General models for defect prediction?

[progress report]

Lonnie Bowe
West Virginia University
lbowe@csee.wvu.edu

Aglika Gyaourova
West Virginia University
agyaouro@csee.wvu.edu

Zack Hutzell
West Virginia University
zhutzell@csee.wvu.edu

ABSTRACT

Keywords

defect predictors, static attributes, general model

1. INTRODUCTION

2. DATABASES

3. METHODS

Our goal is to find a general model for defect predictors. For this purpose, the experiments are divided into within company (WC) and cross-company (CC). For the WC experiments, 90% of one data set is used for training the classifier and the rest 10% for evaluating the classifier performance. This process is repeated 10 times, by randomly selecting the instances to be used for evaluation.

In all data sets, the distribution of the instances from the two classes, i.e. the defect and the non-defect, is imbalanced. The defect class is represented by a smaller number of instances ranging from 0.4% to 32% of all instances in different data set [2]. During training the number of instances of the non-defect class was reduced to the number of instances in the defect class by random deletion.

The Bayes classifier finds a decision that minimizes the Bayes error among classes. This is the theoretically optimal classifier when the attribute values of each class classes have a normal probability distribution. The Bayes classifier operates using the posterior probabilities of the classes as defined by the Bayes' theorem. In a two class problem having only a single attribute, the classifier decision is the attribute value at which the posterior probability of one of the classes becomes larger than the posterior probability of the other class. The naive Bayes classifier assumes independent attributes. Practice in software engineering has shown that this classifier performs well even when its assumptions are not met.


```
equal-frequency-discretizer(dataset, bins)
  for each column in dataset
    //sort the values in the column
    sort(dataset, column)
    for each value in the current column
      //change each value to its discretized replacement
      value=min(n-1, floor((value/(length(column)-1))*bins))
```

Figure 2: Equal frequency discretization.

```
equal-width-discretizer(set, bins)
    minval=min(set)
    maxval=max(set)
    for each value in set
      //find the interval in which the value belongs
      //merge the last two intervals
      value=min(nbins,
                floor((value-minval)/(maxval-minval)*nbins))
```

Figure 3: Equal width discretization (`Nbins`).

Applying naive Bayes on categorical (discrete) data is fast and easy. In this case the naive Bayes operates on the conditional frequencies of the categorical values 1. Since discretization of continuous attributed does not have significant effect on the performance [1], we mapped the attribute values into a discrete range before running the learner. A discretization method that accounts for the distribution of the values is the equal frequency discretization. This method splits a list of numbers into a predefined number of intervals, such that every interval contains the equal count of list elements, figure 3.

Another discretization method, `Nbins`, creates intervals whose value ranges have equal width, figure **??**.

4. EXPERIMENTS

We performed experiments on cross-project, within company data using the sets `ar3`, `ar4`, and `ar5`. The learner was naive Bayes and data was preprocessed using equal frequency discretization. Deletion of rows was used to obtain an equal number of instances in the two classes. Thus, the prior probabilities of the classes are the same. Equal priors may be an appropriate assumption and this is one way to achieve it for data having a large sample bias. Instances

```
naive-bayes-classify(instance train)
  k = 1
  m = 2
  classification = null
  classification_score = inf
  for each class in classes(train)
    prior_probability=(k+get_class_count(train, class))/(length(train)+(k*(length(classes(train)))))
    sum = 0
    for each feature in row
      if(!class_column(feature) && !unknown_feature(feature))
        sum = sum + (class_frequency(feature, class, train)
                  + (m * prior_probability)) / (length(get_class_count(train class)) + m)
    if(sum > classification_score)
      classification = class
  return classification
```

Figure 1: Naive Bayes classifier on categorical data.

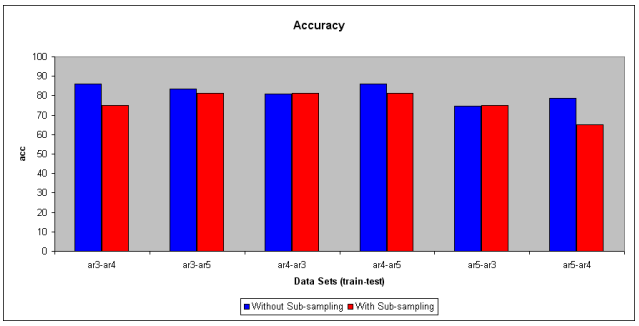

Figure 4: Accuracy for the ar* data sets using cross-project data.

