Parameter Tuning for Comparison of Learners in ordinal data classification

Ankur Garg
North Carolina State University
Raleigh, NC
agarg12@ncsu.edu

Sanket Shahane
North Carolina State University
Raleigh, NC
svshahan@ncsu.edu

Chinmoy Baruah
North Carolina State University
Raleigh, NC
cbaruah@ncsu.edu

Abstract

Class labels are not always nominal. They can sometimes have ordinal relationships among them. Bug priority prediction is one such problem. Such problems give rise to the question whether we treat these problems as classification problems or regression problems. In this paper, we evaluate a technique which treats the problem as a regression problem and provides our critique of their conclusions based on some defined key criteria. We solve the problem using standard classification approaches along with hyper-parameter tuning and compare our results based on statistical measures.

CCS Concepts •**Software Engineering** → Bugs; Bug Priorities; •**Machine Learning** → Classification; Regression; Evaluation; Cross-validation; •**Statistics** → Statistical measures, Bootstrapping, Significance tests, Effect size tests.

Keywords Ordinal Categorical Labels, Regression, Bug Prediction, Statistical Evaluation, Self-tuning models

1 Introduction

Assigning priority levels to bugs is a major factor contributing towards fixing it. High priority bugs are more important to be fixed than low priority bugs. Increasing complexity of the software systems is directly correlated to the number of bugs detected/reported. Human evaluation of every bug reported is not always feasible and thus using machine learning techniques to automatically assign appropriate priority levels is a must. On a high level, Machine learning tasks are divided into supervised and unsupervised tasks depending upon what the nature of the data is. Having labeled data making predictions about it for the future makes it a supervised task whereas unsupervised tasks are generally grouping/clustering tasks where there is no label attribute attached to the data samples. Supervised ML tasks are further divided into Classification and Regression tasks having categorical and continuous labels respectively. Categorical labels are nominal attributes where order doesn't make sense {boy, girl}, e.g. Continuous labels are numerical attributes where order does make sense. Heart rate e.g. 72 bpm < 129 bpm.

An interesting fact about bug priorities is that these can be viewed as categories ranging from {p1 to p5}. However, the difference between p1 and p5 is not the same as the difference between p1 and p2. Thus, we can see that bug priorities are neither just ordinal nor just numerical. They are ordinal and categorical in nature at the same time since we have a fixed number of categories, but they have an ordering relationship between them {p1<p2<p3<p4<p5}. A natural question would be: What kind of Machine Learning technique should we use for such problems? Should we treat it as a pure classification problem or as a regression problem and bin the regression output into categories?

In this paper, we study an interesting approach DRONE proposed by Yuan Tian et.al. [1]. They treat this problem as a regression problem and have proposed a greedy algorithm to determine the appropriate bin ranges of the regression output to map it to bug classes. However, we solve the bug priority prediction problem using standard classification methods and compare our results with DRONE based on statistical measures.

The remainder of the paper is organized as follows: Section 2, briefly explains DRONE proposed in [1] and we also provide our critique of their technique. In Section 3 we establish the research question for this study. Section 4, details out the dataset used for the experiments and the feature generation and processing steps. In this section, we also list our evaluation criteria and goals. Section 5, gives a detailed explanation of the methodology that we follow for our experiments, and the algorithms we choose to explore. In Section 6, we present our experiment's results and compare them with DRONE and also propose a simple modification (DRONE V2) to the original DRONE algorithm which yields better results on the dataset we have chosen to use. Section 7 presents our conclusions and answers to the research questions. The paper concludes with Section 8 with discussion of the future scope, threats to validity, and reproducibility of the results of this research.

2 Related Work and Critique

We study the approach proposed method called DRONE by [1]. It is a regression-based approach which treats the bug priority prediction problem as a regression problem and

then bins the regression output into classes {P1, P2, P3, P4, P5}. They divide the dataset into two parts training set and validation set - 50:50 split.

DRONE works in two stages:

Step 1: Train a linear regression model on the training set.

Step 2: Based on the output of linear regression learner on the validation set:

a. Initialize the thresholds for binning.

b. Optimize these thresholds on the validation dataset using a greedy approach to maximize the Average F1 score.

The exact details about the greedy optimization algorithm can be found in [1].

Though DRONE seems to be a promising approach to predict bug priorities, some conclusions presented in [1] are neither convincing nor supported by statistical evaluations. [1] compares the DRONE with standard classification algorithms like Naive Bayes and SVM [11] without any hyper-parameter tuning and statistical evaluations. [2] has shown that hyper-parameter tuning can have a major impact on the results of the learners. This was the main motivation behind this project - to try to confirm (or contradict) the results in [1] by introducing hyper-parameter tuning.

Some of the conclusions made in [1] which we investigate in this project are:

1. Naive Bayes could not run to completion due to lack of memory even after providing 8GB RAM.

2. DRONE is better than standard classification algorithms (SVM [11]) for predicting bug priorities.

Naive Bayes is a very fast and has a low memory footprint. It works by simply updating the statistics of the data. Naive Bayes does not require keeping the input data stored for testing purposes. It is an eager learner [add reference to this shit]. This raises questions about the first claim.

