International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 08 | Aug 2024 www.irjet.net p-ISSN: 2395-0072

Requirement risk prediction model(Impact of Fine Tuning)

Krishna Kant Bhardwaj

Birla Institue of Tecnology and Science
Pilani
Rajasthan, India

Abstract - *An This Paper address two major problems that are faced in the software development projects. Basically, when defects are detected in testing phase, we have to perform Root causal analysis to find out the root cause whether it is injection or removal root cause and then accordingly we perform the preventive and the corrective actions. However, the causal analysis takes significant amount of effort and time of development team. So, using textual analysis we find out the root cause of various defects and accordingly categories defects and take preventive actions. Another problem is that in testing phase if in last phase we found that the defects are more than the expected count and due to which there is a chance of delay in milestone/release of the product so in order to address this issue. We will be using the dataset of defects of previous 5 years products and using ANN model will be trained to predict risky requirements based on various factors, like design complexity, code changes, side case or coverage area of changes, developer's skill. For example, bigger code changes near the release may risk in quality. Also, one of the factors would be the relevant functionality module. Based on the predicted risky requirements, effort in testing could be increased and also development team will also take action in the relevant module. We will also try using ML algorithm Gradient Boost on same dataset and compare the performance with ANN*

Key Words: **ANN, ML, Gradient Boost, Softmax, SME, SMOTE**

1.INTRODUCTION

Requirement Engineering is the efficient and systematic approach for gathering user's requirements to implement software solution. It is a process of describing, understanding and maintaining requirements in the process of engineering design. Mostly all the software saves some basic features and they are basically the requirements which need to be developed by the project team. QCD depends on the requirements of a project directly or indirectly. In the development some of the requirement share straightforward and easy for the development team to understand and implement however in some requirements there some issues or aps or complexity which make these requirements risky for the success of the project. When we say risky that means the either project KPI Quality/ Cost or time is going beyond target.

If these risks are not captured and mitigated timely, the project may suffer a lot. These risks must be eliminated and diluted to control the software cost and schedule. Because in the testing phase beginning, we have already implemented the requirements and the cane requests or any other possible factors causing risk is already known to development team and so it is best time for forecasting risks at this stage. This may improvise software productivity and quality while reducing the probability of project disaster. When risks are appropriately mitigated or their contingency plan is available then, it helps to reduce the possibility of software project failure. There are many solutions available to provide the prediction of software risk at different phases in SDLC. however, as there is lot of variation of data and it depends on the project nature as well. So still there is a scope of improvement in the model implementation. Whereas our timings are apt to predict risky requirements. A Risk prediction model includes classification methods that are projected to predict risks on the Software Specifications of the project.

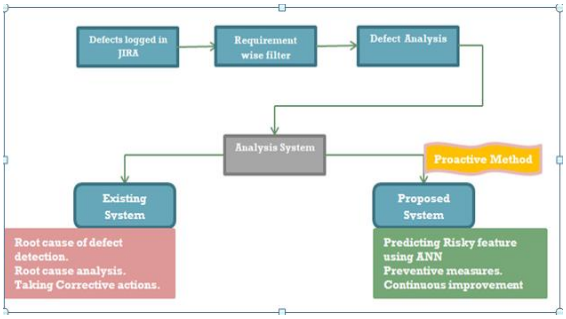


Figure 1: Proactive approach with prediction model

1.1 Solution

As we have been working for this client for more than 5 years and we have delivered several projects for the same base source code. We have the entire data of the requirements along with their defects. We have dataset of around 250 different requirements developed till date. We have taken inputs from SMEs about how to estimate defects in a particular feature. There are few factors identified by SMEs and based the impact of these factors the defects are estimated. If a feature is having higher than the estimated defects, then that requirement is considered risky requirement and accordingly the testing team takes some corrective action by proactively planning of testing that

feature. In this paper we use ANN to analyze Risk and predict the result.

