GME: A Mixture of Experts Model for Software Fault Prediction

AMAN OMER, SANTOSH SINGH RATHORE, and SANDEEP KUMAR

I. Mixture of Experts (MoE)

Combining classifiers approaches are promising ones as they aim to improve the performance of the classification models [1], [2], [3], specifically for complex problems, which involve the limited number of training samples, high-dimensional features, and highly overlapped classes [4], [5]. Many previous works have shown that combining the classifiers approach is the most effective when the used underlying experts' (base learners) are negatively correlated or uncorrelated [6]. Therefore, more improved approaches need to be developed, which can produce accurate and negatively correlated experts that can be combined and produced improved performance [7]. The MoE method is a type of combining approach, which uses individual learning techniques as experts who are specialized in its particular subspace of input space [8]. Jacobs et al. originally proposed this technique in 1991 as an "Adaptive mixture of local experts" [9]. It uses the idea of dividing the input space into the number of subspaces, trains experts in each subspace, and combines the learning of experts using a gating function. The work of Jacobs et al. [10] has shown that in comparison to the common combining approaches, which produce unbiased experts with uncorrelated estimated errors, the MoE method produces biased experts with negatively correlated errors. This special feature of MoE compared to common combining approaches helped in achieving improved model performance. The working of the MoE method is based on the divide-and-conquer principle, where the input space is partitioned randomly into a number of subspaces using a special employed function, the experts become specialized on each generated subspace. After that, with the help of a gating function, the weights of the experts are computed dynamically according to the local efficiency of each expert. The experts are performing supervised learning in that their individual outputs are combined with modeling the desired output [8]. There is, however, the experts are also performing self-organized learning. That is, they self-organize to find a good partitioning of the input space so that each expert does well at modeling its own subspace, and as a whole group, they model the input space well. The working overview of the MoE is explained in Figure 1.

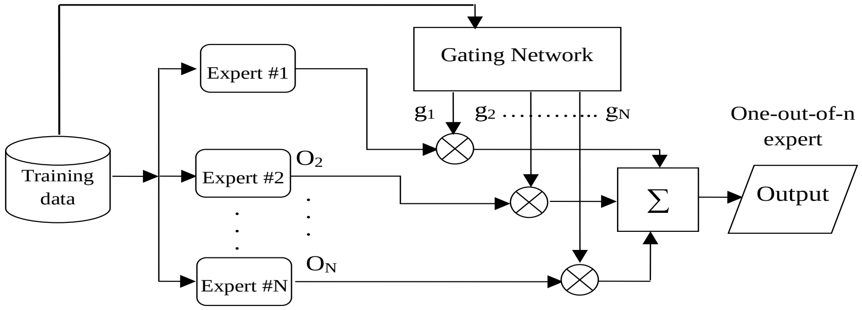

Fig. 1. Overview of the MoE method (based on [11])

The MoE method mainly has three components: (1) several intermediate experts, which are either regression or classifiers depending upon the given problem domain, (2) a gating function that partitions the input space into a number of subspaces using soft partition boundaries, and (3) and a probabilistic model to combine the experts and the gate. The final prediction output of the MoE method is a weighted sum of experts, where weights are dynamically updated via an input-dependent gating function. These properties of MoE help in representing non-stationary or piecewise continuous data in a prediction process and identification of the nonlinearities in a classification process. One of the main advantages of MoE is that it is flexible to combine a variety of different learning techniques. The simulation of the MoE model as presented by R. Jacobs [10] showed that MoE leads to the negatively correlated experts, which is the foremost requirement of any combining approach. Further, the author stated that the negative correlations come because the MoE method adaptively partitions the input space into regions so that the target function has different properties in each region. The experts learn from these different regions and different experts provided different "basis" functions.

The modeling of the MoE method is done by placing the networks in different structural arrangements. A probabilistic interpretation exists, which provides a corresponding likelihood function that characterizes the model performance [6]. Learning occurs by maximizing the likelihood function. The presented MoE method consists of two types of networks, a gating network and a number of expert networks. The gating network is used to model the input-dependent multinomial distribution to select a rule. The output of gating network are computed using a function [12]. The expert networks are used to model the input-dependent statistical models associated with the different rules. The output of expert network is a linear function of its input. The final output of the MoE method is given by a combination of the experts' outputs. During the learning, the parameters of the gating and expert networks are optimized to maximize the likelihood function.

