Is the Number of Faults Helpful for Cross-Company Defect Prediction?

Yiyang Jing[1], Jiansheng Zhang[2], *Jin Liu[1]
[1]*State Key Lab. of Software Engineering*, Computer School, Wuhan University, Wuhan, China
[2]*School of Computer Science and Information Engineering*, HuBei University, Wuhan, China
*Corresponding author
*Corresponding author email: jinliu@whu.edu.cn

***Abstract*—In the field of Cross-company Defect Prediction (CCDP), how to deal with the data to make it more accurately predict the cross-company software defects is the focus problem we need to consider. Now a mainstream idea is to determine the weight of the training data based on the similarity between the training data and the test set, and then build the model on the basis of these weighted data. However, sometimes, when we deal with some problems with imbalance class, directly using the weight calculated above may lead to errors. Because a large number of non-defective instance's weight accumulation will lead to the defective instance's weight has a little impact on the final result. This is why we need to consider the addition of the number of defects. Considering the number of defects will effectively eliminate the impact of a large number of non-defective instance's weight accumulation. Therefore, we propose a Transfer-learning Naïve Bayes model considering the number of defective information(NTNB). The method consists of two major stages: weight the data and build the prediction model. In the stage of weighting the data, we not only consider the similarity between the data but also consider the number of defects to get the final weights for data. And we conducted a set of comparative experiments on six open cross-company datasets. The results show that considering the number of defects information can effectively avoid some defective instance is misjudged as non-defective instance, and improve the accuracy of prediction in some unbalanced problems.**

Keywords—software defect prediction;cross-company defect prediction; transfer learning; NTNB

I. INTRODUCTION

Software defect prediction is one of the most important software quality assurance techniques. It aims to detect the defect proneness of new software modules via learning from defect data. So far, many efficient software defect prediction approaches [1-2] have been proposed, but they are usually confined to within company defect prediction (WCDP). WCDP works well if sufficient data is available to train a defect prediction model. However, it is difficult for a new company to perform WCDP if there is limited historical data. Cross-company defect prediction (CCDP) is a practical approach to solve the problem. It trains a prediction model by exploiting one or multiple projects of a source company and then applies the model to target company [3].

Recent year, there have sprung up a large number of excellent articles to solve the cross-company defect prediction problem. For example, Turhan et al. use a Nearest Neighbor Filter model (NN-filter) to select the similar data from source data as training data for more accurate forecasts [4]. Be different from this, Ying Ma et al. select all the source data as training data but add weights to them to build a Transfer Naive Bayes model(TNB)[5]. However, all of these models are not considered the number of faults whether has an impact on the forecast results. That is to say, the information about the number of faults has not been fully utilized.

However, when calculating the weight between training instances and test instances, the number of faults information has an important additive effect on it. For example, we tend to give a higher weight to an instance which has fewer bugs, and these higher weights allow these instances play a greater role in predicting the presence or absence of defects in the test instance. For instance, when calculating the similarity between instances, we get the same degree of similarity between several training instances and the test instance, so which of them is more convincing in determining whether the test instance is defective? In this time, we need to take the number of faults information into consider.

Therefore, this paper firstly gives the corresponding initial weights to the cross-project training instances through the Transfer Learning [6]. Then give the final weight combines with number of faults information and calculate the prior probabilities of the presence or absence of defects. Finally build the forecast model (NTNB) on them. We select six available and commonly used software project datasets and choose five of them as training data, the remaining one as the testing data to do the experiments [7]. The experimental results show that the NTNB model which considering the number of faults is superior than the traditional prediction models.

The structure of this article is as follows, Section II is the related work in this area; Section III introduces the NTNB approach for CCDP. Section IV shows the experiment setup and experiment results. Section V addresses the conclusion and points out the future work.

II. RELATED WORK

In recent years, Machine Learning is more and more widely applied in various fields [8-10], In the field of defect prediction, the machine learning methods are also used a lot. For example, Xiao Yu propose a novel probabilistic graphical model called Bayesian Network based Program Dependence Graph (BNPDG) to locate the defect in the software. In addition, decision trees,

10.18293/SEKE2017-034

neural networks and other methods have also been widely used to predict the defect of software [23-24].