2. THE RESEARCH METHOD

We are going to use Multilayer Perceptron which is basic of the neural networks (NN). Basically, it consists of are an input layer and minimum one hidden layer followed by an output layer. When Input data is given as the input layer in to a NN, the neurons in the layer becomes active layer and the passes the output to the next layer until an output value is produced at each of the output layers neurons. Before passing the data to model the data collection and cleaning is also required which is part of feature engineering.

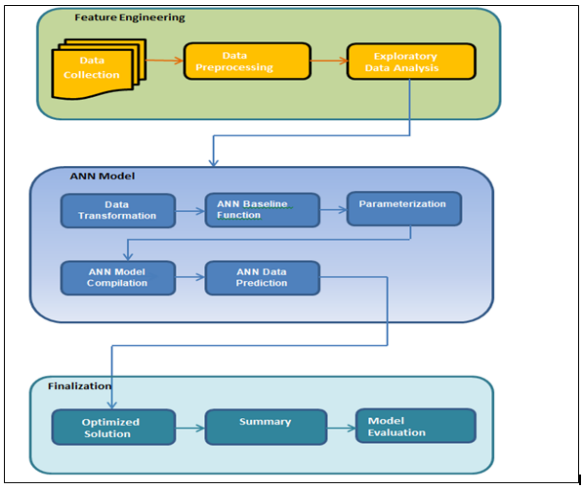


Figure 2: Risky Requirement prediction model

3. EXPLORATORY DATA ANALYSIS

We have taken defects data from JIRA for the last 7 years developed model and we analyze all those defects, during the fixing of defect development team put the feature/ requirement ID in the Jira's so during the retrospection we were able to find out that which requirement results the exceeding defect counts from the defined goal of KPI. Total defect counts were ~7000 defects which were manually analyzed by SMEs.The dataset name is risky_requirements which is having almost 10,000 records. There is total ~250 unique requirements which are related to these defects and based on these defect analysis the different influencing factors were identified.

Requirement_ID	New	Existing	Historical_bug_tendency	Specification_Modify	Complexity	Defect_Steps	Hardware_Dependencies	Testers_Confidence	Coverage_Area	Risk_Rating
REQ_1	0	1	0	1	1	1	1	1	1	Low Risk
REQ_2	1	0	1	0	0	0	0	0	0	High Risk
REQ_3	1	1	0	1	1	0	1	1	1	Medium Risk
REQ_4	0	0	0	0	0	1	1	0	1	Low Risk
REQ_5	1	0	1	0	0	1	0	1	0	Medium Risk
REQ_6	0	1	0	0	1	0	1	0	1	Medium Risk
REQ_7	0	0	0	1	1	1	0	1	0	Low Risk
REQ_8	1	0	1	1	0	0	1	0	0	High Risk

Table 1: Sample data of featuring Data and Target

3.1 Factor Identification

At the very first stage in the risk prediction model, this is conducted using a checklist. SRS requirements having the influencing factors as identified by SMEs in last 7 years historical based data. There are several factors however only some could be applicable in a project so accordingly the factors are marked. The factors were identified after analysis the defects in the previous year's projects.

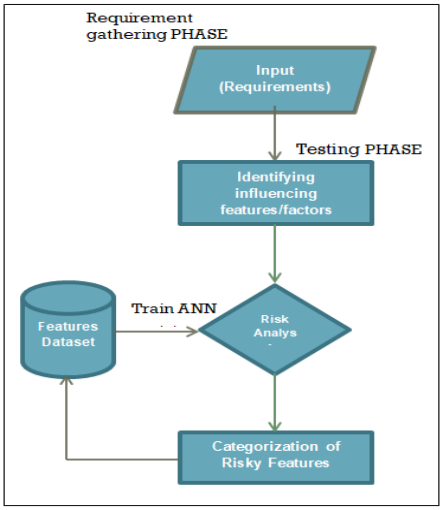


Figure3: Data collection and ANN model interaction

3.2 Different Factors

Factors/Features	Description
New	The requirement is to be developed from scratch.
Existing	Enhancements/Modification in existing requirement.
Historical_bug_tendency	Trend of defects in previous model
Specification_Modify	There was change in requirements.
Complexity	Development SMEs point of view.
Hardware_Dependency	Any requirement depending on hardware
Confidence	Tester SMEs confidence from User point of view.
Coverage_Area	Affected area from coding and relevant features.