Comparison between any learners must be supported by statistical evaluations. No evidence was found in [1] pertaining to any such evaluations. In addition to this, no details with regards to the parameters used to train the SVM learner or reasons behind selecting such parameter settings were found in [1]. Finally, the comparison was made only with SVM. Thus, we decide to investigate the second claim by introducing hyper-parameter tuning for a standard classification algorithm like Random Forest Classification.

3 Research Questions

In this section, we establish our research questions based on the critique provided in Section 2.

RQ1: Can Naive Bayes [12] run to completion on the data set used by [1]?

Yes

RQ2: Can Hyper-Parameter Tuning of standard classifiers have a significant impact in predicting bug priorities when compared to DRONE?

Yes

RQ3: Is there statistical evidence to support that DRONE is better than standard classification approaches for predicting bug priorities.

No

4 Background

4.1 Dataset

The dataset used in this project for all experiments is sourced from Eclipse Bugzilla [15] repository. We consider bug reports submitted between October 2001 and December 2007. In total, we used 103,805 bug reports. The raw data had 11 features for each bug reports such as severity, creation date, summary, author, component, etc. The dataset contains five classes - representing the five priority levels of the bugs (P1, P2, P3, P4, P5). Figure 1 shows a few sample from the dataset.

We use the raw features to derive 4 kinds of features - Temporal Features, Author related features, Product (or Component) related features and Summary features. The methodology used to generate these features was sourced from the original paper [1].

	ID	Summary	Product	Component	Severity	Priority
1	4629	Horizontal scroll bar appears too soon in editor (1GC32LW)	Platform	SWT	normal	P4
	4664	StyledText does not compute correct text width (1GELJXD)	Platform	SWT	normal	P2
2	4576	Thread suspend/resume errors in classes with the “same” name	JDT	Debug	normal	P1
	5083	Breakpoint not hit	JDT	Debug	normal	P1
3	4851	Print ignores print to file option (1GKXC30)	Platform	SWT	normal	P3
	5126	StyledText printing should implement ”print to file”	Platform	SWT	normal	P3

Figure 1. Sample rows from the dataset

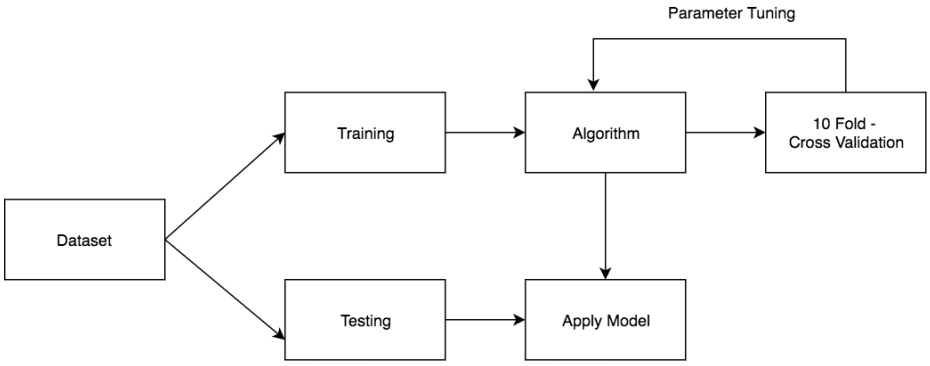

Figure 2. Overview of Methodology

Temporal Features represent the number of bugs that were reported in the last x days of the current bug report with same severity or priority. Author-related features represent the number of bugs that were authored by the same author and in the last x days. Product-related features represent the number of bugs that belong to the same component and were reported in the last x days. Based on these three types, total of 38 features are generated - 12 Temporal, 3 Author, 22 Product related. Apart from these, Severity of the bug report was used as is.

Finally, each bug report has a text feature which contains the summary of the bug report. To preprocess this, we use a count vectorizer and generate 18k sized count vector for each bug report. More details about each of these preprocessed features can be found in [1].

4.2 Evaluation Criteria

In this project, we use multiple criteria to compare the results of our experiments with the results of DRONE from [1]. We use macro F1 score, Average Precision, and Average Recall to compare the learners that we get from hyper-tuning standard classification methods with the DRONE algorithm. We use statistical t-test on cross-validation scores for making all such comparisons.

Apart from evaluating our learners in comparison with original DRONE algorithm, we also delve into following criterion to evaluate the usefulness of our model - Model Complexity and Model Stability [5].

More details about how these criteria are used in our experiments are provided in Section 6.

5 Methodology

Figure 2 gives an overview of the methodology that we follow in this paper. The original dataset after shuffling is divided into 2 parts in 80:20 ratios. 20% of the dataset is kept for final testing and is not used in any step of preprocessing or training. Remaining 80% of the dataset is used for first preprocessing the features as explained in Section 3.1. The trained preprocessors are used to generate features for test data as well.

Following algorithms are explored in the project and are compared with the original DRONE algorithm - Random Forest, Naive Bayes, and SVM. The original DRONE algorithm as mentioned in [1] is also implemented for the purposes of comparison.

I. Parameter Tuning

Our main focus in this project is on studying the effects of parameter tuning on classification techniques. Differential Evolution based parameter tuning is used to tune a Random Forest Classifier. Differential Evolution is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Figure 3 from [3] briefly explains how Differential evolution finds the 'best' solution. Differential Evolution is used for tuning three different measures - Average F1 Score, Average Precision and Average Recall.