However, with the evolution of the algorithm, it has been found that it is more difficult to improve the prediction accuracy only by improving the algorithm [5]. So people began to improve the prediction accuracy by using more appropriate training data. For example, by removing the data which is poorly related to the company, to mitigate the impact of irrelevant data on forecasting [15].

In the process of cross-company forecasting(CCDP), The choice of training data becomes more important. Because the cross-company defect prediction often means that in the prediction process, the training data and test data have different feature distribution and prior knowledge. This requires us to process the original data to eliminate this priori difference. In order to achieve this purpose, Turhan et al. use a Nearest Neighbor Filter model (NN-filter) to select the similar data from source data as training data for more accurate forecasts [4], Ying Ma et al. select all the source data as training data but add weights to them to build a Transfer Naive Bayes model(TNB) [5].

In addition to selecting the appropriate training data, people also thought that by selecting the valuable characteristics to improve the accuracy of prediction. Zhou Xu proposed a Maximal Information Coefficient with Hierarchical Agglomerative Clustering (MICHAC) method, first of all, sorting the attributes according to the amount of information of them, and then remove the redundant attributes through the hierarchical clustering to get the most valuable attributes [14] [16].

III. METHODOLOGY

In this section, we present our NTNB approach for CCDP. The method consists of two major stages: weight the data and build the prediction model. In the stage of weighting the data, we not only consider the similarity between the data but also consider the number of defects to get the final weights for data. In the stage of building the prediction model, first of all, we calculate the prior probabilities and conditional probabilities based on the previously calculated weights, and then use the principles of the Naïve Bayes classifier to predict the label of the test instance.

A. The Naive Bayes Model

The reason why do we use the Naive Bayes as our forecasting model is that when we encounter such a problem which need to consider all attribute information to get the probability of the result, many algorithms may ignore some weak features, but the Naive Bayes model will use all available information to correct the forecast results. It's important for the defect prediction field which need to consider a large number of attributes information. Although many attributes may have little impact on the result separately, but the combination of them will make a great influence.

The Naïve Bayes' thought is to calculate the probability that an instance belongs to each category under given conditions, and which category's probability is higher, which category we think it belongs to.

For example, there is an instance x, and the category set is $C = \{c_1, c_2\}$. If the probability of x belongs to the category c_1 is larger than it belongs to c_2, then we regard the instance x's category as c_1, otherwise as c_2.

$$P(c_1|x) > P(c_2|x) \rightarrow x \in c_1 \quad (1)$$

$$P(c_1|x) < P(c_2|x) \rightarrow x \in c_2 \quad (2)$$

And $P(c_i \mid x) = \frac{P(c_i)*P(x \mid c_i)}{P(x)}$ .The $P(c_i)$ is the prior probability of class c_i, The $\frac{P(x \mid c_i)}{P(x)}$ is an adjustment factor to adjust the value of the posteriori probability, in order to make it more close to the true probability.

B. The Weight Of Training Data

This part we intend to calculate the weight of the training data by considering the number of faults and the distance between training instances and test instances.

First, we measure the distance between instances by Transfer Learning [6]. The goal of Transfer Learning is to move the knowledge learned from one environment to deal with the problem in a new environment. It's particularly suitable for the cross-company defect prediction, because that the cross-company defect prediction is just using the cross-company code defect information to predict the code deficiencies for different companies. Then how do we determine the weights of training instances in the migration process? An important principle is: the higher the similarity between the training instance and the testing instance, the higher the weight.

Then we first calculate the similarity between instances and calculate the weights based on similarity.

