# Parameter Tuning for Comparison of Learners in ordinal data classification

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

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

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

## Abstract

Class labels are not always nominal in nature 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 provide our critique on their conclusions based on some defined key criteria. We also propose our plan to statistically check their conclusions based on the key criteria**.**

***CCS Concepts*** •**Software Engineering** → Bugs; Bug Priorities; •**Machine Learning** → Classification; Regression; Evaluation; Cross-validation; •**Statistics** → Statistical measures, Boot-strapping, 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 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 labelled 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 ordering doesn't make sense {boy, girl} for e.g. Continuous labels are numerical attributes where order does make sense. Heart rate for e.g. 72 bmp < 129 bpm.

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 difference between p1 and p2. Thus, we can see that bug priorities are neither just ordinal nor just numerical. They are ordinal categorical in nature 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 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. They compare their study with [2] and show that their approach is better than treating the problem as a classification problem. This conclusion is however, a complex technique and is supported by less statistical evidence. We have evaluated their technique according to the criteria presented in section 2 and focus on the key criteria presented in section 3 of the paper. In section 4 we provide our critique of [1] and in section 5 and 6 we propose the technologies, methodologies, and our plan to compare techniques which enable the key criteria and statistically compare it with the approach presented by [1].

## 2 Criteria

As per [3], the following are important criteria for evaluating a learner. We discuss these criteria in this section and then in the Section 3, provide key criteria which we are going to focus on in our case study.

### 2.1 Model Readability

Model Readability is a measure that defines how easily can the model be interpreted by the user. A model is readable if it is easily understood by someone. It is readable if the results of the model "make sense" and can be easily explained to the user. It is an important measure as better the readability of the model, more comfortable the end user will be when using it. Simpler a model, easier it is to change something in it or experiment with it.

The ease with which a model is understood by someone can affect its usability. Major part of data mining is to create learners and models which can be explained to the business users. Some such learners which perform well on this criterion are Decision Trees, Rule based classifiers, etc.

## 2.2 Learnability and Repeatability of Results

Learnability of model is measure of the amount of resources required by the model to learn. It defines how easy it is for us to train a model. The effort could be in terms of time taken or the amount of computing resources required etc. The amount of time is takes to train an instance of a learner could seriously affect the ease with which you can experiment with it. For example, most deep learning algorithms take a lot of time to train (days and even weeks) even on high end hardware

If an algorithm or a learner requires a lot of expensive hardware or huge amounts of time, it makes it very difficult for someone to conduct same or similar experiments. This restricts the amount of experiments that can be conducted on the learner. Repeatability is very important to determine how well a particular learner performs in various scenarios. If a result of a learner or a model is not repeatable, it is little or no use as it can't be used to build further research on top of it.

Apart from the kind of resources that a particular learner requires, there could be other reasons why the results from a particular algorithm may not be repeatable. A lot of results published do not make the data publicly accessible, especially for industry research papers. For any reason, if the dataset or any other resources used in the research paper are not made publicly (or in limited terms) accessible to someone else, the results of that paper may not be reproducible at all.

## 2.3 Anomaly Detection

Anomaly detection is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset. There are three broad categories of anomaly detection techniques. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier. This classification task would work on a highly imbalanced dataset. Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set, and then testing the likelihood of a test instance to be generated by the learnt model.

An important property of a learner is to be able to detect anomalous behavior in data. A "good" learner or model would be one which not only performs good on relevant data but also detects it when "strange" data is given as input to it.

Every learner work good only for a particular use case that they were trained for (no free lunches). It is extremely important that the results of the learner are used carefully after examining that the input given was actually relevant to it. Any model that is able to detect possible anomalous behavior would be able to warn users to interpret the results cautiously.

This ability of a learner to be able to detect that input data is not the "usual" input can have very serious consequences. There are many learners which perform or can be modified to perform such operations. For example, Random Forests, KNN etc.

## 2.4 Context Aware

Most learners are designed in a way that they are able to identify "commonalities" in the dataset. This is the objective of most learners, which is useful. But, another very important aspect of learner being "good" is its ability to identify differences between various parts of the dataset. This means that a "good" learner should be able to give different inferences for different sections of the data.

If a model makes assumption about the dataset which ignore possible sub-sections in the data and treat the whole dataset as a single homogeneous set, it could lead to model giving results which vary a great deal from the actual facts.

Building models with ability to detect such differences or sub-contexts in a data set is not easy. At least not without prior knowledge about the data set. The source of the dataset, its semantic meaning, the context in which it was collected etc.