Table 2: Factors definition

3.3 Preprocessing

	New	Existing	Historical_bug_tendency	Specification_Modify	Complexity	Defect_Steps	Hardware_Dependencies	Testers_Confidence	Coverage_Area
0	0	1	0	1	1	1	1	1	1
1	1	0	1	0	0	0	0	0	0
2	1	0	0	0	1	0	1	1	1
3	0	0	0	0	0	0	1	0	1
4	1	0	1	1	0	1	0	1	0
...	...	...	...	...	...	...	...	...	...
993	0	0	0	1	0	1	1	0	1

Table 3: Preprocessed Training data

Target label as 3 different Labels High_risky_feature, Medium_risky_feature, Low_risky_feature, so we need to convert these labels in to numeric values so that the machine can understand it. We perform feature encoding as the following table. Features and requirements are same.

3.4 One hot Encoding

Risk_Rating	Risk_Rating_Encode
High_risky_feature	3
Medium_risky_feature	2
Low_risky_feature	1

Table 4: Feature Encoding on Target

In the Multiclass classification, it is required to create target variable in to matrix of the class size. For example in our case it would be [1,0,0],[0,1,0],[0,0,1] corresponding to each of the target value.

4. ANN IMPLEMENTATION

In this implementation we have 9 inputs with 1 hidden layer and 1 output layer with 3 outputs. We will have 18 neurons in first hidden layer and 3 neurons in second layer. This is our basic function which we will be changing in the fine tuning to improve the accuracy. The activation function used for the first hidden layer is **ReLU**; the reason of taking the **ReLU** is that in current neural networks, the usage of rectified linear unit or **ReLU** is recommended.

Function g (z) = max {0, z}

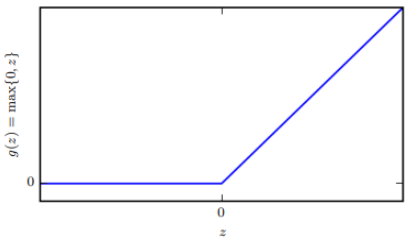


Figure 4: ReLU activation function

In the output layer will be having **Softmax** as activation function the reason of using **Softmax** here is as we have to predict the probability of any of three targets. As it is a multi-class classification so the output would be in terms of probability which is between [0, 1] Softmax for a vector x is calculated as per the formula below

exp(x)/Sum(exp(x)

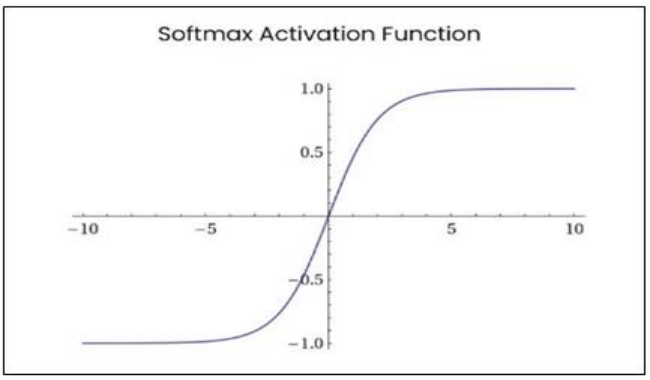


Figure 5: Softmax activation function

4.1 Evaluation Method

The categorical cross entropy is used in multiclass classification. It easily distinguishes between two different probability distribution.

Two parameters were analyzed. Suppose we have probabilities p1, p2, p3, p4……pn of different variables and variables Z1, Z2, ……Zn are unnormalized probability. We define **Softmax** function Qi as below with exponentiation, division and summation operations.