According to TNB [5], the similarity between instances is measured as follows: 1) we need to find the maximum and minimum values of each attribute in the test set and store them into arrays.2) Next we judge whether each attribute of each training instance is within the maximum and minimum range of the corresponding attribute in the test set. If yes, the corresponding instance's support factor will plus one. For example, giving three training instances: x_1= (1, 2, 2, 'false'), x_2= (2, 1, 3, 'false'), x_3= (2, 2, 4, 'true'|3), where the bit after 'true' is the number of defects, and one training instance: y_1= (2, 2, 3). Then Max = (2, 2, 3), Min = (2, 2, 3). So for the first instance = (1, 2, 2, 'false'), the support factor $s_1 = 1$; Similarly, $s_2 = 2$, $s_3 = 2$.

Next, we will calculate the weight of each training instance by the above support factors. Here, according to L. Peng's [17] paper, we use the gravitational formula to simulate the gravitational force between the data, that is the weight.

$$w_i = G\frac{m_1 m_2}{(r)^2} = \frac{kmMs_iM}{(k - s_i + r)^2} \propto \frac{s_i}{(k - s_i + r)^2} \quad (3)$$

In Eq.(3), w_i is the weight of each training instance; G is the universal gravitational constant, m_1 and m_2 is the mass of two objects. r is the distance between the two objects; k is the number of attributes, m is the number of test cases, M is the mass of each attribute, so kmM is the mass of all test cases; s_i is the support factor of each training instance, so s_iM is the mass of each training instance; Similarity, $(k-s_i+1)$ is the distance between each training instance and the test set. In this process, we can remove some fixed constants, and thus get the right part

of the Eq.(3). In the end, we can get the final weight w by accumulate each training instances' weight w_i.

But that's not our final weights. Sometimes, when we deal with some problems with imbalance class, directly using the value calculated above may lead to errors. Because a large number of non-defective instance's weight accumulation will lead to the defective instance's weight has a little impact on the final result. For example, for training instances x_1= (1, 2, 2, 'false'), x_2= (2, 1, 3, 'false'), x_3= (2, 2, 4, 'true'|3), and the test instance: y_1= (2, 2, 3). According to Eq.(3), we can calculate the weight of each training instance, w_1 = 1/9, w_2 =1/2, w_3 = 1/2. If we predict the presence of defects directly according to these weights, then the test instance is likely to be judged as having no errors due to the accumulation of the non-defective instance's weight. However, based on the similarity between the test instance and the defective training instance, this test instance is still likely to be defective. So, calculating weights without considering the number of defects can cause some defective instance is mistaken for non-defective instance, which will reduce the prediction accuracy.

Therefore, we consider the number of defects information in calculating the weight of each instance. That is to say we add the number of defects on the basis of Eq.(3).

$$w_i = \frac{s_i n_i}{(k - s_i + 1)^2} \tag{4}$$

n_i is the number of defects for the *i*-th training instance.

Next we will describe how to predict the presence of defects on the basis of this weight.

C. NTNB Approach

The general idea of our algorithm is to use the Naïve Bayes as our forecasting model on the weighted data. According to Eq.(1) and Eq.(2), in order to calculate the posterior probability $P(c_i|\, x)$, we need to calculate $P(c_i)$, $P(x \mid c_i)$ and $P(x)$. To better explanation, we define the indicative function $f(x, y)$: if x = y, $f(x, y) = 1$, otherwise, $f(x, y) = 0$.

According to [22], The prior probabilities $P(c)$ can be expressed as:

$$P(c) = \frac{\sum_{i=1}^{n} f(c_i,c) w_i + 1}{\sum_{i=1}^{n} w_i + n_c} \tag{5}$$

n is the total number of training instances, $f(c_i, c)$ is the indication function described above, c_i is the category of the i-th training instance, c is the label of the attribute that we want to calculate the prior probability, w_i is the weight of the *i*-th training instance, and n_C is the total number of categories.

Next, we will calculate the conditional probability of the *j*-th attribute a_j in the training instance x_i according to the formula in [18].

$$P(a_j|c) = \frac{\sum_{i=1}^{n} f(a_{ij},a_j) f(c_i,c) w_i + 1}{\sum_{i=1}^{n} f(c_i,c) w_i + n_j} \tag{6}$$

n is the total number of training instances, a_{ij} is the value of *j*-th attribute in *i*-th training instance, c_i is the category of the *i*-th training instance, c is the label of the attribute that we want to calculate the conditional probability, w_i is the weight of the *i*-th training instance, and n_j is the number of different values for attribute a_j in the training set.

