Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

vol. 15, no. 10, October 2019, pp. 2701-2708
DOI: 10.23940/ijpe.19.10.p16.27012708

# Active Learning using Uncertainty Sampling and Query-by-Committee for Software Defect Prediction

Yubin Qu[a], Xiang Chen[b,*], Ruijie Chen[c], Xiaolin Ju[b], and Jiangfeng Guo[a]

[a]*Jiangsu College of Engineering and Technology, Nantong, 226001, China*
[b]*Nantong University, Nantong, 226019, China*
[c]*Nanjing Foreign Language School, Nanjing, 210008, China*

**Abstract**

In the process of software defect prediction dataset construction, there are problems such as high labeling costs. Active learning can reduce labeling costs when using uncertainty sampling. Samples with the most uncertainty will be labeled, but samples with the highest certainty will always be discarded. According to cognitive theory, easy samples can promote the performance of the model. Therefore, a hybrid active learning query strategy is proposed. For the sample with lowest information entropy, query-by-committee will analyze it again using vote entropy. Empirical studies show that the proposed HIVE approach outperforms several state-of-the-art active learning approaches.

*Keywords*: active learning; vote entropy; software defect prediction; uncertainty sampling

(Submitted on August 21, 2019; Revised on September 23, 2019; Accepted on October 20, 2019)



## 1. Introduction

Software defect modules can cause operational failures in the production process of the enterprise, causing significant losses to the company and reducing satisfaction. The software defect prediction model is used to discover software defect modules early in the software development phase. Traditional models include supervised models and unsupervised models [1]. A supervised software defect prediction model relies on a large number of labeled samples. The defect module labeling work has high time consumption. For software module labeling, more professional software testers are needed. Therefore, the establishment of a software defect prediction model requires a large amount of time and investment. More manpower has increased the cost of software development. The trade-off between the accuracy of the prediction model and the cost of creating the training data is always present. Achieving an accurate model typically requires as large a training dataset as possible. Active learning provides multiple query strategies for solving sample labeling problems, enabling enterprises to actively select a certain sample for labeling when facing massive labeling modules [2]. A software defect prediction model can be constructed quickly. Some scholars have proposed active learning strategies that consider informative and representative examples [3-4]. There have been many studies focusing on active learning in the field of within-project defect prediction [5-7], Lu et al. [8-9] tried to apply active learning to cross-project defect prediction from the perspective of dimensional reduction. Sample labeling is a time-consuming and tedious work, so algorithms based on the cost-sensitive method can better meet practical needs. At present, some scholars have tried to complete the sample selection of active learning from the point of view of cost sensitivity. Wang et al. [10] proposed to incorporate self-paced learning into the sample query strategy and to treat examples with low information entropy as easy-learning courses and pseudo-label unlabeled samples using a trained model in the field of image recognition. However, this method may have used complicated learning models. The sample with highest information entropy calculated by the trained model was labeled manually, and the low information entropy samples was pseudo-labeled. As far as we know, there is currently no algorithm that uses a hybrid active learning query strategy to solve the label problem in software defect prediction.

We propose a hybrid active learning query strategy in this paper. This hybrid query strategy can effectively reduce the

* Corresponding author.
*E-mail address*: xchencs@ntu.edu.cn

cost of software defect module labeling. The method first uses an active learning query strategy based on information entropy. After calculating the information entropy for unlabeled samples, the sample with the highest uncertainty is manually labeled by the domain expert or software test engineer. After the labeled sample is added to the training dataset, query-by-committee using the vote entropy method is used to pseudo-label the examples with the lowest information entropy [11]. If the votes are consistent, the samples are pseudo-labeled with the agreed label, and the labeled sample is added to the training dataset. If the votes are inconsistent, the sample is discarded.

We have conducted experiments in public AEEEM datasets, compared to random, DWUS, query-by-committee, QUIRE, and svmMarginRepresentativeCluserKMeans.

The main contributions of this paper are the following:

(1) A hybrid active learning query strategy is proposed using information entropy and vote entropy.

(2) Compared with other traditional methods, the hybrid active learning query strategy can achieve higher defect prediction performance on public datasets, thereby reducing manual labeling costs.

The rest of this paper is organized as follows: Section 2 reviews the related work on software defect prediction and active learning. Section 3 presents the proposed algorithm in detail. The experimental setup is reported in Section 4. The experimental results are reported in Section 5. Finally, Section 6 concludes the work with issues to be addressed in the future.