## 2.5 Incremental Learning

Incremental learning is a machine learning paradigm where the learning process takes place whenever new example(s) emerge and adjusts what has been learned according to the new example(s). It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time or its size is out of system memory limits.

A learner which can learn incrementally can be quite useful in particular scenarios. Any scenario where the dataset changes very frequently and past data becomes increasingly irrelevant in the context, would need a learner which can incrementally learn new patterns from new incoming data and slowly decrease the influence of old data.

There are many applications of Incremental algorithms: Frequently applied to data streams or big data, addressing issues in data availability and resource scarcity respectively. Stock trend prediction and user profiling are some examples of data streams where new data becomes continuously available. Applying incremental learning to big data can help in producing faster classification or forecasting times.

### 2.6 Self-Tuning

Conclusions drawn from a learner can be extremely dependent on the various control parameters of any algorithm. Most algorithms or software systems in use have very large number of input parameters which need to be tuned by the user before the learner can be modelled. In fact, in most scenarios, the "right" or the "best" parameters would change drastically based on data, context and point of application of the drawn conclusions.

Increasingly, we find most learners with more and more set of tuning parameters which provide for great flexibility in terms of the kind of learners that can be built but that also means the person building this learner needs to provide the "correct" parameters. This can be very challenging and time consuming.

Any learner or algorithm that incorporates a mechanism through which these parameters can learnt automatically through dataset is extremely useful as it can reduce the effort of the developer drastically and further reduce the number of scenarios where a sub-optimal (or rather totally irrelevant) learners are produced because of "wrong" control parameters.

### 2.7 Multi Goal Reasoning

There are some systems which tend to provide multiple outcomes/goals for a particular problem, and these goals are in direct competition with each other. In such a case, we are supposed to select that goal which would provide the best outcome for that particular scenario.

Most problems in real world consist of scenarios where we have more than one goal. It is quite usual to find problems which need learners to find optimal solutions constrained by more than one parameter. In most cases, there is no one "correct" solution, instead, there are trade-offs among multiple goals based on different solutions provided by the learner.

On way to solve the problem of multiple goals is to map the multiple goals onto a single goal. This could be done using something like domination score. This single goal could then be used to optimize the learner easily (easy in comparison to training a leaner on multiple goals).

### 2.8 Shareable

Share-ability comes into picture when the model is built using data sets which contain sensitive information. These datasets can't be shared (at-least not in their raw form), due to multiple reasons like privacy concerns or security concerns. A shareable model would be one which can be shared with other without the need for such concerns.

In such scenarios, we need to find ways to share data by data modification or anonymization. Any such technique may lead to changes in data which may not be able to reproduce the same models. A shareable model would be one where we would be able to reproduce models even after datasets have been obfuscated.

This can be really important in many scenarios where information is confidential and can't be made public as is.

## 3 Key Criteria

### 3.1 Parameter Tuning

Parameter tuning is a very significant task while making inter-model comparisons. Most learners available provide a large set of parameters which provide a lot of flexibility in terms of the kind of learners that are generated. Fu, Menzies et. al. [4] provide an extensive analysis of how control parameters can drastically affect the performance of the software systems. For example, learners like Random Forest have many parameters like number of estimators, max depth of tree, min nodes in a leaf, etc. Exploring and finding the optimal (or close-to-optimal) values of such parameters is really important. But it is also very expensive in terms computation resources and time needed if one needs to try out all possible parameter settings to find an optimal one. It is highly important to formulate an efficient strategy to find such optimal parameters.

[4] Fu, Menzies et. al. discuss techniques like Differential Evolution which can be used to find close-to-optimal parameter settings with much less computational requirements. Such techniques can be used to parameter settings which can improve results as compared to default parameter settings for most learners.

Comparison between learners requires such techniques to be used to find appropriate parameter settings for the learners before making any conclusions or claims about the performance of learners. All comparisons between learners can be challenged based on lack of parameter tuning as it can have drastic effect on the kind of results that are obtained.

### 3.2 Learnability and Repeatability of Results

It is very important to be able to reproduce the results of any learner. Any results or claims made using any kind of experiment need to be tested and repeated to improve the reliability of such claims. Most learner comparisons involve a lot of assumptions inherent in the models. Repeating such experiments can help identify such assumptions and detect possible errors in the experimental setup which could have altered the results.

# 4 Critique

In this case study, we plan to analyze, reproduce and challenge the results from the “DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis” by Yuan Tian, David Lo & Chengnian Sun. The paper introduces a technique to use regression for modelling ordinal categorical data and compares it with standard classification techniques (SVM, NB).