Suppose that x is an instance. Then:

$$P(x) = \sum_{i=1}^{n_c} P(c_i) \prod_{j=1}^{k} P(a_j \mid c_i) \tag{7}$$

n_C is the total number of categories, k is the number of attributes in instance x, $P(ci)$ is introduced in Eq.(5), $P(aj \mid ci)$ is introduced in Eq.(6).

So we can predict the x's category by:

$$P(c_i \mid x) = \frac{P(c_i) \prod_{j=1}^{k} P(a_j| \, c_i)}{P(x)} \tag{8}$$

The $P(c_i| \, x)$ is the probability that instance x is predicted as class c, $P(c_i)$ is introduced in Eq.(5), k is the number of attributes in instance x, $P(aj \mid ci)$ is introduced in Eq.(6), $P(x$ is introduced in Eq.(7).

If $P(c_i \mid x) > P(c_i \mid x), 1 \ll j \ll nc, j \neq i$,then, $x \in c_i$. Otherwise, $x \in c_j$. where n_c is the total number of categories.

Next, we will classify the below examples from considering the number of defects and not considering it.

For training instances x_1= (1, 2, 2, 'false'), x_2= (2, 1, 3, 'false'), x_3= (2, 2, 4, 'true'|3), and the test instance: y_1= (2, 2, 3). Then n = 3, n_C = 2, k = 3.

1) Considering The Number Of Defects

① According to Eq.(4), we can calculate the weight of each training instance, w_1 = 1/9, w_2 =1/2, w_3 = 3/2.

② According to Eq.(5)

$$P('true') = \frac{(w_3 * 1) + 1}{(w_1 * 1 + w_2 * 1 + w_3 * 1) + 2} = 0.608$$

$$P('false') = \frac{(w_1 * 1 + w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1 + w_3 * 1) + 2} = 0.392$$

③ According to Eq.(6), n_1=2 ,n_2=2, n_3= 3.

$$P(a1 = 2|'true') = \frac{(w_3 * 1) + 1}{(w_3 * 1) + 2} = 0.714$$

$$P(a1 = 2|'false') = \frac{(w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 2} = 0.574$$

$$P(a2 = 2|'true') = \frac{(w_3 * 1) + 1}{(w_3 * 1) + 2} = 0.714$$

$$P(a2 = 2|'false) = \frac{(w_1 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 2} = 0.426$$

$$P(a3 = 3|'true') = \frac{1}{(w_3 * 1) + 3} = 0.222$$

$$P(a3 = 3|'false') = \frac{(w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 3} = 0.415$$

④ According to Eq.(7),

$$P(y_1) = P('true') \prod_{j=1}^{3} P(a_j \mid 'true') + P('false') \prod_{j=1}^{3} P(a_j \mid 'false')$$

$$= 0.0688 + 0.0398 = 0.1086$$

⑤ According to Eq.(8)

$$P('true'|y_1) = \frac{P('true')\prod_{j=1}^{3}P(a_j|'true')}{P(y_1)} = 0.6335$$

$$P('false'|y_1) = \frac{P('false')\prod_{j=1}^{3}P(a_j|'false')}{P(y_1)} = 0.3665$$

Obviously, $P('true'|y_1) > P('false'|y_1)$, So the test instance y_1's category is true.

2) Not Considering The Number Of Defects

① According to Eq.(4), we can calculate the weight of each training instance, $w_1 = 1/9$, $w_2 = 1/2$, $w_3 = 1/2$.