## 2. Related Work

### *2.1. Software Defect Prediction*

If the software project has rich historical data, a within-project defect prediction model can be built using a supervised machine learning model, and then the probability of software module defects can be evaluated or the number of defects in a module can be calculated [8]. Ghotra et al. [12] evaluated the effects of different classifiers on the performance of software defect prediction models from the perspective of empirical research. Xu et al. [13] studied the effects of different feature selection methods on software defect prediction from the perspective of empirical research. In the actual software development process, if the software project is a completely new project, or the training data of this project is relatively small [1], then the transfer learning method can be used (cross-project defect prediction). If the new project does not have more data, and the labeling cost is higher, a training dataset can be constructed by means of technology such as active learning.

### *2.2. Active Learning*

Active learning is a sub-domain of machine learning. The key assumption is that if a learning algorithm actively chooses the sample for training, it can obtain a better training model with fewer datasets [2]. The sample selection can be divided into stream-based selective sampling and pool-based sampling [14] depending on how samples are chosen.

Active learning provides a variety of learning strategies to select unlabeled samples. The most widely used and perhaps the simplest active learning query strategy is uncertainty sampling. The strategy is based on the probability calculated by the classification model, and the calculation method of the information entropy is used to calculate the uncertainty of the current sample. The higher information entropy, the higher uncertainty. For a two-class model, when the probability is 0.5, the information entropy is largest [2].

Another common active learning strategy is query-by-committee, which creates a query model set based on the classification model $C = \{\theta^1, \cdots, \theta^m\}$. Each classifier creates a model based on the labeled training dataset and predicts each unlabeled sample. The sample with the highest degree of inconsistency will be selected and labeled. This strategy is based on the strategy of minimizing the version space. Each classification model provides different classification spaces and finds the sample to be labeled with the largest amount of information according to the degree of inconsistency.

In previous studies [5-9], the active learning query strategy was used to select high-value samples from software defect prediction datasets. At the same time, the use of dimension reduction, feature selection, and other machine learning methods were combined to improve the performance of software defect prediction. The query strategy using uncertainty sampling does not focus on examples with low information entropy. An example with low information entropy is often discarded during the active learning of a query. We propose a hybrid query strategy using information entropy and vote entropy. This

method firstly uses the sample with the highest information entropy and the sample with the lowest information entropy to create training datasets.

## 3. Hybrid Query Strategy using Information Entropy and Vote Entropy

This section will give the implementation details of a hybrid query strategy using information entropy and vote entropy (short for *HIVE*). First, the uncertain information entropy and vote entropy used in *HIVE* are defined, and then the algorithm is described in detail.

### *3.1. HIVE Query Strategy*

The *HIVE* query Strategy will be applied in open *AEEEM* datasets in software defect prediction. By incremental sample select, the classifier can achieve better classification performance when the same data is annotated. The model integrates the idea of collaborative training into the active learning query strategy and completes model construction by minimizing the labelling cost.

We start with a synthesized example that illustrates the importance of querying an instance with highest uncertainty and another instance with highest certainty.

The motivation behind uncertainty sampling is to find some unlabeled examples near the decision hyperplane and assume that these examples have the maximum uncertainty. Figure 1 shows two unlabeled examples, A and B, which have the highest uncertainty at the $i^{th}$ learning iteration, while example C has the highest certainty at the $i^{th}$ learning iteration. Regardless of whether A or B is selected, either is the right choice, even though it seems that B is an outlier in the current data distribution. In another iteration, however, example B may be an outlier. It is uncertain because the labeled data distribution is uncertain. Meanwhile, at the ith learning iteration, example C is almost certain because example C is fastest to hyperplane. Example C maybe added to the labeled dataset for this certainty. The philosophy under this learning paradigm is to simulate the learning principle of humans, which generally starts by learning easier content of a learning task and then gradually takes more complex examples into the training dataset [15-16].