II. Performance Evaluation Measures and Statistical Tests

TABLE I
PERFORMANCE EVALUATION MEASURES

Measures	**Description**	**Formula**
Precision	It denotes number of correctly classified faulty data points among the total number of data points classified as faulty.	Precision = TP/(TP+FP)
Recall	It indicates the number of correctly classified faulty data points amongst the total number of data points, which are faulty.	Recall = TP/(TP+FN)
F1-score	It is the harmonic mean of precision and recall values.	F1-score = (2*precision*recall)/(Precision+Recall)
Probability of false alarm (PF)	It is also known as false positive ratio. It is calculated as the ratio between the number of false positives and the total number of actual negative events (false positive and true negative). A value close to 0 is preferred for the PF measure.	PF = FP/(FP+TN)
G-means	It stands for geometric means. G-mean 1 is calculated as the square root of the precision and recall. G-mean 2 is calculated as the square root of the product of recall and specificity.	G-mean 1 = $\sqrt{Precision * Recall}$ G-mean 2 = $\sqrt{Specificity * Recall}$ Specificity (True negative rate) = TN/(FP+TN).
Matthews correlation coefficient (MCC)	MCC calculates the correlation coefficient between actual values of the class label and predicted values of the class label. The value of MCC varies between -1 and $+1$. -1 indicates the perfect disagreement, 1 indicates the perfect agreement, and 0 indicates the random prediction with respect to the actual label values.	MCC = (TP*TN-FP*FN) / $\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}$
Area under ROC curve (AUC)	It calculates the area under the receiver operating characteristic curve. It is a graphical plot that depicts the diagnostic capabilities of a prediction model under different threshold values. It plots the true positive rate in the y-axis and false positive rate in the x-axis. Area under the curve shows the probability that a classifier will classify a randomly chosen positive module higher than a randomly chosen negative module.	
Statistical test	We perform Friedman's test and the Wilcoxon signed rank sum test to evaluate significance of performance difference between the presented GME methods and other used techniques. Both of these tests are non-parametric statistical hypothesis tests [13]. Additionally, we have reported effect size value using Pearson effect *r* measure [14]. The effect size shows the magnitude of performance different among the groups. r = z/$\sqrt{(2n)}$, Where 2n = the number of observations, including the cases where the difference is 0 and z is the z-score value. z = $\|U - \mu\|$-0.5/σ According to Cohen (1988, 1992) [15], the effect size is low if the value of *r* varies around 0.1, medium if *r* varies around 0.3, and large if *r* varies more than 0.5.	

** TP= True Positive, TN = True negative, FP = False Positive, and FN = False Negative. Precision, recall, f1-score, G-means, and AUC measures can take value between 0 and 1. Where, 0 shows the worst performance and 1 shows the best performance.*