② According to Eq.(5)

$$P('true') = \frac{(w_3 * 1) + 1}{(w_1 * 1 + w_2 * 1 + w_3 * 1) + 2} = 0.482$$

$$P('false') = \frac{(w_1 * 1 + w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1 + w_3 * 1) + 2} = 0.518$$

③ According to Eq.(6), $n_1=2$,$n_2=2$, $n_3=3$.

$$P(a1 = 2|'true') = \frac{(w_3 * 1) + 1}{(w_3 * 1) + 2} = 0.6$$

$$P(a1 = 2|'false') = \frac{(w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 2} = 0.574$$

$$P(a2 = 2|'true') = \frac{(w_3 * 1) + 1}{(w_3 * 1) + 2} = 0.6$$

$$P(a2 = 2|'false) = \frac{(w_1 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 2} = 0.426$$

$$P(a3 = 3|'true') = \frac{1}{(w_3 * 1) + 3} = 0.286$$

$$P(a3 = 3|'false') = \frac{(w_2 * 1) + 1}{(w_1 * 1 + w_2 * 1) + 3} = 0.415$$

④ According to Eq.(7),

$$P(y_1) = P('true')\prod_{j=1}^{3}P(a_j|'true') + P('false')\prod_{j=1}^{3}P(a_j|'false')$$

$$=0.0496 +0.0526=0.1022$$

⑤ According to Eq.(8)

$$P('true'|y_1) = \frac{P('true')\prod_{j=1}^{3}P(a_j|'true')}{P(y_1)} = 0.485$$

$$P('false'|y_1) = \frac{P('false')\prod_{j=1}^{3}P(a_j|'false')}{P(y_1)} = 0.514$$

$P('true'|y_1) < P('false'|y_1)$, So the test instance y_1's category is false.

So, we can see clearly that calculating weights without considering the number of defects can cause serious mistakes, such as some instances of potentially defective are misjudged as having no defects.

Why did this happen? The reason is that a large number of non-defective instance's weight accumulation will leads to the defective instance's weight has a little impact on the final result. That is to say, the class imbalance caused part of information has little influence on the result, unfortunately it maybe just the part of information we need. This is why we need to consider the addition of the number of defects.

Algorithm1 presents the pseudo-code of NTNB approach.

Algorithm 1. NTNB approach

Input: Training set **L:** $\{x_1, x_2, \ldots, x_m\}$, Test set **U:** $\{u_1, u_2, \ldots, u_n\}$

Output: The classifier P

```
1.  for each attribute a_j in U do:
2.      get the max value and min value for a_j.
3.      get the number of different values n_j for a_j:
4.  end for
5.  for each instance x_i in L do:
6.      calculate w_i through Eq.(4)
7.      get the total number of categories n_c
8.  end for
9.  According to Eq.(5, 6, 7, 8):
10.     Build weighted Naïve Bayes
11. for each instance u_i in U do:
12.     predict u_i through Eq.(1, 2)
13. end for
14. return P
```

IV. EXPERIMENTS

In this section, we evaluate our proposed NTNB approach to perform CCDP. We first introduce the experiment dataset and the performance measures. Then, in order to investigate the performance of NTNB, we perform some contrast experiment.

A. Data set

In this experiment, we employ 6 available and commonly used software project datasets with their 26 releases which can be obtained from PROMISE [7]. The details about the datasets is shown in Table I, where *#Instance* represents the number of instances, *#Defects* represents the total number of faults in the release, *%Defect* represents the percentage of defect-prone instances, and *Max* is the maximum value of faults.

TABLE I. DETAILS OF EXPERIMENT DATASET

Project	*Release*	*#Instance*	*#Defects*	*%Defects*	*Max*
Ant	Ant-1.3	125	33	16.0%	3
	Ant-1.4	178	47	22.5%	3
	Ant-1.5	293	35	10.9%	2
	Ant-1.6	351	184	26.2%	10
	Ant-1.7	745	338	22.3%	10
Camel	Camel-1.0	339	14	3.4%	2
	Camel-1.2	608	522	35.5%	28
	Camel-1.4	872	335	16.6%	17
	Camel-1.6	965	500	19.5%	28
Jedit	Jedit-3.2	272	382	33.1%	45
	Jedit-4.0	306	226	24.5%	23
	Jedit-4.1	312	217	25.3%	17
	Jedit-4.2	267	106	13.1%	10
	Jedit-4.3	492	12	2.2%	2
Synapse	Synapse-1.0	157	21	10.2%	4
	Synapse-1.1	222	99	27.0%	7
	Synapse-1.2	256	145	33.6%	9
Prop	Prop-1	18471	2738	14.8%	37
	Prop-2	23014	2431	10. 6%	22