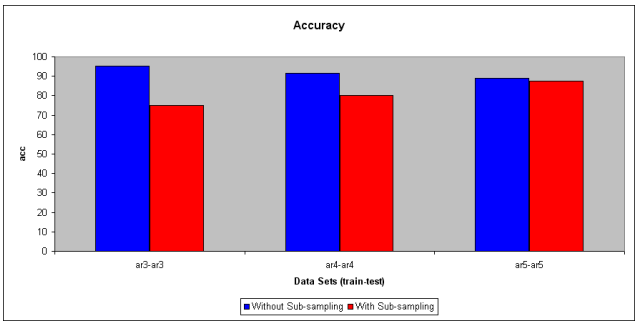

Figure 5: Accuracy for the ar* data sets using within project data.

from the over-represented class were deleted at random before training the classifier.

4.1 Performance evaluation

To evaluate the performance of the learned model we used the accuracy, recall, and precision measures, defined by:

$$\begin{aligned} accuracy &= (A+D)/(A+B+C+D) \\ precision &= D/(C+D) \\ recall &= D/(B+D)\ , \end{aligned} \tag{1}$$

where A, B, C, and D are the true negatives, false negatives, false positives, and true positives respectively. The accuracy gives the portion of correct decision over all classes. The precision is the portion of true positives from all instances predicted as a given class and the recall is the portion of true positives from all instances belonging to a given class. All these measures have a range between 0 and 1, with 1 being the best score.

4.2 Results

The accuracy of performance degrade when using cross-project data compared to within project data and the original data sets, figure 6 and figure 7. However, when the training data was subsampled to equal priors there was significant improvement in performance when training on the `ar3` set and testing on the `ar5` set. Moreover, the performance did not degrade when moving from within project to cross project experiments. This suggests, that subsampling gives a more accurate estimation for the performance of a model in a cross-project prediction. Training on data with equal priors is also supported by the precision versus recall plots, figure **??** and figure **??**. When using these two measures, the prediction degraded substantially when using the original data sets. In contrast, after subsampling, the prediction degraded less and in some cases was better in the cross-project experiments compared to within project experiments.

5. CONCLUSIONS

Our goals for the next project are:

- Implement attribute selection.
- Implementing knn to be used in the cross-company experiments.
- Run cross-company experiments.
- Running the experiments using cross-validation.
- Trying different subsampling, e.g. the Burak or the super-Burak filters.
- Try different learners.

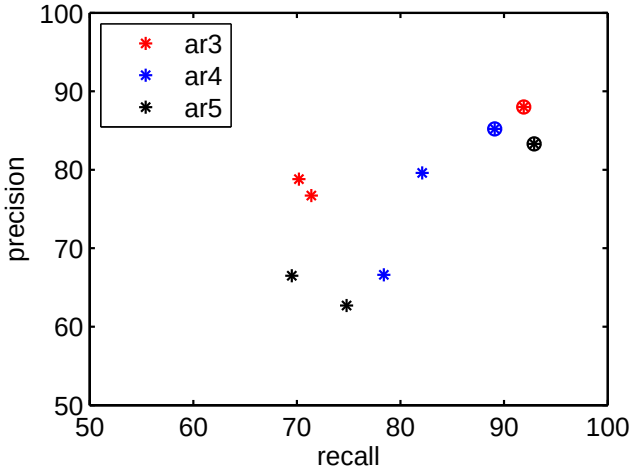

Figure 6: Precision versus recall without subsampling. The star inside a circle markers indicate the within project prediction.

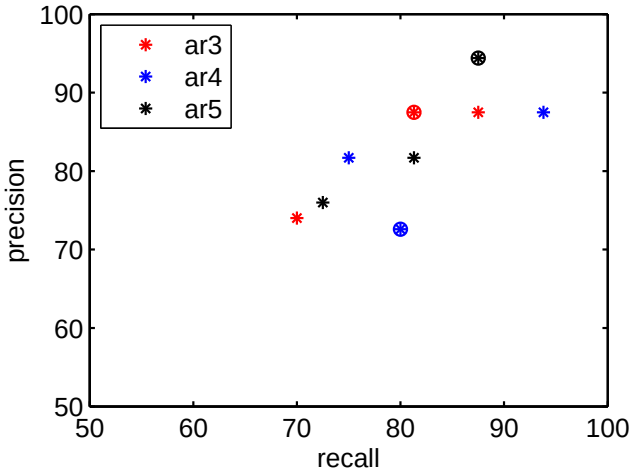

Figure 7: Precision versus recall with subsampling. The star inside a circle markers indicate the within project prediction.

6. REFERENCES

[1] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.

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