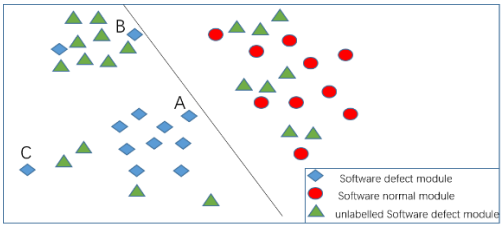



Figure 1. An example of three points A, B, and C, where A and B have the highest uncertainty and C has with highest certainty at the ith learning iteration

The *HIVE* strategy chooses the sample with the highest information entropy from the unlabeled data, and then this sample will be labeled by the oracle. The sample with lowest information entropy will be justified by query-by-committee using vote entropy. When the vote entropy of this sample is less than the threshold (the current empirical threshold is set to 0.5), the sample is added to the training dataset, and the sample is pseudo-labeled using the prediction label.

$$x_{u,max} = argmax\left(-\sum_{i} P_\theta(y_i|x)\, log\, P_\theta(y_i|x)\right) \tag{1}$$

Define a dataset $D_l = \{x_i\}_{i=1}^{l}$ containing $l$ labeled samples, and define a dataset $D_u = \{x_i\}_{i=1}^{u}$ containing $u$ unlabeled samples. $x_u^{(i)}$ represents the $i^{th}$ unlabeled sample ($i$ = 1, 2, ⋯, $u$), $x_{u,\max}$ represents the sample with the largest information entropy in the unlabeled dataset obtained according to Equation (1), $P_\theta(y_i|x)$ represents the predicted probability value of $x_{u,\max}$ belonging to the category $y_i$ based on the data distribution of the labeled dataset, and $\text{argmax}(-\sum_i P_\theta(y_i|x)\, log\, P_\theta(y_i|x))$ indicates that the sample with the largest value in the unlabeled dataset is selected.

$$x_{ve}^{*} = \underset{x_{u,min}}{argmax} -\sum_{i} \frac{V(y_i)}{C} \times log \frac{V(y_i)}{C} \tag{2}$$

$x_{u,\min}$ represents the sample with the smallest information entropy in the unlabeled dataset calculated according to Equation (1), and $C$ represents the number of classifiers of the query committee. The classification committee defines $C = \{\theta^1, \cdots, \theta^m\}$. The classifier members of the classification committee represent different classification strategies and are able to predict one label for the unlabeled sample. $y_i$ again ranges over all possible labelings, $V(y_i)$ is the number of "votes" that a label receives among the committee members' predictions, and $C$ is the committee size.

### *3.2. HIVE Algorithm*

A two stage strategy is used to solve the model. The optimization process is as follows:

(1) System initialization

Before the system starts running, a part of the sample is taken from the sample pool and manually labeled by the oracle. The collection is recorded as $D_l$. Choosing samples is random from the sample pool, and the $D_l$ dataset can be used as the first training dataset. The classification model $M$ can be trained using $D_l$.

(2) Actively select samples from the unlabeled sample pool

The information entropy of each unlabeled sample according to Equation (1) is computed using classification model $M$, and the sample with the largest information entropy $x_{u,\max}$ is manually labeled by the oracle. $x_{u,\max}$ is added to the labeled dataset $D_l$ and removed from the unlabeled dataset $D_u$.

(3) The pseudo-label of the highest degree of certainty

The vote entropy according to Equation (2) is calculated for the label with the lowest information entropy. The vote entropy is compared to a threshold. When the vote entropy is less than the threshold, the sample is pseudo-labeled using the prediction label, and this sample is added to the dataset $D_l$.

(4) Update classification model $M$

The classification model $M$ is trained again using the dataset $D_l$ and then looped until the termination condition is met. The entire algorithm process can be summarized as Algorithm 1.

Algorithm 1: The *HIVE* Algorithm

Input:
labeled dataset $D_l = \{x_1, x_2, \cdots, x_l\}$, unlabeled dataset $D_u = \{x_{l+1}, x_{l+2}, \cdots, x_{l+u}\}$, $y \in \{0,1\}$:, max iterations $U_{\max}$, *Query-by-committee* $C = \{C_1, C_2, \cdots, C_U\}$
Output:
$\{AUC_1, AUC_2, \cdots, AUC_u\}$
Initialize classification model *M*
while itor < $U_{\max}$ || not convergence do
  for *i* =1 to *len*($D_u$) do:
    calculate $P(x^i)$
    calculate information entropy according(1), get
    $x_{u,\max}$, $x_{u,min}$, according (2), calculate $x_{ve}^*$.
    If $x_{ve}^*$ < threshold
      $D_l = D_l \cup x^i$
      $D_u = D_u \backslash x^i$
  end for
end while
return $\{AUC_1, AUC_2, \cdots, AUC_{U\max}\}$

## 4. Experiments

### *4.1. Experimental Dataset*

We will analyze and evaluate the impact of the *HIVE* query strategy on the public dataset *AEEEM* in software defect prediction. The *AEEEM* dataset [17] will be used to evaluate different learning strategies. This dataset is widely used in the field of software defect prediction [18-20]. It was first proposed as a benchmark dataset for performance comparison. The

dataset provides 61 indicators, including software development process metrics. In this experiment, 61 indicators are used for classifier modelling. The summary information of the *AEEEM* dataset is shown in Table 1. The specific details can be found on the corresponding website.