[1] compares their complex yet wonderful approach with Menzies’ algorithm [2] and shows that their’ is better. The comparison has been done between their proposed approach and SVM. However, we find that their experimental setup has some possible flaws which could have impacted the results. One of the things missing is their validation approach. Single 50-50 split has been used which can introduce high risk of bias in the models.

Handling Imbalanced data: The problem discussed in the paper is a class imbalance problem. Before using any kind of model or training technique, we need to find ways to handle the class imbalance between the various classes present in the data. The paper makes comparison of their technique with standard classification techniques without using any techniques to pre-process the data to handle class imbalance problem.

The paper doesn’t use any sampling (under-sampling/over-sampling of majority/minority classes) to balance the classes in the dataset. It is observed that more analysis is required with different class imbalance techniques to understand how well the proposed technique performs in comparison to the existing classification techniques. We did not find its comparison with cases, where any such techniques are used. We plan to make the comparison between their approach and other classification algorithms by including such techniques.

Using Regression for Ordinal data: The paper claims that the proposed approach of treating output variable (ordinal variable) as continuous regression problem works better than treating the output variable as categorical variable and using standard classification techniques have multiple problems. One of the results stated in the paper is that Naive Bayes could not be run despite of supplying 9GB RAM. This is highly surprising and we would like to investigate this result.

The claims made in the paper are based on comparisons between the proposed approach to SVM. But we couldn’t find any details pertaining to the hyper-parameter tuning for SVM that was done to arrive at an optimal (or close-to-optimal) SVM model. This raises serious concerns with regards to the claims made in the paper.

Based on these observations we plan to investigate the following hypotheses:

Hypothesis 1: Hyper-parameter tuning will give better results than using off the shelf parameters. Hence, our approach will be to do hyper-parameter tuning on different classification techniques like SVM, Random Forest, and Naïve Bayes.

The paper’s result state that Naïve Bayes’ could not run to completion despite of 9 GB RAM. We find this highly surprising given that Naïve Bayes’ is very fast and has low memory footprint. We will try to reproduce the results of Naïve Bayes algorithm.

Hypothesis 2: Standard classification techniques require data preprocessing to handle class imbalance problem. Paper does not seem to use any such technique while comparing SVM with their proposed approach. Thus, our hypothesis is that handling class imbalance problem for standard classification techniques should yield better results.

# 5 Review

For the above hypotheses, we review 2 simple yet effective technologies in this section:

## 5.1 SMOTE with Under Sampling of Majority Class

SMOTE (Synthetic Minority Over-Sampling Technique) [5] combined with under sampling can be used to handle the imbalanced dataset. It works in two steps:

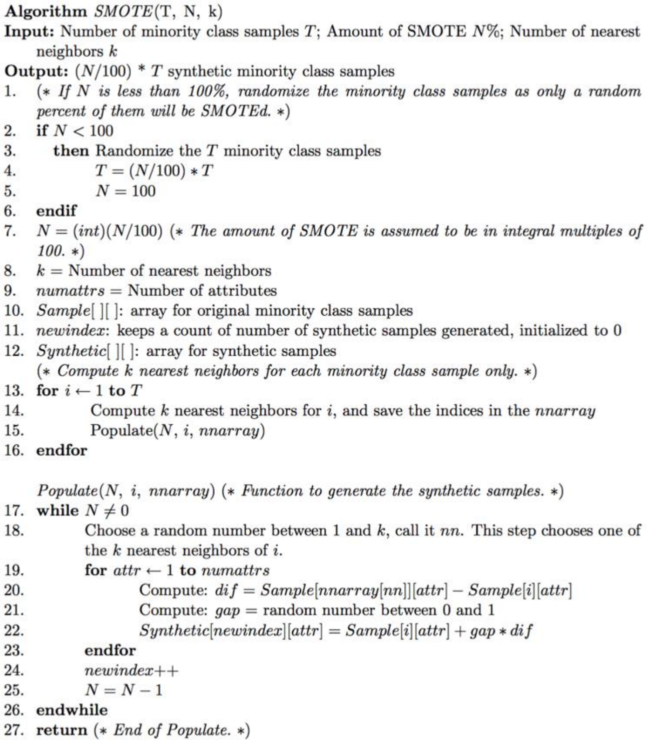
1. Under-sampling majority class
2. Smart Oversampling of the minority class using SMOTE.

