan is to use Azure workbench for computations because of the computational flexibilities provided by the platform and it can retain the state of a data model which otherwise would have to be performed by writing to a file. Azure workbench can use GPU enable VMs to reduce the computational time associated with the algorithms.

References

[1] Nadeem Ahmed Syed, Huan Liu and Kah Kay Sung. *Incremental Learning with Support Vector Machines*

[2] Haibo He, Sheng Chen, Kang Li and Xin Xu. *Incremental Learning from Stream Data*. IEEE Transactions on Neural Networks (Volume: 22, Issue: 12, Dec. 2011)

[3] Deb K, Burke E., Kendall G. *Multi-Objective Optimization.* Search Methodologies. Springer, Boston, MA

[4] Wei Fu, Tim Menzies, Xipeng Shen. "Tuning for Software Analytics: is it Really Necessary?"

[5] Chidamber and Kermer. "A metrics suite for Object Oriented Design" *IEEE transactions on Software Engineering (Volume 20, No 6, June 1994)*

[6] Tibor Gyimo´thy, Rudolf Ferenc, and Istva´n Siket. *Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction.* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 10, OCTOBER 2005

[7] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, Kenichi Matsumoto. *Automated Parameter Optimization of Classification Techniques for Defect Prediction Models*. 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering

[8] Cagatay Catal. *Software fault prediction: A literature review and current trends.*

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

[10] 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, vol. 34, no. 4, pp. 485–496, 2008

[11] A.A. Porter, R.W. Selby. *Empirically guided software development using metric-based classification trees*. IEEE Software, 7 (2) (1990), pp. 46-54

[12] Danijel Radjenović, Marjan Heričkob, Richard Torkarcd and AlešŽivkovič. *Software fault prediction metrics: A systematic literature review*. Information and Software Technology, Volume 55, Issue 8, August 2013, Pages 1397-1418

[13] Cagatay Catal, Banu Diri. *A systematic review of software fault prediction studies.* Expert Systems with Applications. Volume 36, Issue 4, May 2009, Pages 7346-7354

[14] Y. Jiang, B. Cukic, and T. Menzies. *Can Data Transformation Help in the Detection of Fault-prone Modules?* In Proceedings of the workshop on Defects in Large Software Systems.

[15] A. Günes Koru, Hongfang Liu. *An investigation of the effect of module size on defect prediction using static measures.* PROMISE '05 Proceedings of the 2005 workshop on Predictor models in software engineering.

[16] E. Kocaguneli, T. Menzies, A. B. Bener, and J. W. Keung. *Exploiting the essential assumptions of analogy based effort estimation*. Transactions on Software Engineering, 38(2):425–438, 2012.

[17] Baljinder Ghotra, Shane McIntosh, Ahmed E. Hassan. *Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models.* Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference

[18] R.Y. Wang, V.C. Storey, C.P. Firth. *A Framework for Analysis of Data Quality Research*. IEEE Transactions on Knowledge and Data Engineering 7 (1995) 623-640 doi: 10.1109/69.404034

[19] M. Shepperd, D. Bowes, and T. Hall. *Researcher bias: The use of machine learning in software effect prediction.* IEEE Transactions on Software Engineering, vol. 40, no. 6, pp. 603–616, 2014.

[20] Kaszycki, G. (1999). *Using process metrics to enhance software fault prediction models.* In Tenth international symposium on software reliability engineering, Boca Raton, Florida

[21] R.Y. Wang, V.C. Storey, C.P. Firth, *A Framework for Analysis of Data Quality Research*, IEEE Transactions on Knowledge and Data Engineering 7 (1995) 623-640 doi: 10.1109/69.404034

[22] N. Bettenburg, M. Nagappan, and A. E. Hassan, *Think locally, act globally: Improving defect and effort prediction models.* in Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. IEEE Press, 2012, pp. 60–69

[23] Ali W.Mohamed, Motaz Khorshid, Hegazy Z. Sabry. *An alternative differential evolution algorithm for global optimization*. Journal of Advanced Research (Volume 3, Issue 2, April 2012, Pages 149-165)

[24] Javier Béjar. *K-means vs Mini Batch K-means: A comparison*