Differential Evolution gives the best set of parameters for a particular goal {F1, Precision, Recall} but we find that there are many sets of parameters which yield similar results to that of the optimal set given by DE. Keeping in mind the model complexity [6][7] and model stability [5], we choose a simpler model (few trees in case of Random Forest Classifier) of optimal performance.

- Initialize all agents $\mathbf{x}$ with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
  - For each agent $\mathbf{x}$ in the population do:
    - Pick three agents $\mathbf{a}, \mathbf{b}$, and $\mathbf{c}$ from the population at random, they must be distinct from each other as well as from agent $\mathbf{x}$
    - Pick a random index $R \in \{1, \ldots, n\}$ (n being the dimensionality of the problem to be optimized).
    - Compute the agent's potentially new position $\mathbf{y} = [y_1, \ldots, y_n]$ as follows:
      - For each $i \in \{1, \ldots, n\}$, pick a uniformly distributed number $r_i \equiv U(0, 1)$
      - If $r_i < CR$ or $i = R$ then set $y_i = a_i + F \times (b_i - c_i)$ otherwise set $y_i = x_i$
      - (In essence, the new position is the outcome of the binary crossover of agent $\mathbf{x}$ with the intermediate agent $\mathbf{z} = \mathbf{a} + F \times (\mathbf{b} - \mathbf{c})$.)
    - If $f(\mathbf{y}) < f(\mathbf{x})$ then replace the agent in the population with the improved candidate solution, that is, replace $\mathbf{x}$ with $\mathbf{y}$ in the population.
- Pick the agent from the population that has the highest fitness or lowest cost and return it as the best found candidate solution.

Figure 3. Overview of Differential Evolution

II. Validation

10-Fold Cross-Validation [14] over the training dataset (80%) is used while optimizing for each of the three objectives using differential evolution. Results from the 10-fold cross-validation are also used for performing statistical tests (t-test).

III. Model Stability [5]

By model stability, we aim to determine whether the model has learned sufficiently from the given dataset and will providing additional training data drastically change the results expected errors. Stability test essentially tests whether the models has found a settlement between bias-variance. To conduct this test training data is incrementally provided in different percentages of the actual data and we measure and plot the training and testing performance based on the 10-fold cross-validation of the goal of that model. In an ideal case scenario, we expect the two curves (training, testing) to stabilize to a point where the slope is zero.

IV. Modification to DRONE – DRONE V2

The original algorithm follows a percentile based strategy to initialize the thresholds for binning the output of regression for each class. We modify this approach to make it much simpler. Instead of using a percentile based approach, we decide to initialize the thresholds for each class - (P1, P2, P3, P4, P5) as 0, 1, 2, 3, 4. Starting with these thresholds, we follow the original approach of greedy optimization over the average F1 score. We present the results for the same in the next section

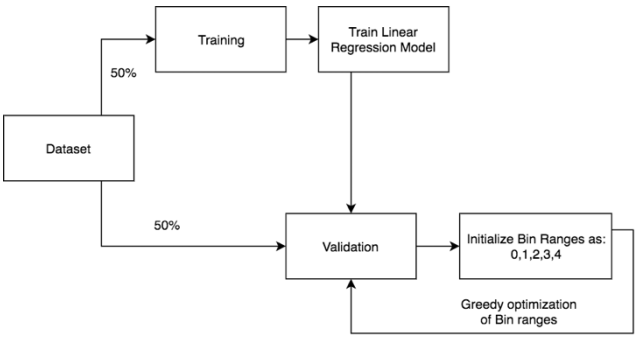

Figure 4: DRONE v2

6 Experimental Results

For all the experiments, we tune Random Forest learner [13], using the methodology described in section 5, on 3 hyper-parameters viz. 1. Number of Trees, 2. Minimum samples to split, and 3. Minimum samples at the leaf. We do not regulate the Max Depth of the trees as in our initial observations we found out that regulating this parameter has a negative effect on the overall results. The main reason for this being Depth of the tree directly contrasts Min Samples to Split and Minimum samples at leaf parameters and hence we leave Max Depth of the trees to be unregulated.

We use *scipy.optmize.differential_evolution* from the python package scipy [4] for running differential evolution. The parameters settings used for differential evolution function are as follows:

strategy: 'rand2bin', 'population_size' (also known as frontier): 30, mutation: (0.5, 1.9), recombination: 0.7. It was run for max iterations = 3.

In case of multiple learners with similar performance on any metric, we select one which is simpler. In this case of Random Forest, we choose the one with a lesser number of trees as model complexity tends to be very important during generalization [6][7]. We also check for model stability for that learner based on the methodology mentioned in previous section.

6.1 Experiment 1

In this experiment, we tune the learners for Average F1 scores across all the classes of bug priorities. Figure 5 shows the statistical evaluations based on Cliff's delta effect size test and parametric t-test on 10-fold cross-validation results on each of the learners. It can be observed from the figure that: 1. Random Forest with or without tuning significantly performs better than the DRONE. 2. Tuning the hyper-parameters of Random Forest Classifier yield significantly better models than the default off the shelf parameters.