Project	*Release*	*#Instance*	*#Defects*	*%Defects*	*Max*
	Prop-3	10274	1180	11. 5%	11
	Prop-4	8718	840	9.6%	22
	Prop-5	8516	1299	15.3%	19
	Prop-6	660	66	10%	4
Forrest	Forrest-0.6	6	1	16.7%	1
	Forrest-0.7	29	5	17.2%	8
	Forrest-0.8	32	2	6.3%	4

There are the same 20 independent variables (the 20 feature metrics) and one dependent variable (the number of faults) in each dataset [19].

And we combine the different versions of the same project as a data set. And choose five from them together as a training set, the remaining one as a test set to do experiment.

B. Performance measures

In the experiment, we employ three commonly used performance measures including *pd*, *pf* and *f-measure* to judge the experimental results [5]. They are summarized as follows.

TABLE II. PERFORMANCE MEASURES

		Actual	
		yes	***no***
Predicted	***yes***	***TP***	***FP***
	no	***FN***	***TN***

- TP is the number of defective instances are correctly predicted. FP is the number of defective instances are predicted as non-defective. FN is the number of non-defective instances are predicted as defective. TN is the number of non-defective instances are correctly predicted.

- The precision is the measure of defective modules that are correctly predicted within the instances which is predicted to be defective. The higher the precision, the better the results.

$$\text{precision} = \frac{TP}{TP+FP} \tag{9}$$

- The recall rate or *pd* is the measure of defective modules that are correctly predicted within the defective class. The higher the *pd*, the fewer the false negative results.

$$pd = \frac{TP}{TP+FN} \tag{10}$$

- Probability of false alarm or *pf* is the measure of non-defective modules that are incorrectly predicted within the non-defective class. Unlike *pd*, the lower the *pf* value, the better the results.

$$pf = \frac{FP}{FP+TN} \tag{11}$$

- The *f*-measure is the harmonic average of *pd* and precision. The higher the measure, the better the results.

$$f = \frac{2*pd*\text{precision}}{pd+\text{precision}} \tag{12}$$

Here we generally use *f*-measure as the total performance evaluation standard because it's the combination of *pf* and precision.

C. Experimental results

In order to better verify the performance of NTNB algorithm, we use the training set and test set in table III to do a set of comparative experiments. We mainly compared the *pd*, *pf* and f parameters of NTNB, TNB, and NB. Their respective characteristics are as follows:

NTNB considering the number of defects based on the TNB to avoid the imbalance problem. The imbalance problem is in the process of forecasting, the accumulation of a large number of defective instance's weights resulting in the defective instance's weight has a little impact on the final result.

TNB assigns the data weights according to the similarity between the training instance and the test set, and builds the Naïve Bayes model on these weighted data [5].

While NB did not consider the difference between cross-company data, directly building the model.

The following table III shows the *pf* and *pf* values in the experimental results for our three approaches, where the boldface represents the best result of this experiment. And Fig. 1 shows the scatter plots of *pd* and *pf* for three CCDP models on 6 datasets. If it has a higher recall rate *pd* and less false alarm *pf*, then the defect model has more points distributed in the lower right corner, which also represents a better performance for the forecasting model.