Table 1. AEEEM dataset

| Project | Modules | Defects modules | Defects percentage |
|---|---|---|---|
| Eclipse JDT Core | 997 | 206 | 20.7% |
| Equinox | 324 | 129 | 39.8% |
| Apache Lucene | 691 | 64 | 9.3% |
| Mylyn | 1862 | 245 | 13.2% |
| Eclipse PED UI | 1497 | 209 | 14.0% |

### *4.2. Experiment Setup*

The experiment uses $20 \times 2$ fold cross-validation to conduct experiments. Each experiment performs random stratified sampling on the data. Half of the data is used as training data, and the other half of the data is used as test data to prevent data overlap between the training data and the test data [21]. A certain proportion of data is taken out from the training data for manual labelling. In this experiment, the proportion of the initial marker data is 30%. The initial training dataset is used to train the classification model, and the remaining 70% of data is used as the unlabeled sample pool. Support vector machine is used as the classification model using the RBF kernel implemented by libsvm. The RandomForestClassifier implemented by sklearn is used in the *HIVE* query strategy to construct the query committee. All the classifiers use the default parameters. In the *HIVE* strategy, the unlabeled data is iterated, and each of the iterations selects a sample with the highest degree of uncertainty to be manually labeled. At the same time, the sample with the lowest degree of certainty is selected and further judged by vote entropy, and the threshold of the judgment is set to an empirical value of 0.05. We evaluate the classification model by its performance on the holdout test data. The area under the ROC curve (*AUC*) is used for evaluation metrics.

We compare *HIVE* with the following five baseline approaches: (1) random: randomly select a query instance, (2) DWUS: select instances using information entropy and the K-medoid algorithm for representative instances, (3) quire: select instances with high information and representation, (4) committee: select instances using query-by-committee, and (5) svmMarginRepresentativeCluserKMeans: select instances with the lowest distance to the decision hyperplane and representation using k-means.

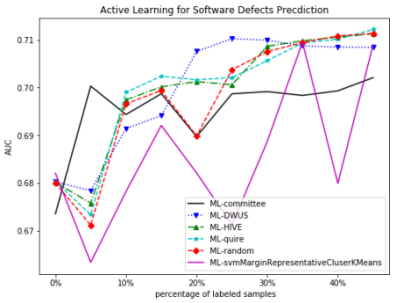
### *4.3. Results*

Figure 2 shows the classification AUC of different active learning approaches with varied numbers of queries. Only 50% of unlabeled data is displayed because, when using the *HIVE* strategy, there is no unlabeled data left when querying to almost 50%.

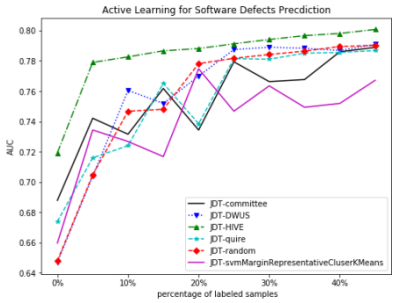
Table 2 shows the change in the AUC value when the labeled sample ratios are 10%, 20%, 30%, 40%, and 50%. When the proportion of the labeled samples is greater than 50%, the *HIVE* active learning strategy has completed the marking of all the samples using the pseudo-marking method, and there are no unlabeled samples. Statistical analysis was performed using paired *t*-tests with 95% confidence, and the best performing model was noted. Table 3 shows the win/tie/loss analysis of different learning strategies with different scales. The statistics of the *HIVE* learning strategy are compared with other strategies.

First, we can observe that in *AEEEM* datasets as the unlabeled sample is gradually reduced, the AUC performance can basically keep rising. This trend indicates the effectiveness of the active learning strategy.