Before choosing a simpler Random Forest model with 19 trees, 26 Min samples to split, and 1 Min Sample at the leaf, we look at the stability curve and conclude that the model is simpler. Figure 9 shows the stability curve of RF {19,13,1}.

Table 1 shows the results of RF {19,13,1} compared to DRONE on the evaluation criteria of the Average F1 score. We find that Random Forest model tuned for average F1 Score performs significantly better as compared to DRONE.

We observe an improvement of 70% compared to DRONE.

6.2 Experiment 2

In this experiment, we tune the learners for Average Precision scores across all the classes of bug priorities. Figure 6 shows the statistical evaluations on 10-fold cross-validation results on each of the learners. It can be observed from the figure that: 1. Random Forest with or without tuning significantly performs better than the DRONE. 2. Tuned and non-tuned random forest model are very similar in precision scores.

We choose model with 25 trees, 2 min split samples and 1 min sample leaf. The reason being that this model has least variability as compared to other models. Stability curve for the same in Figure 10 shows that the model is quite stable.

Table 2 shows the results of RF {25,2,1} compared to DRONE on the evaluation criteria of Average Precision. Random Forest model tuned for average Precision performs around 100% better as compared to DRONE.

hyper-parameters of Random Forest Classifier yield a huge difference than the default off the shelf parameters.

Before choosing a simpler Random Forest model with 18 trees, 12 Min samples to split, and 2 Min Sample at the leaf, we look at the stability curve and conclude that the model is simpler. Figure 11 shows the stability curve of RF {18,12,2}.

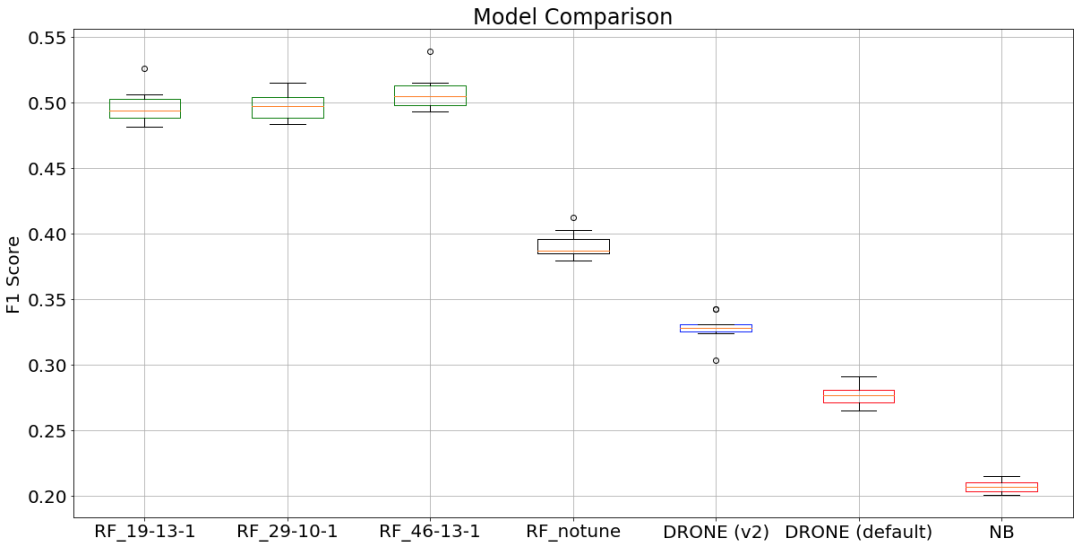

Figure 5: Comparison of algorithm based on Average F1 Score

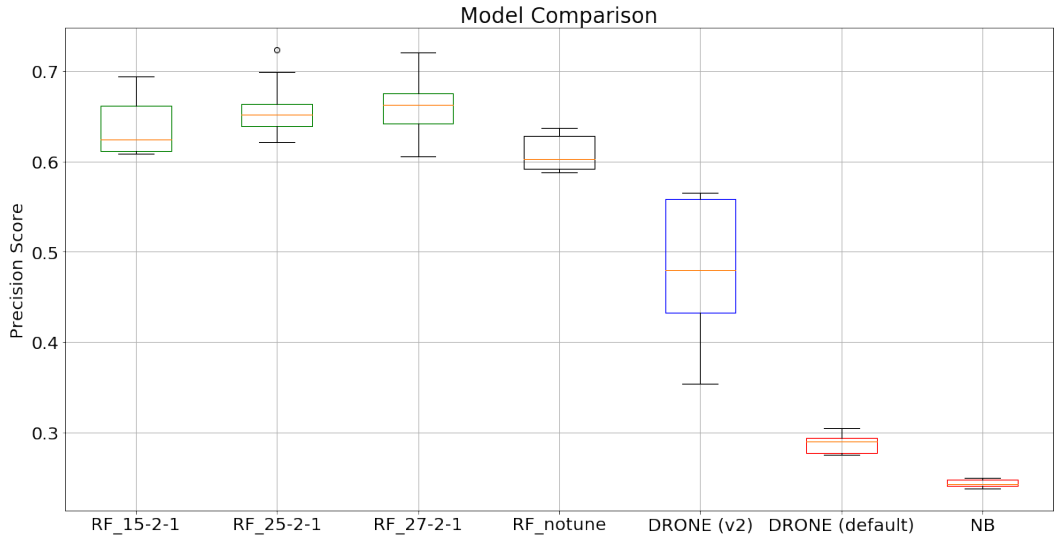

Figure 6: Comparison of algorithm based on Average Precision Score

6.3 Experiment 3

In this experiment, we tune the learners for Average Recall scores across all the classes of bug priorities. Figure 7 shows the statistical evaluations based on 10-fold cross-validation results on each of the learners. It can be observed from the figure that Random Forest with or without tuning significantly performs better than the DRONE. Tuning the

Table 3 shows the results of RF {18,12,2} compared to DRONE on the evaluation criteria of Average Recall score. We observe an improvement of around 200% as compared to non-tuned parameters and around 80% improvement compared to DRONE.