III. Used learning techniques

TABLE II
Description of the used learning techniques

Techniques	Description
Classification techniques	
Decision Tree (DT)	The decision tree is a classification technique, which builds a tree type of structure for classification [16]. Each feature (attribute) of the dataset is represented by an internal node in the tree, and class labels are represented by leaf nodes in the tree. Tree construction is started by selecting the best attribute of the dataset as the root node of the tree. Further, training data is split into subsets, where each subset is made in such a way that each subset contains data with the same value for an attribute. These two steps are repeated on each subset until all branches of the tree find leaf nodes. Tree leaves are named by class labels [17].
Multilayer perceptron (MLP)	MLP is a feed-forward neural network, which uses a series of interconnected processing units knows as layers∼\cite{thota2013optimum}. The learning of MLP involves the development of a representation that maps the knowledge of the input domain into the output domain. In each iteration, input data examples (training data) are repeatedly fed into the neural network, and the output obtained is compared with the desired output, and error is calculated. Based on the error value, hidden layer weights are updated and re-fed the network [18].
Ensemble techniques	
Bagging	It stands for {B}ootstrap {agg}regat{ing} (Bagging) [19]. The working of bagging is based on the idea of generating different intermediate learning models by using different subsets of training datasets. It uses a learning technique as a base learner and generates different trained versions of it by training it on different bootstrapped subsets of the training dataset. The final prediction is performed by combining the decisions of all generated learning models.
Boosting	The learning of the boosting method is based on incremental learning. Here, a set of intermediate models is generated in different iterations, and incrementally, these intermediate models are refined by updating examples (observations) weights in every iteration for the learning technique [20]. In each iteration, a distinct model is generated, which is now capable of classifying the examples that were misclassified earlier. The final prediction is performed by combining the outputs of all intermediate generated models. In this work, we use AdaBoost, which is an adaptive version of the classic boosting method [21].
Stacking	It is a meta-learner ensemble method that involves the training of a learning technique to combine the outputs of several other trained learning techniques. The model building in the stacking method is a two steps process. In the first step, a set of learning algorithms are trained on the training dataset. In the second step, a combiner learning technique is trained to make the final prediction using all the predictions of the other techniques as additional inputs [22].
Other used techniques	
Classic-ME (Hierarchical Mixtures of Experts)	Jordan and Jacobs proposed a hierarchical mixture of experts (HME) method, which uses the EM algorithm for adjusting the parameters of the model [23]. The proposed HME method is based on supervised learning that divides the input space into a nested set of regions and having soft boundaries. The HME architecture uses a tree structure, which acts as gating networks. These networks take a vector having features and the dependent variable information as input and produce scalar outputs that are the partitions of input space into multiple regions. The expert networks are at the leaves of the tree that produces an output vector. These output vectors are then going at the level up to the tree and multiplied by the gating network outputs. The final output is summed at the non-terminals.

IV. Used software metrics

Table III provides descriptions of the used software metrics available in different software fault datasets used for experimentation in this work. The detailed description of these metrics can be found in [24], [25], [26], [27]. The used object-oriented metrics are corresponding to the CK metrics suite [28], Henderson-Sellers metrics suite [29], Tang metrics suite [30], and Martin metrics suite [31]. Eclipse component such as Lucene, JDT_Core, PDE_UI, Equinox, and Mylyn contained different software metrics proposed by D'Ambros et al. [25]. Different Eclipse versions (2.0, 2.1, and 3.0) contained complexity, source code, and code churn metrics proposed by Zimmermann et al. [26]. Other used datasets are from the JIRA defect data repository and contained different code, process, and ownership metrics proposed by Yatish et al. [27].

TABLE III
Description of the software metrics [24], [25], [26], [27]

“Datasets	Metric	Description
Ant, Camel, Ivy, Jedit, Poi, Prop, Synapse, velocity, Xalan, Xerces	WMC	Number of methods defined in a class
	CBO	Count the number of classes coupled to class
	RFC	Count the number of distinct methods invoked by a class in response to a received message
	DIT	Depth of a class within the class hierarchy from
	NOC	Number of immediate descendants of a class
	IC	Count the number of coupled ancestor classes of a class
	CBM	Count the number of added or redefined methods those are coupled with the inherited methods
	CA	Count the number of dependent classes for a given class
	CE	Count the number of classes to which a class
	MFA	Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class
	LCOM	Subtraction of methods pairs that do not share a field to the methods pairs that do
	LCOM3	Counts the number of connected components in a method graph
	CAM	Computes the cohesion among methods of a class based on the parameters' list
	MOA	Count the number of data members declared as class type
	NPM	Number of public methods defined in a class
	DAM	Computes the ratio of private attributes in a class
	AMC	Shows the average method size of a class
	LOC	Counts the total number of lines of code of a class
	CC	Counts the number of logically independent paths
Lucene, Mylyn, PDE_UI, Equinox	pre (pre-release faults), NOM_sum, NSM_avg, ArrayCreation, ArrayInitializer, ArrayType, CharacterLiteral, ConditionalExpression, ContinueStatement, DoStatement, FieldAccess, Javadoc, LabeledStatement, ParenthesizedExpression, PrefixExpression, QualifiedName, ReturnStatement, SuperMethodInvocation, SwitchStatement, ThisExpression, ThrowStatement	
Eclipse 2.0, Eclipse 2.1, and Eclipse 3.0	Fout	Number of method calls (fan out) in a method (avg, max, total)
	MLOC	Method lines of code in a method (avg, max, total)
	NBD	Nested block depth in a method (avg, max, total)
	PAR	Number of parameters in a method (avg, max, total)
	VG	McCabe cyclomatic complexity in a method (avg, max, total)
	NOF	Number of fields in class (avg, max, total)
	NOM	Number of methods in a class (avg, max, total)
	NSF	Number of static fields in a class (avg, max, total)
	NSM	Number of static methods in a class (avg, max, total)
	ACD	Number of anonymous type declarations in a file (avg, max, total)
	NOI	Number of interfaces in a file (avg, max, total)
	NOT	Number of classes in a file (avg, max, total)
	TLOC	Total lines of code in a file (avg, max, total)
Activemq-5.0.0, derby-10.5.1.1, groovy-1 6 BETA 1, hbase-0.94.0, hive-0.9.0, jruby-1.1, wicket-1.3.0-beta2	File-level	AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountDeclClass, CountDeclClassMethod, CountDeclClassVariable, CountDeclFunction, CountDeclInstanceMethod, CountDeclInstanceVariable, CountDeclMethod, CountDeclMethodDefault, CountDeclMethodPrivate, CountDeclMethodProtected,CountDeclMethodPublic, CountLine, CountLineBlank, CountLineCode, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountSemicolon, CountStmt, CountStmtDecl, CountStmtExe, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, RatioCommentToCode, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential
	Class-level	CountClassBase, CountClassCoupled, CountClassDerived, MaxInheritanceTree, PercentLackOfCohesion
	Method-level	CountInput (Min, Mean, Max), CountOutput (Min, Mean, Max), CountPath (Min, Mean, Max), MaxNesting (Min, Mean, Max)”