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}.$$

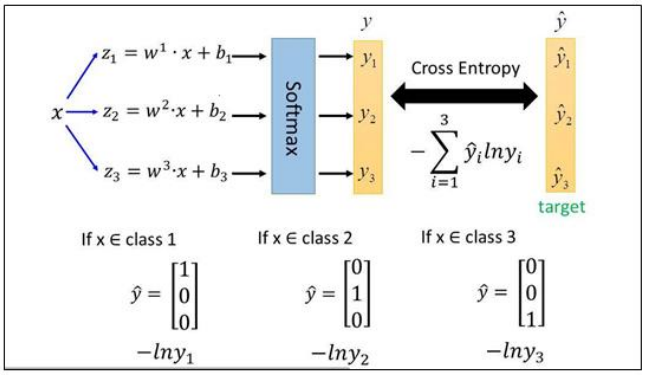


Figure 6: ANN Model

In a situation when all Xi is equal to a constant C. Analytically all the output should be equal to 1/n.

We construct a cross entropy loss J=-∑iPilogQi

Dataset

We have used another evaluation metric **named K-fold cross validation** which creates a process where every sample in the data is included in the dataset in the set at some test. **K represents a number of folds**, usually in ranges of 3 to 10. The data was split into **K** equal folds and

their deviations for each running were analyzed. We found that data set was not imbalanced. Data distribution

Root Cause	Number of Data	Distribution
High_Risk_Feature	351	31.7%
Medium_Risk_Feature	372	33.6%
Low_Risk_Feature	384	34.6%

Table 5: Data Distribution

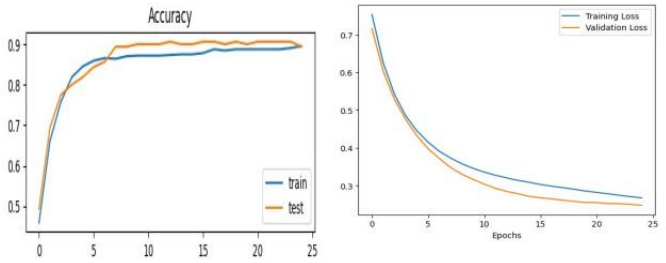
Accuracy & Precision

We have run tree different sets for different combinations of epoch and the batch sizes also with input and output number of neurons. We have compared the accuracy and precision of the model for the probability of the output of the features.

Following result is based on **25 epochs and batch size 32 with 64 input neurons**

ANN Multiclass	
Test Scenario	Score
Accuracy(Test Data)	84.50%
Accuracy(Train Data)	90%
Precision(Test Data)	83.5%
Precision(Train Data)	88%
Recall(Test Data)	87.1%
Recall(Train Data)	89%

Table 6: Observation (Without Tuning)



4.2 Fine Tuning

Following fine tuning approaches were adopted to improve the accuracy of ANN

Number of hidden layers

We can take even a single hidden layer with many neurons this also can help us in extracting pattern in the dataset. It's always better to have multiple layers with neurons than single layer with higher neurons. Since deep learning works on the representing layer mechanism, so first layer captures primitive features and then next layer links factors and then next layer identifies pattern. This also **uses transfer learning**. We can add hidden layer until overfitting does not occur.

Number of neurons/layers to decide

Number of neurons is like in pyramid shape, means from the initial hidden layers to the output layer the neuron count decreases. Like **64->32->16** As number of primitive features is more and then they make complex patterns which are lesser in count and further reduced. However, it is not having any impact on the performance so every layer can have same number of neurons or nodes, however the numbers should be sufficient. In the hidden layer we need to have sufficient number of neurons/nodes.

Batch Size

a) **Batch gradient decent** All rows and then weights are updated slow Training

b) **Stochastic gradient decent** After one row the weights are updated. Very Fast

c) **Mini Batch gradient decent** We decide the batch size, after how many rows the weights would be updated. After how many rows traversal the weights would be updated. Maximum we can have been 8192, however the best is to have smaller batch size like 8, 24 or 32.