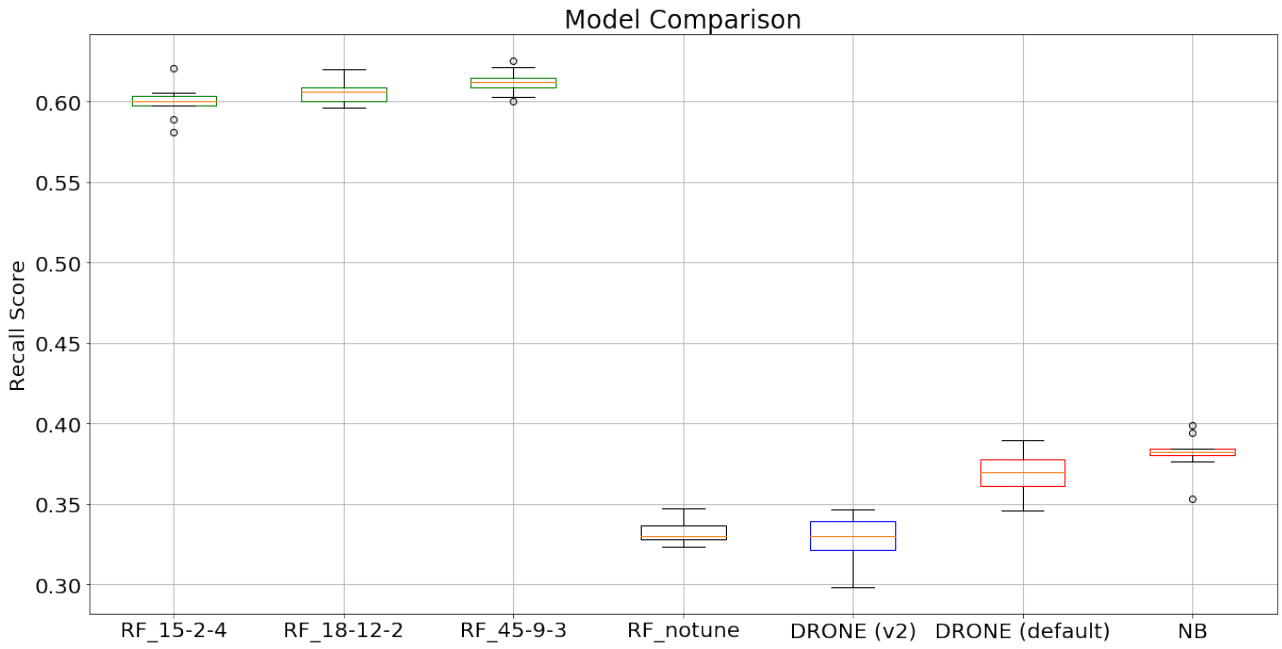

Figure 7: Comparison of algorithm based on Average Recall Score

Naive Bayes	DRONE	RF: no tuning	RF: tuning	SVM (1 Iter)
3.8 sec	15 mins	1 min 47 sec	3 mins 41 secs + 45 mins tuning	5 hrs 52 mins