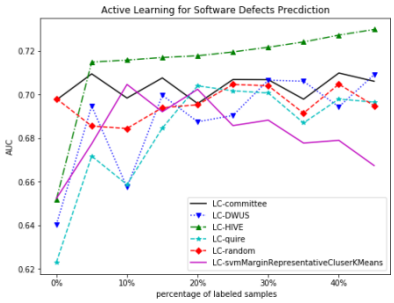
The query-by-committee strategy performance changes are relatively large, especially in the EQ dataset. This may be caused by the data distribution of the EQ dataset. In the JDT dataset, the query-by-committee strategy performance vibrates relatively large. The quire strategy is robust in *AEEEM* datasets, and it is a state-of-art strategy. The main problem is that this strategy has high time consumption. The DWUS strategy works well in datasets. Because DWUS uses uncertainty and the k-means algorithm, it also has high time consumption. Finally, we observe that the *HIVE* strategy is able to outperform the baseline methods significantly in Tables 2 and 3. At the same time, it uses the least labeled samples to achieve the same performance. The *HIVE* strategy's computational complexity is the lowest because it only uses information entropy. We attribute the success of *HIVE* to the principle of choosing unlabeled instances that have the highest uncertainty and the highest certainty.



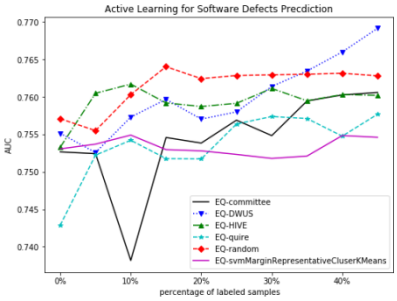

(a) AUC compare on ML dataset



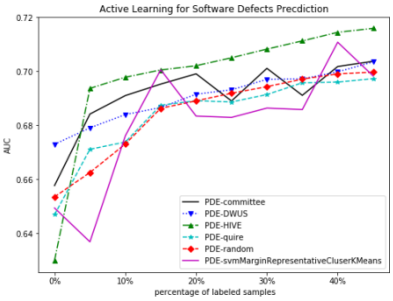

(b) AUC compare on JDT dataset




(c) AUC compare on LC dataset




(d) AUC compare on EQ dataset




(e) AUC compare on PDE dataset

Figure 2. Comparison on AUC

Table 2. Comparison on AUC values (mean ± std)

| Data | Algorithm | Number of queries (percentage of the unlabeled data) | | | | |
|---|---|---|---|---|---|---|
| | | 10% | 20% | 30% | 40% | 50% |
| Equinox | committee | 0.752±0.041 | 0.755±0.033 | 0.757±0.037 | 0.759±0.035 | 0.761±0.037 |
| | DWUS | 0.753±0.037 | 0.76±0.038 | 0.758±0.032 | 0.763±0.033 | **0.769±0.032** |
| | HIVE | **0.76±0.034** | 0.759±0.035 | 0.759±0.034 | 0.759±0.035 | 0.76±0.036 |
| | quire | 0.752±0.037 | 0.752±0.035 | 0.756±0.031 | 0.757±0.032 | 0.758±0.034 |
| | random | 0.755±0.036 | **0.764±0.036** | **0.763±0.032** | **0.763±0.031** | 0.763±0.031 |
| | svmMargin Representative CluserKMeans | 0.754±0.036 | 0.753±0.039 | 0.752±0.038 | 0.752±0.037 | 0.755±0.038 |
| Eclipse JDT Core | committee | 0.742±0.124 | 0.762±0.085 | 0.779±0.019 | 0.768±0.097 | 0.789±0.016 |
| | DWUS | 0.704±0.183 | 0.752±0.124 | 0.787±0.015 | 0.788±0.015 | 0.791±0.016 |
| | HIVE | **0.779±0.015** | **0.787±0.013** | **0.791±0.013** | **0.797±0.013** | **0.801±0.012** |
| | quire | 0.716±0.168 | 0.765±0.095 | 0.781±0.016 | 0.785±0.017 | 0.787±0.017 |
| | random | 0.705±0.181 | 0.748±0.123 | 0.782±0.015 | 0.786±0.015 | 0.79±0.014 |
| | svmMargin Representative CluserKMeans | 0.734±0.153 | 0.717±0.189 | 0.747±0.157 | 0.749±0.158 | 0.767±0.132 |
| Apache Lucene | committee | 0.709±0.029 | 0.708±0.029 | 0.707±0.029 | 0.698±0.072 | 0.706±0.027 |
| | DWUS | 0.695±0.071 | 0.7±0.031 | 0.69±0.078 | 0.706±0.03 | 0.709±0.027 |
| | HIVE | **0.715±0.028** | **0.717±0.027** | **0.719±0.027** | **0.724±0.029** | **0.73±0.029** |
| | quire | 0.672±0.107 | 0.685±0.08 | 0.702±0.032 | 0.687±0.073 | 0.696±0.027 |
| | random | 0.685±0.096 | 0.694±0.083 | 0.705±0.03 | 0.691±0.079 | 0.695±0.073 |
| | svmMargin Representative CluserKMeans | 0.677±0.122 | 0.692±0.101 | 0.686±0.117 | 0.678±0.138 | 0.667±0.159 |
| Mylyn | committee | 0.7±0.02 | 0.699±0.024 | 0.699±0.025 | 0.698±0.025 | 0.702±0.024 |
| | DWUS | **0.678±0.087** | 0.694±0.068 | 0.71±0.019 | 0.709±0.018 | 0.708±0.02 |
| | HIVE | 0.676±0.103 | 0.7±0.065 | 0.701±0.067 | **0.71±0.026** | 0.711±0.024 |
| | quire | 0.673±0.093 | **0.702±0.019** | 0.702±0.019 | 0.709±0.019 | **0.712±0.02** |
| | random | 0.671±0.095 | 0.699±0.02 | **0.704±0.021** | 0.709±0.019 | 0.711±0.022 |
| | svmMargin Representative CluserKMeans | 0.663±0.124 | 0.692±0.086 | 0.671±0.124 | 0.709±0.024 | 0.709±0.024 |
| Eclipse PED UI | committee | 0.684±0.018 | 0.695±0.019 | 0.689±0.068 | 0.691±0.066 | 0.704±0.018 |
| | DWUS | 0.679±0.016 | 0.687±0.017 | 0.693±0.014 | 0.697±0.014 | 0.703±0.016 |
| | HIVE | **0.694±0.019** | 0.7±0.019 | **0.705±0.02** | **0.711±0.019** | **0.716±0.018** |
| | quire | 0.671±0.059 | 0.687±0.017 | 0.689±0.015 | 0.696±0.015 | 0.697±0.016 |