V. Hyperparameter tuning of used techniques

SMOTE: SMOTE method of class *over_sampling* in package *imblearn* is used for sampling with 5 neighbors to generate synthetic samples.

Cross-Validation: A 10-fold cross-validation is implemented using the *K-Fold* method of *sklearn* library with shuffle set as true, which means data will be shuffled before splitting into batches. The process continues for the ten iterations, and the results of each iteration are averaged over the iterations.

Decision Tree: Implemented function *DecisionTreeClassifier* of library *sklearn* is used with parameters, splitting criteria as gini index and tree will be extended up to the height until all leaves are pure.

Multilayer Perceptron: Implemented method *MLPClassifier* of library *sklearn* is used with initial parameters.

Ensemble methods (Bagging, Boosting, and Stacking): For the implementation of bagging method, *BaggingClassifier* function of *sklearn* library is applied. The number of estimators (base learners) is set as 10. The maximum number of samples and features to be drawn from original data is set as 0.7. That shows 70% of total instances and 70% of all features will be used to construct the subset for every model. Bootstrap is set to be true for both samples and features that indicate a data point or feature will be included in more than one training subsets. For the implementation of boosting method, *AdaBoostClassifier* function of *sklearn* library is used. The number of estimators is set to 10, and *SAMME.R* algorithm is used. *Base_estimator* is set to DT as MLP does not support sample weighting for the AdaBoost method. For the stacking method, *StackingClassifier* of sklearn is used. The used estimators are DT and MLP, and for final_estimator *DecisionTreeClassifier* and *MLPClassifier* are used.

Classic-ME: Softmax classifier is used as an expert to the HME model, and also softmax classifier is used as the gating function. We have used python implementation of the HME model available on the GitHub repository[1]. Level's value is set to 6 and the branching value is set to 2.

For all the presented methods and techniques, including GME-DT, GME-MLP, bagging, boosting, stacking, GMENB, GME-LR, classic-ME, and individual techniques, we used the same 10-fold cross-validation.

Gaussian model with experts (GME): The number of experts or estimators is set to 10. The purpose is to evaluate the performance of GME and compare it with individual techniques and ensemble methods. Therefore, the same number of base models are used in ensemble methods and GME methods. Data selection threshold (DS) and Label Prediction threshold (LP) are set as 0.2 and 0.6, respectively. These values are concluded after performing a preliminary experimental study. First, ten datasets are selected at random, and among them, on five datasets, DT is used as the base model, and for remaining MLP is used as the base model. Second, the GME model with the number of experts as 10 for different values of DS and LP is applied, with 10-fold cross-validation. Third, the accuracy score of each dataset for every combination of DS and LP values is collected. Lastly, calculate the standard deviation (SD) of collected accuracy of 10 datasets for every pair of DS and LP values. SD is low for LP=0.6 for almost every value of DS, and it is minimum when DS=0.2.