TABLE III. PD,PF VALUES, THE REASULTS OF THREE APPROACH

No.	Test Set	NTNB		TNB		NB	
		PD	PF	PD	PF	PD	PF
1	synapse	**0.556**	0.220	0.506	0.180	0.305	**0.112**
2	prop	0.182	**0.073**	**0.257**	0.118	0.197	0.137
3	jedit	0.696	0.321	**0.706**	0.327	0.423	**0.207**
4	forrest	**0.625**	0.305	0.375	0.237	0.000	**0.017**
5	camel	**0.393**	0.187	0.383	0.184	0.110	**0.099**
6	ant	0.726	0.301	**0.729**	0.300	0.346	**0.137**

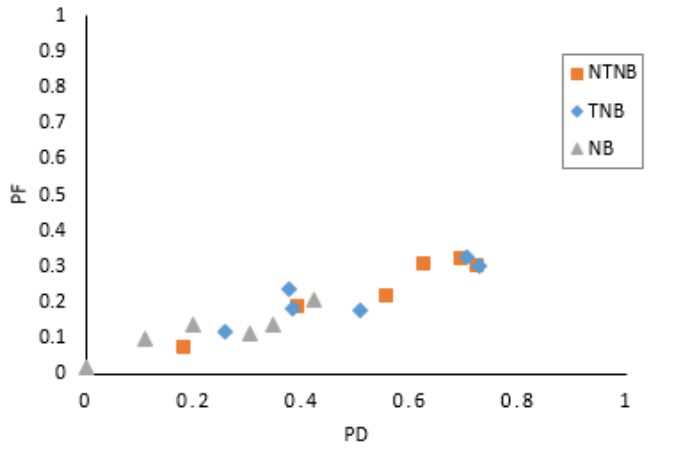

Fig. 1. Scatter plots of *pd* and *pf* for three CCDP models on 6 dataset

The table IV is the *f*-measures in the experimental results for our three approaches, where the boldface represents the best result of this experiment. The experimental results show that in some data sets with unbalanced problems, considering the number of defects information can effectively avoid some defective instance is misjudged as non-defective instance, and improve the accuracy of prediction [20].

TABLE IV. F-MEASURE VALUES, THE REASULTS OF THREE APPROACH

No.	Test data	NTNB	TNB	NB
1	synapse	**0.506**	0.498	0.337
2	prop	0.213	**0.245**	0.151
3	jedit	0.431	**0.432**	0.240
4	forrest	**0.323**	0.240	0.000
5	camel	**0.369**	0.363	0.116
6	ant	0.504	**0.506**	**0.**314

V. CONCLUSION AND FUTURE WORK

In the field of cross-company defects prediction, many models do not consider the defects number information in predicting the result. However, according to our analysis, in some imbalance problems, considering the number of defects will effectively eliminate the impact of a large number of non-defective instance's weight accumulation. To prove this, we conducted a set of comparative experiments on six open cross-company datasets. The results show that considering the number of defects information can effectively avoid some defective instance is misjudged as non-defective instance, and improve the accuracy of prediction in some unbalanced problems. In the following work, we want to find a balance to avoid defects number information overfitting the forecast result. For example, give an extreme example: there are 100000 errors for one instance, then what can we do to avoid it exceeding intervening the result? In addition, we will apply our model on the social data to further prove its effectiveness [21-22], and try to apply our model to other areas [11-12].

ACKNOWLEDGMENT

This work is partly supported by the grants of National Natural Science Foundation of China (No.61572374, No.U163620068, No.U1135005) and the Academic Team Building Plan from Wuhan University and National Science Foundation (NSF) (No. DGE-1522883).

REFERENCES

[1] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," IEEE Transactions on Reliability,2014, 63(2):676-686.

[2] X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in: Proc. of the 36th International Conference on Software Engineering (ICSE), 2014,pp. 414-423.

[3] Xiaoyuan Jing et al, "Heterogeneous Cross-Company Defect Prediction by Unified Metric Representation and CCA-Based Transfer Learning," in: Proc. of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp.496-507.

[4] B. Turhan, T. Menzies, A.B. Bener, J.D. Stefano, "On the relative value of crosscompany and within-company data for defect prediction," Empirical Software Engineering, 2009, 14(5):540-578.

[5] Ying Ma, Guangchun Luo, Xue Zeng and Aiguo Chen, "Transfer learning for cross-company software defect prediction," Information and Software Technology,2012, 54:248–256.