| | | | | | |
|---|---|---|---|---|---|
| random | 0.662±0.08 | 0.686±0.021 | 0.692±0.019 | 0.697±0.017 | 0.7±0.018 |
| svmMargin Representative CluserKMeans | 0.637±0.138 | **0.701±0.018** | 0.683±0.092 | 0.686±0.095 | 0.698±0.074 |

The best performance and its comparable performance based on paired t-tests at 95% significance level are highlighted in boldface.

Table 3. Win/tie/loss counts of HIVE versus the other methods with varied numbers of queries

| Algorithms | Number of queries (percentage of the unlabeled data) | | | | | |
|---|---|---|---|---|---|---|
| | 10% | 20% | 30% | 40% | 50% | In All |
| committee | 4/0/1 | 5/0/0 | 5/0/0 | 4/1/0 | 4/0/1 | 22/1/2 |
| DWUS | 4/0/1 | 4/0/1 | 5/0/0 | 3/0/2 | 4/0/1 | 20/0/5 |
| quire | 5/0/0 | 4/0/1 | 4/0/1 | 5/0/0 | 4/0/1 | 22/0/3 |
| random | 5/0/0 | 4/0/1 | 3/0/2 | 4/0/1 | 4/0/1 | 20/0/5 |
| svmMarginRepresentative CluserKMeans | 5/0/0 | 4/0/1 | 5/0/0 | 5/0/0 | 5/0/0 | 24/0/1 |
| In All | 23/0/2 | 21/0/4 | 22/0/3 | 22/1/3 | 21/0/4 | 108/1/16 |

## 5. Conclusions

In order to make full use of the results of information entropy in the sample labeling process and reduce the cost of labeling in the field of software defect prediction, we proposed an active learning query strategy based on information entropy and vote entropy. The experiment was conducted on the *AEEEM* dataset. We compared our proposed strategy with other common active learning strategies. The empirical research showed that the algorithm can obtain better prediction performance under the same data. The use of hybrid active learning strategies in project software defect prediction is effective, and cross-project software defect prediction [22] is a problem that remains to be studied in the future.

## Acknowledgements

This work was supported by the Nantong Science and Technology Project (No. JC2018134).