Reducing the number of samples of the majority class to a certain threshold. This is the under-sampling step of handling imbalanced dataset. However, we also need to increase the number of minority class samples. One way to it would be by sampling with replacement but it creates a higher chance of overfitting the models during training. Thus, to avoid this SMOTE is used. SMOTE works on the principle of nearest neighbors. It finds *k* neighbors (a hyper-parameter of SMOTE) of a minority class data sample and randomly chooses one of the *k* samples. Then it creates a new sample which has properties in between these two samples. Consider the current sample *X* and chosen neighbor *Y*, SMOTE creates a new sample *Z*, where $Z = X+c*Y$, where *c* ranges from 0 to 1 and is another hyper-parameter of SMOTE. Figure 1 shows the algorithm presented in [5].

## 5.2 DE (Differential Evolution)

DE algorithm for Hyper-parameter tuning. Hyper-parameter tuning has been the much known yet ignored part of building Machine Learning models. A study done in [4] Menzies et al. have shown the effect that hyper-parameter tuning and the relative cost associated with it. Grid Search is a brute force approach of trying every combination in the Cartesian product of the parameter set and choosing the best one. However, it takes a lot of time to find the optimal parameter and the parameter space explored is limited due to computational limitations.

Differential Evolution for hyper-parameter optimization is a technique based off of genetic algorithm where the population is not mutated independently but by preserving the relationship among them. In a nutshell, we initialize a frontier (vector of sets of parameters) of the size of 10**n* where n is the number of decisions to be taken. Then for each of the set in the frontier, three other sets are randomly chosen from the frontier to mutate as *New = X + c*f*(Y-Z).* Here *c, f* are the hyper-parameters and stand for the probability with which the mutation occurs (cross over factor), proportion of (*Y-Z)* to include into *New* respectively. Both range from 0 to 1. Finally, we train the model on *New* and *X* and if *New* is better than *X* replace *X* with *New*. *X* is an item in the frontier chosen in a circular queue logic. [4] describes a special Differential Evolution algorithm with early stopping criteria and shows that hyper-parameter tuning is necessary and affordable.



```
Algorithm $SMOTE$(T, N, k)
Input: Number of minority class samples $T$; Amount of SMOTE $N$%; Number of nearest
    neighbors $k$
Output: $(N/100) * T$ synthetic minority class samples
(* If $N$ is less than 100%, randomize the minority class samples as only a random
    percent of them will be SMOTEd. *)
if $N < 100$
  then Randomize the $T$ minority class samples
        $T = (N/100) * T$
        $N = 100$
endif
$N = (int)(N/100)$ (* The amount of SMOTE is assumed to be in integral multiples of
    100. *)
$k$ = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(* Compute k nearest neighbors for each minority class sample only. *)
for $i \leftarrow 1$ to $T$
        Compute $k$ nearest neighbors for $i$, and save the indices in the $nnarray$
        Populate($N$, $i$, $nnarray$)
endfor

Populate($N$, $i$, $nnarray$) (* Function to generate the synthetic samples. *)
while $N \neq 0$
        Choose a random number between 1 and $k$, call it $nn$. This step chooses one of
        the $k$ nearest neighbors of $i$.
        for $attr \leftarrow 1$ to $numattrs$
                Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
                Compute: $gap$ = random number between 0 and 1
                $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
        endfor
        $newindex$++
        $N = N - 1$
endwhile
return (* End of Populate. *)
```

Figure 1: SMOTE Algorithm

## 6 Planning

In this section, we layout our plan for testing the hypothesis presented in Section 3.

1. Collect the dataset described in [1] through Bugzilla.

2. Handle the imbalanced class problem through data preprocessing and using SMOTE [5] as described in Section 5 using [6].

3. Reproduce results from the paper for SVM [7] and Naïve Bayes [8].

4. Extract features and run multiple learners like SVM [7], Random Forest [9], Naïve Bayes [8] etc. with appropriate hyper-parameter tuning using DE [4] as described in Section 5.

5. Use cross-validation [10] to compare the learners.

6. Use statistical test like significance test and effect size test to make conclusions.

## 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] T. Menzies and A. Marcus, “Automated severity assessment of software defect reports,” in ICSM, 2008.

[3] https://txt.github.io/fss17/essay

[4] 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.

[5] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. Journal of Artificial Intelligence Research 16 (2002) 321–357. Comm. ACM 38, 4 (2002), 393–422.

[6] http://contrib.scikit-learn.org/imbalanced-learn/stable/generated/imblearn.over_sampling.SMOTE.html

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

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

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

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