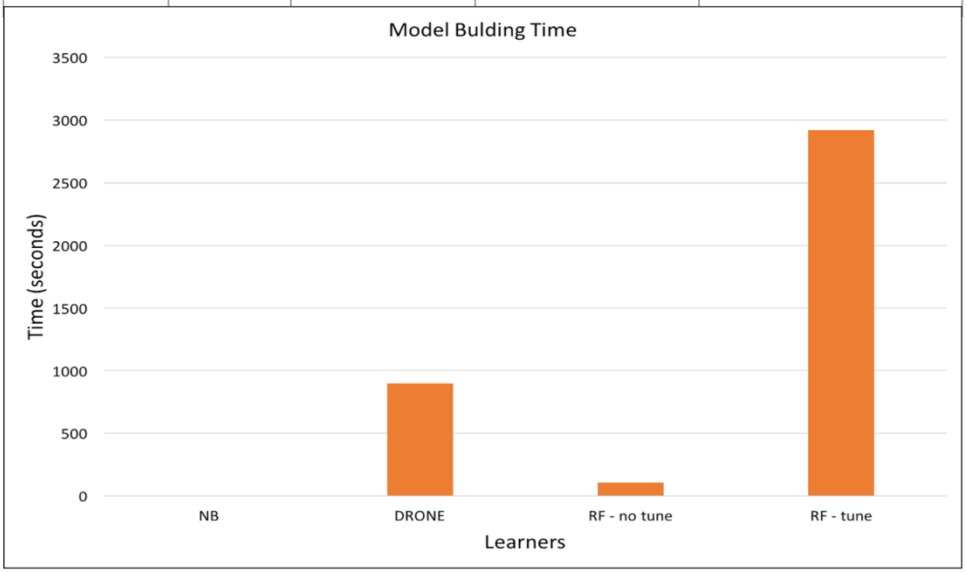

Figure 8: Comparison of Model building time

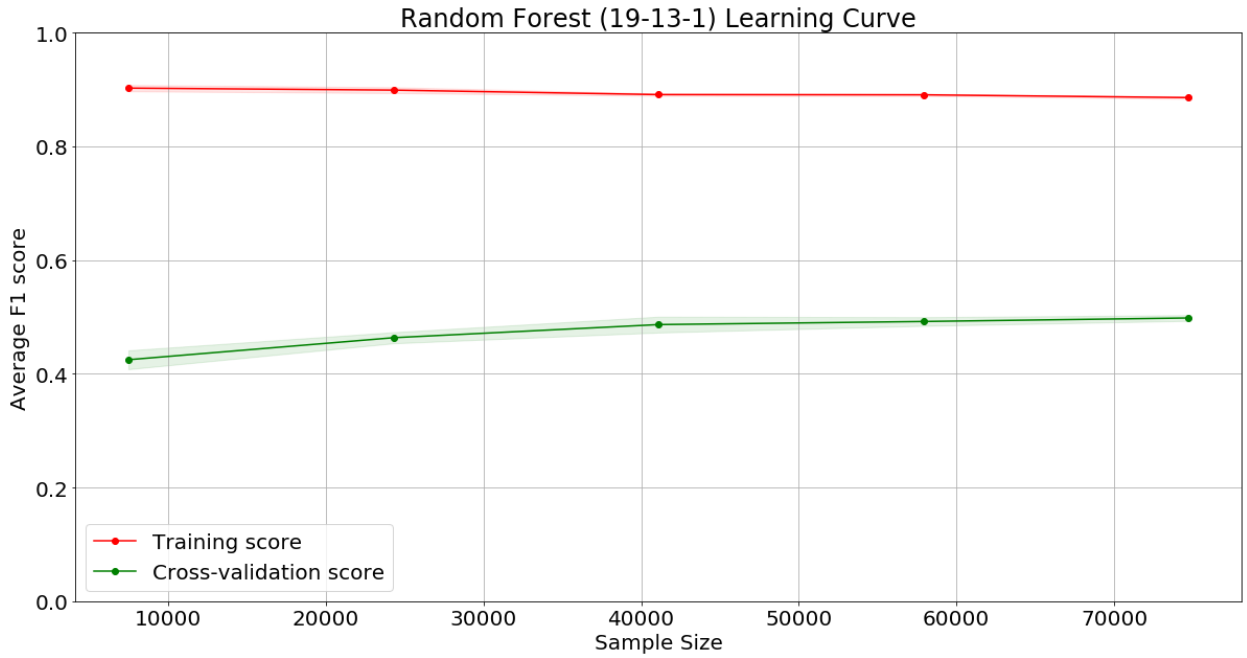

Figure 9: Model Stability for RF Model selected after tuning for F1 Score

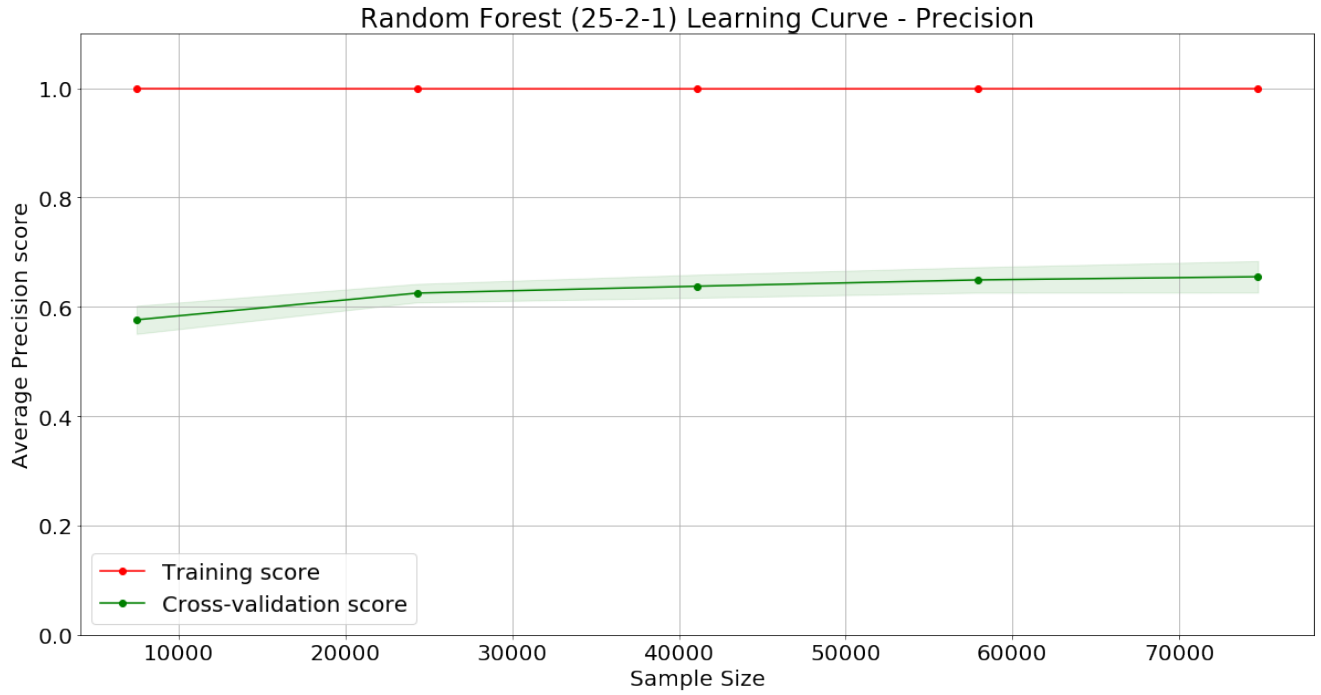

Figure 10: Model Stability for RF Model selected after tuning for Precision Score

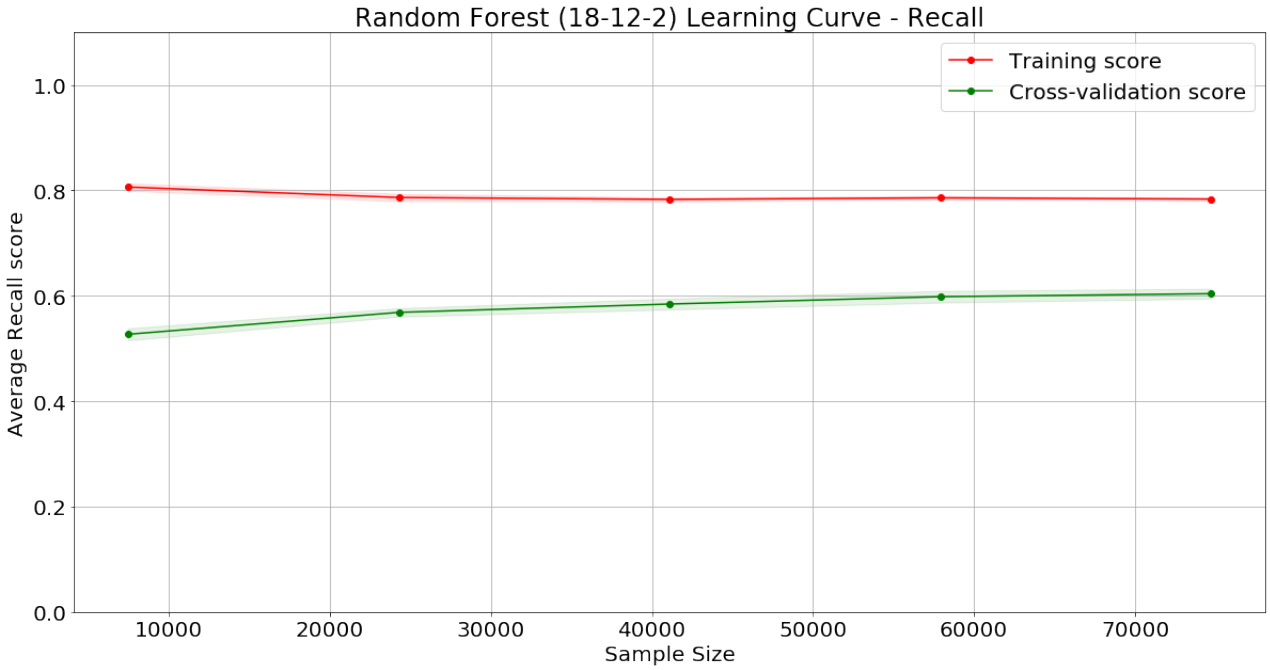

Figure 11: Model Stability for RF Model selected after tuning for Recall Score

Class	DRONE	RF – {19, 13, 1}
P1	29.4%	38.1%
P2	20.0%	33.7%
P3	88.7%	91.5%
P4	0.0%	33.8%
P5	6.5%	47.7%
Average	**28.9%**	**49.1%**

Table 1: Average F1 Score

Class	DRONE	RF – {18, 12, 2}
P1	22.4%	61.97%
P2	14.8%	46.05%
P3	79.2%	68.34%
P4	0.0%	55.13%
P5	6.55%	65.83%
Average	**36.4%**	**59.47%**

Table 2: Average Recall Score

Class	DRONE	RF – {25, 2,1}
P1	29.4%	61.80%
P2	20.0%	48.46%
P3	88.7%	87.79%
P4	0.0%	53.27%
P5	6.5%	60.13%
Average	**28.9%**	**62.30%**

Table 3: Average Precision Score

SVM was also used to train a learner on this dataset. But when using and Intel i7 processor with 8 cores and 16 GB RAM, it took around 6 hours to run a single iteration of training and testing (without validation). Due to constraints of computing resources, we could compare results from SVM with other algorithms.