Number of experts in the GME: The number of experts in GME is set to 10 (k=10). We have performed an experiment where the value of the number of experts was varied from 2 to 20 and recorded the performance of GME for different performance measures given in Section IV.D. Based on the observations and the experimental analysis, we select the number of experts value as 10.

References

[1] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. Combined classifier for cross-project defect prediction: an extended empirical study. *Frontiers of Computer Science*, 12(2):280–296, 2018.

[2] Chubato Wondaferaw Yohannese, Tianrui Li, Macmillan Simfukwe, and Faisal Khurshid. Ensembles based combined learning for improved software fault prediction: A comparative study. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE, 2017.

[3] Fatih Yucalar, Akin Ozcift, Emin Borandag, and Deniz Kilinc. Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability. *Engineering Science and Technology, an International Journal*, 23(4):938–950, 2020.

[4] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. Bpdet: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications*, 144:113085, 2020.

[5] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. *Information and Software Technology*, 87:206–220, 2017.

[6] Isobel Claire Gormley and Sylvia Frühwirth-Schnatter. Mixture of experts models. *Handbook of Mixture Analysis*, pages 271–307, 2019.

[7] Pierre Geurts. Bias vs variance decomposition for regression and classification. In *Data mining and knowledge discovery handbook*, pages 733–746. Springer, 2009.

[8] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2):275–293, 2014.

[9] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[10] Robert A Jacobs. Bias/variance analyses of mixtures-of-experts architectures. *Neural computation*, 9(2):369–383, 1997.

[1] Hierarchical Mixture of Experts, https://github.com/AmazaspShumik/Mixture-Models/tree/master/Hierarchical%20Mixture%20of%20Experts

[11] Saeed Reza Kheradpisheh, Fatemeh Sharifizadeh, Abbas Nowzari-Dalini, Mohammad Ganjtabesh, and Reza Ebrahimpour. Mixture of feature specified experts. *Information Fusion*, 20:242–251, 2014.

[12] Xin Wang, Fisher Yu, Lisa Dunlap, Yi-An Ma, Ruth Wang, Azalia Mirhoseini, Trevor Darrell, and Joseph E Gonzalez. Deep mixture of experts via shallow embedding. In *Uncertainty in Artificial Intelligence*, pages 552–562. PMLR, 2020.

[13] Donald W Zimmerman and Bruno D Zumbo. Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks. *The Journal of Experimental Education*, 62(1):75–86, 1993.

[14] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *annual meeting of the Florida Association of Institutional Research*, pages 1–33, 2006.

[15] SA McLeod. What does effect size tell you. *Simply psychology*, 2019.

[16] Ronny Kohavi and J Ross Quinlan. Data mining tasks and methods: Classification: decision-tree discovery. In *Handbook of data mining and knowledge discovery*, pages 267–276. Oxford University Press, Inc., 2002.

[17] Santosh S Rathore and Sandeep Kumar. A decision tree logic based recommendation system to select software fault prediction techniques. *Computing*, 99(3):255–285, 2017.

[18] Momotaz Begum and Tadashi Dohi. Optimal stopping time of software system test via artificial neural network with fault count data. *Journal of Quality in Maintenance Engineering*, 24(1):22–36, 2018.

[19] September Leo Breiman. Bagging predictors. Technical report, Technical Report, 1994.

[20] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[21] Eric Emer. Boosting (adaboost algorithm). *Consultado el*, 26, 2017.

[22] Mete Ozay and Fatos T Yarman Vural. A new fuzzy stacked generalization technique and analysis of its performance. *arXiv preprint arXiv:1204.0171*, 2012.

[23] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[24] Marian Jureczko and Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 9. ACM, 2010.

[25] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41. IEEE, 2010.

[26] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pages 9–9. IEEE, 2007.

[27] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. Mining software defects: Should we consider affected releases? In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 654–665. IEEE, 2019.

[28] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[29] Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[30] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*, pages 242–249. IEEE, 1999.

[31] Robert Martin. Oo design quality metrics. *An analysis of dependencies*, 12(1):151–170, 1994.