Activation function

Sigmoid gives vanishing gradient problem. However, in our case it is unavoidable as we have **multi-class classification problem.**

No of epochs

Early stopping implementation, it automatically stops the model training or epoch's when it finds that there is no more improvement. Using **Keras callback APIs.**

Following result is based **on 200 epochs and batch size 8 with 64 input neurons**

ANN Multiclass	
Test Scenario	Score
Accuracy(Test Data)	94.50%
Accuracy(Train Data)	96%
Precision(Test Data)	92.5%
Precision(Train Data)	93.3%
Recall(Test Data)	92.1%
Recall(Train Data)	94.8%

Table 7: Observation (After Tuning Model)

We have used **ML based XG BOOST** method on same dataset.

Method	Accuracy
ANN(MLP)	94.5%
ML(XG Boost)	87.4%

Table 8: Observation (Comparison)

5. CONCLUSIONS

The presented approach automatically predicts the risky feature for a project. There are few factors identified by SMEs and based on the presence of these factors the defects are estimated. If a feature is having higher than the estimated defects, then that features is considered risky feature and accordingly the testing team takes some corrective action by proactively planning of testing that feature. In this paper we train ANN Model based on this data and analyzes Risk and predicts the result. This information can be used by the development team to choose a proper preventive action. It can also be used by testing team to create a test suite or design for that specific feature. In addition, this approach could be utilized for building benchmarks of specific bug types. The dataset size was almost 1000 records with 3 outputs, in particular, High_risky_feature, Medium_risky_feature, and Low_risky_feature. 64 hidden nodes and batch size of 32 with 25 epochs for the initial condition was used in this method. When evaluated using the categorical cross entropy and k-fold cross validation, this method has an accuracy of 90% and if 200 epochs was used, the accuracy increased to approximately 95%. XGBoost was used to compare with the current method it has an accuracy of 87%. The ANN Multiclass outperforms this method and has an outstanding precision score of 94%. It can be concluded that the ANN method is more stable in terms of how the network is supposed to respond to a particular input. However further exploration could be done with respect to XG BOOST accuracy as compare o MLP.

Threats to validation

During the manual classification, we have noticed recurring fault patterns. Manually categorizing the root cause might be error-prone and the true root cause of the bug can only be determined by the original programmer. We indicated the confidence level for each bug we categorized and excluded bugs with a low confidence level.

REFERENCES

[1] Assim, Marwa, Qasem Obeidat, and Mustafa Hammad. "Software defects prediction using machine learning algorithms." In 2020 International conference on data analytics for business and industry: way towards a sustainable economy (ICDABI), pp. 1-6. IEEE, 2020.

[2] Shah, Mitt, and Nandit Pujara. "A review on software defects prediction methods." arXiv preprint arXiv:2011.00998 (2020)..

[3] Hirsch, Thomas, and Birgit Hofer. "Root cause prediction based on bug reports." In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 171- 176. IEEE, 2020.

[4] Bhanage, Deepali Arun, Ambika Vishal Pawar, and Ketan Kotecha. "It infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine & deep learning approaches and automated tool." IEEE Access 9 (2021): 156392-156421.

[5] Tan, Lin, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. "Bug characteristics in open-source software." Empirical software engineering 19 (2014): 1665- 1705.

[6] Herzig, Kim, Sascha Just, and Andreas Zeller. "It's not a bug, it's a feature: how misclassification impacts bug prediction." In 2013 35th international conference on software engineering (ICSE), pp. 392-401. IEEE, 2013.

[7] Mahmud, Mahmudul Hoque, Md Tanzirul Haque Nayan, Dewan Md Nur Anjum Ashir, and Md Alamgir Kabir. "Software risk prediction: systematic literature review on machine learning techniques." Applied Sciences 12, no. 22 (2022): 11694.

[8] Elzamly, Abdelrafe, and Burairah Hussin. "Classification and identification of risk management techniques for mitigating risks with factor analysis technique in software risk management." Review of Computer Engineering Research 2, no. 1 (2015): 22-38.

[9] Gouthaman, P., and Suresh Sankaranarayanan. "Prediction of Risk Percentage in Software Projects by Training Machine Learning Classifiers." Computers & Electrical Engineering 94 (2021): 107362.