[6] Y. Shi, Z. Lan, W. Liu and W. Bi, "Extending semi-supervised learning methods for inductive transfer learning," In: Ninth IEEE International Conference on Data Mining, 2009, pp.483–492.

[7] G. Boetticher, T. Menzies, T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, 2007 <http://promisedata.org/repository>.

[8] Xu, Z., et al, " Hierarchy-Cutting Model based Association Semantic for Analyzing Domain Topic on the Web," IEEE Transactions on Industrial Informatics, doi:10.1109/TII.2017.2647986.

[9] Xu, Z., et al, "The Mobile Media based Emergency Management of Web Events Influence in Cyber-Physical Space," Wireless Personal Communications: ,DOI: 10.1007/s11277-016-3689-7.

[10] Xu, Z., et al, "Building Knowledge Base of Urban Emergency Events based on Crowdsourcing of Social Media," Concurrency and Computation: Practice and Experience, 2016 ,28(15) :4038-4052

[11] Ziwei Liu, Chuanbo Wei, Yang Ma, Hui Li, Xiaoguang Niu* and Lina Wang, "UCOR: An Unequally Clustering-based Hierarchical Opportunistic Routing Protocol for WSNs," Springer Berlin Heidelberg, 2013, 7992:175-185.

[12] Ziwei Liu, Xiaoguang Niu, Xu Lin, Ting Huang Yunlong Wu and Hui Li, "A Task-Centric Cooperative Sensing Scheme for Mobile Crowdsourcing Systems," 《Sensors》, 2016, 16(5):746.

[13] Xiao Yu, Jin Liu, Zijiang James Yang, Xiao Liu, Xiaofei Yin and Shijie Y, "Bayesian Network Based Program Dependence Graph for Fault Localization". ISSRE Workshops 2016: 181-188.

[14] Zhou Xu, Jin Liu, Zijiang Yang, Gege An and Xiangyang Jia, "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison". ISSRE 2016: 309-320.

[15] Xiao Yu, Jin Liu, Mandi Fu, Chuanxiang Ma and Guoping Nie, "A Multi-Source TrAdaBoost Approach for Cross-Company Defect Prediction". SEKE 2016: 237-242.

[16] Zhou Xu, Jifeng Xuan, Jin Liu and Xiaohui Cui, "MICHAC: Defect Prediction via Feature Selection Based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering". SANER 2016: 370-381.

[17] L. Peng, B. Yang and Y. Chen, "A. Abraham, Data gravitation based classification," Information Sciences,2009,179(6):809-819.

[18] E. Frank, M. Hall, B. Pfahringer, "Locally Weighted Naive Bayes," in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2003, pp. 249–256.

[19] Xiao Yu, Jin Liu, Mandi Fu, Chuanxiang Ma, Guoping Nie: "A Multi-Source TrAdaBoost Approach for Cross-Company Defect Prediction.," SEKE 2016: 237-242

[20] Lin Chen, Bin Fang, Zhaowei Shang and Yuanyan Tang, "Negative samples reduction in cross-company software defects prediction," Information and Software Technology,2015,62:67-77.

[21] Zheng Xu et.a, "Multi-modal Description of Public Safety Events using Surveillance and Social Data," IEEE Transactions on Big Data, 2017, 10.1109/TBDATA.

[22] Zheng Xu, Yunhuai Liu, Hui Zhang, Xiangfeng Luo, Lin Mei and Chuanping Hu, "Building the Multi-Modal Storytelling of Urban Emergency Events Based on Crowd sensing of Social Media Analytics," Mobile Networks and Applications: ,DOI: 10.1007/s11036-016-0789-2.

[23] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl and S.J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," IEEE Transactions on Neural Networks,1997, 8 (4): 902–909.

[24] S. Kanmani, V. Rhymend Uthariaraj, V. Sankaranarayanan and P. Thambidurai, "Object-oriented software fault prediction using neural networks," Information and Software Technology,2007, 49 (5) :483–492.