## References

1. X. Chen, Q. Gu, W. S. Liu, S. L. Liu, and C. Ni, "State-of-the-Art Survey of Static Software Defect Prediction," *Ruan Jian Xue Bao/Journal of Software*, Vol. 27, No. 1, pp. 1-25, 2016
2. B. Settles, "Active Learning Literature Survey," Computer Sciences Technical Report, University of Wisconsin-Madison, pp. 12-25, 2010
3. S. J. Huang, R. Jin, and Z. H. Zhou, "Active Learning by Querying Informative and Representative Examples," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 36, No. 10, pp. 1936-1949, October 2014
4. C. -L. Li, C. -S. Ferng, and H. -T. Lin, "Active Learning using Hint Information," *Neural Computation*, Vol. 27, No. 8, pp. 1738-1765, August 2015
5. M. Li, H. Zhang, and R. Wu, "Sample-based Software Defect Prediction with Active and Semi-Supervised Learning," *Automated Software Engineering*, Vol. 19, No. 2, pp. 201-230, June 2012
6. G. Luo and K. Qin, "Active Learning for Software Defect Prediction," *IEICE Transactions on Information and Systems*, Vol. E95-D, No. 6, pp. 680 - 683, June 2012
7. H. Lu and B. Cukic, "An Adaptive Approach with Active Learning in Software Fault Prediction," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, pp. 79-88, New York, USA, September 2012
8. X. Zhou, L. Jin, X. Luo, and T. Zhang, "Cross-Version Defect Prediction via Hybrid Active Learning with Kernel Principal Component Analysis," in *Proceedings of 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering* (*SANER*), pp. 209-220, Campobasso, Italy, March 2018
9. H. Lu, E. Kocaguneli, and B. Cukic, "Defect Prediction Between Software Versions with Active Learning and Dimensionality Reduction," in *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 312-322, Washington DC, USA, November 2014
10. K. Wang, D. Y. Zhang, Y. Li, R. M. Zhang, and L. Lin, "Cost-Effective Active Learning for Deep Image Classification," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 27, No. 12, pp. 2591-2600, December 2017
11. K. Nigam and A. McCallum, "Employing EM in Pool-based Active Learning for Text Classification," in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 350-358, San Francisco, USA, July 1998
12. B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models," in *Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pp. 789-800, Florence, Italy, May 2015
13. Z. Xu, J. Liu, Z. J. Yang, G. An, and X. Y. Jia, "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison," in *Proceedings of 2016 IEEE 27th International Symposium on Software Reliability Engineering*, pp. 309-320, Ottawa, Canada, October 2016

14. W. Yang, X. X. Tian, and S. L. WANG, "Recent Advances in Active Learning Algorithms," *Journal of Hebei University* (*Natural Science Edition*), Vol. 37, No. 2, pp. 216-224, 2017
15. F. Khan, X. Zhu, and B. Mutlu, "How Do Humans Teach: On Curriculum Learning and Teaching Dimension," in *Proceedings of Twenty-Fifth Conference on Neural Information Processing Systems*, pp. 1449-1457, Granada, Spain, December 2011
16. D. Y. Meng and Q. Zhao, "What Objective Does Self-Paced Learning Indeed Optimize," arXiv:1511.06049, 2015
17. M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Proceedings of 2010 7th IEEE Working Conference on Mining Software Repositories* (*MSR 2010*), pp. 31-41, Cape Town, South Africa, May 2010
18. F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-Project Defect Prediction using a Connectivity-based Unsupervised Classifier," in *Proceedings of 2016 IEEE/ACM 38th International Conference on Software Engineering* (*ICSE*), pp. 309-320, Austin, USA, May 2016
19. Y. B. Qu and X. Chen, " Software Defect Prediction Method based on Cost-Sensitive Active Learning," *Journal of Nantong University* (*Natural Science Edition*), Vol. 18, No. 1, pp. 9-15, February 2019
20. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction," in *Proceedings of 15th Annual Conference on Evaluation and Assessment in Software Engineering* (*EASE 2011*), pp. 96-103, Durham, UK, November 2011
21. T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," *Neural Computation*, Vol. 10, No. 7, pp. 1895-1923, October 1998
22. X. Chen, L. P. Wang, Q. Gu, Z. Wang, C. Ni, W. S. Liu, et al., "A Survey on Cross-Project Software Defect Prediction Methods," *Chinese Journal of Computers*, Vol. 41, No. 1, pp. 254-274, January 2018