Parameter Tuning for Random Forest, for each objective, on the above-mentioned computing resource, took about 45 mins to find the optimal parameters. But in each case, we find statistically significant improvement in the results as compared to non-tuned leaners. This is significant because we see drastic improvement without many tradeoff in terms of time taken to tune the parameters. Figure 8 also provides a comparison among the algorithms used in these experiments. We find that Naïve Bayes is one of the fastest and Random Forest with tuning takes about 3 times the time for DRONE.

We also observe that, the results we get for our DRONE implementation as very close to the ones mentioned in [1]. Specifically, for class P4, both our implementation as well as the [1] seem to give very close to zero for all metrics.

For all metrics (F1-measure, Precision, and Recall), we find that Random Forest performed significantly better than DRONE. We observe that tuning for specific objectives (one of the metrics), can make a huge difference in performance. We find no evidence to support the claim that DRONE performs better than standard classification algorithms.

Finally, we also find that in DRONE v2, simplifying the methodology of DRONE to use simple initial thresholds does improve the results significantly as compared to original algorithm DRONE.

7 Conclusions

In this section, we answer the research questions:

RQ1: Can Naive Bayes run to completion on the data set used by [1]?

Yes, we were able to run Naive Bayes [12] to completion of the dataset described in section 4. In fact, Figure 8 shows that Naive Bayes was the fastest of all the learners to train, cross validate, and test.

RQ2: Can Hyper-Parameter Tuning of standard classifiers have a significant impact in predicting bug priorities when compared to DRONE?

Yes, Figures 5,6,7 discussed in section 6 show that hyper-parameter provides a statistically significant improvement in results.

RQ3: Is there statistical evidence to support that DRONE is better than standard classification approaches for predicting bug priorities.

No, there is not enough statistical evidence to support the claim that DRONE is better than standard classification approaches for predicting bug priorities. In case of Random Forest (with and without tuning) we have found enough statistical evidence to claim that a standard classification technique is better than DRONE on the eclipse bug report dataset.

8 Discussion and Future Work

Through the experiments, we find that, parameter tuning can make a statistically significant difference to the performance of a learner. We observe this effect across different types of objectives. While comparing any kind of learners, it is imperative that we perform parameter tuning on the measure for which we are making such comparisons.

Another important inference was, before choosing any model, it is really important to measure the stability of the model. It provides a measure of how well the model has been trained. Does it need more data for better performance? Or has it over-fit on the current dataset? In our experiments, we choose any model only after performing this check.

I. Threats to Validity of the results

In this paper, we do not claim that standard classification methods are always better than DRONE in every case of predicting bug priorities. Only in case of Eclipse bug reports from 2001 to 2007 based on our experiments we conducted we claim that predicting bug priorities using a standard classification approach is better than treating it as a regression approach as opposed to the claims made by [1]. We suggest that the above methodology must be followed whenever the dataset itself changes. There might be cases where regression might outperform classification. The only way to know the best technique to predict bug priorities is to try both.

II. Future Scope

In case of ordinal data classification, deciding between regression-based methods or classification based methods is tricky. For this particular dataset, we find that classification based methods perform better than DRONE. We would like to research this further by conducting this experiment on multiple datasets of similar ordinal properties. We would also like to investigate other approaches for ordinal data classification (apart from standard classification algorithms) such as the ones described in [8]. Another possible point to investigate would be the thresholding method in DRONE as we found

that making simple changes to it resulted in statistically significant improvement in results (DRONE v2). This will provide a better understanding in deciding which methodology performs better. We would also like to explore other methods of hyper-parameter tuning. One such method which we found interesting is Particle Swarm Optimization [9]. These experiments would provide further insight into the problem of ordinal data classification and how parameter tuning affects it.

III. Reproducibility of results

Reproducibility of results being the utmost requirement of any scientific endeavor, we open source all of our code, data, and results in our GitHub repository [10] along with the instructions to reproduce them and advance the field of research.

References

[1] Yuan Tian, David Lo, and Chengnian Sun. 2013 DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis. IEEE International Conference on Software Maintenance DOI. http://dx.doi.org/10.1145/1188913.1188915

[2] Wei Fu, Tim Menzies, Xipeng Shen. 2016. Tuning for Software Analytics: is it Really Necessary?. Department of Computer Science, North Carolina State University, Raleigh, NC, USA.

[3] Differential Evolution Wikipedia. https://en.wikipedia.org/wiki/Differential_evolution

[4] Python Scipy Library. https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.optimize.differential_evolution.html

[5] K.M. Ting and R.J.Y. Quek. Model stability: a key factor in determining whether an algorithm produces an optimal model from a matching distribution

[6] Jae Myung. The Importance of Complexity in Model Selection.

[7] Garret Wu. Why More Data and Simple Algorithms Beat Complex Analytics Models?

[8] Eibe Frank and Mark Hall. A Simple Approach to Ordinal Classification.

[9] J. Kennedy, R. Eberhart. Particle Swarm Optimization. IEEE Neural Networks 1995.

[10] https://github.com/iankurgarg/FSS-Fall-2017-GRP-I/

[11] http://scikit-learn.org/stable/modules/svm.html#multi-class-classification

[12] http://scikit-learn.org/stable/modules/naive_bayes.html

[13] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[14] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[15] Eclipse Bugzilla for Dataset. https://bugs.eclipse.org/bugs/