Summer Intern Project
Machine Learning

Software Defect Prediction using Transfer Learning

Submitted by:
Nitin Kumar Jangra
2K18/AE/036

Submitted to:
Dr. Ruchika Malhotra

Index

1. Abstract

Transfer learning is a method of storing and applying knowledge acquired while solving one problem to a different but related problem. A defect prediction model created with a C++ dataset, for example, may be used to a software dataset created with the Java programming language.
Cross-project defect prediction (CPDP) uses prediction models created by prior projects and that model predicts faults for new projects. Most studies, however, have the same flaw: they require homogeneous data, which means that various projects must have the same metric sets or features. This paper discusses methods for predicting defects in two projects having different features or heterogeneous feature sets.
While matching metrics, the question arises about the size of the sample of distributions in both the source and target projects. This research demonstrates that data sets where as few as 50 examples are available is sufficient to develop a defect prediction model using a mathematical model. The outcome of this study is that, even in defect datasets of distinct metric sets it is possible to transfer the defect prediction learning.
In this project we used feature selection and metric matching to transfer defect prediction knowledge from one dataset to another. Various python libraries were used to create the model and test it.

2. Formulation of model

Figure 1 shows the basic principle behind our defect prediction model. The techniques used in our model are shown – metric selection in source datasets, metric matching between source and target dataset and the model training and prediction.

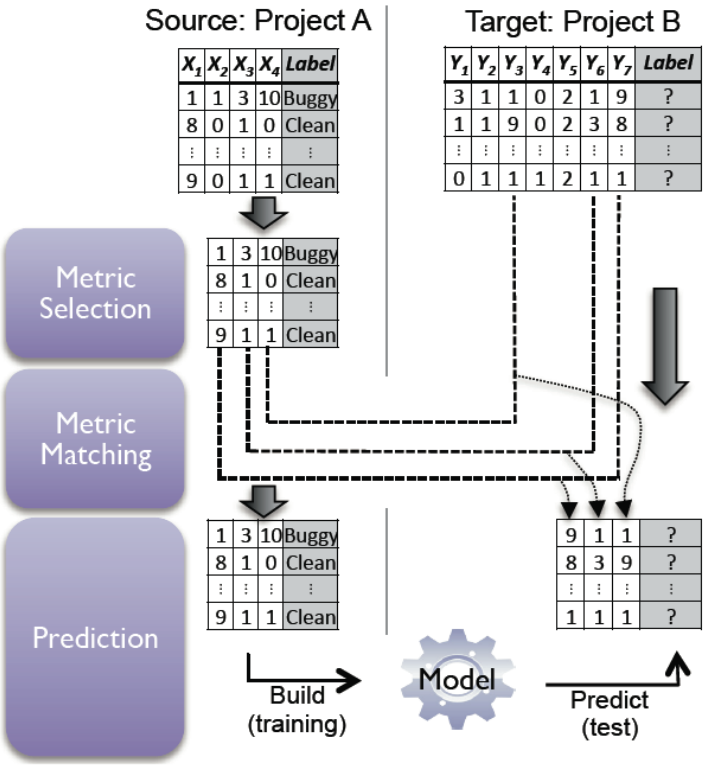

Figure 1 Heterogenous Defect Prediction

2.1 Data pre-processing

Our prediction model requires data in the form of a ***pandas*** dataframe so that the model can be easy to create the model. The defect data from publicly available sources is present in the form of ***.arff*** files. First, we import the ***arff*** function from the ***scipy.io*** library and load the .arff files. The files are then converted into NumPy dataframe by using the ***pd.DataFrame*** option in the ***pandas*** library. The last column of every dataframe has a different name for example- buggy or defects and the values are in the form of yes/no or true/false but we need the values to be in the form of 0's and 1's so we create a new column named "**Defects"** in both source and target datasets and using ***LabelEncoder.fit_transform*** function from the ***sklearn.preprocessing*** library we transform the yes/no and true/false in the form of 0 and 1.

2.2 Metric selection in Source Datasets

Various metric selection procedures are known, including chi-square, relief-F, gain ratio, and so on. For metric selection, we used the gain ratio strategy in this model. We used the ***mutual_info_classifier*** function from the ***sklearn.feature_selection*** library in our experiments. As proposed by Gao et al., we select the top 15% of metrics when using feature selection. For selecting top 15% metrics we used the ***SelectPercentile*** function also from the ***sklearn.feature_selection*** library. For example, there are 36 metrics in the NASA group, and the top 5 metrics with the highest gain ratio value are chosen for our model.

2.3 Source and Target Metrics Matching

Our prediction model is built around matching source and target datasets. We do metric matching as almost all software defect metrics follow the same defect occurring tendency. It is evident from past studies that a more complicated source code and development process increases the defect occurring tendency of the software. Also, the metric value is also generally proportional to the complexity of the source code and development process. That means if a metric value is high, the chances of the software being buggy are also high. As a result, a variety of product and process measures, such as "McCabe's cyclomatic", "Lines of code" (LOC), and the number of developers editing a file, show similar trend.
Metric matching is done to transfer the knowledge of defect occurring conditions from a source project to anticipate faults in a target project by matching measurements or metrics. Assume that in a given Java project (source), there is a metric, RFC (the number of methods invoked by a class). Previous studies show that if a class file has an RFC value greater than 40 is prone to be buggy. Now assume that the target project has a metric – the number of operands and it also shows the same trend as the metric, RFC. Now if we match these two metrics, we can easily predict the defects in the target dataset using the model trained on the source dataset.

There are various statistical tests used to find these matching scores between metrics for example-

- Percentile based matching (**PAnalyzer**)
- Kolmogorov-Smirnov Test based matching (**KSAnalyzer**)
- Spearman's correlation-based matching (**SCoAnalyzer**)

Here, in our defect prediction model we use the **Kolmogorov-Smirnov Test** to find out the set of matched metrics pair by comparing their matching scores.

The **p-value** from the Kolmogorov-Smirnov Test (KS-test) is used as the **matching score** between two metrics. The KS-test compares two samples using a non-parametric statistical test. The KS-test is particularly useful when we are unsure about their variance and whether the two samples are normal. The features in some project datasets included in this study have exponential distributions and metrics in other datasets have unknown distributions and variances, the KS-test is a helpful statistical test to compare two metrics. The ***scipy.stats*** library's ***ks_2samp*** implementation was used for the KS-test implementation.
The matching score is the p value of the source and target metric pair. It's worth noting that with KSAnalyzer, a better matching score does not imply that two metrics are more similar.

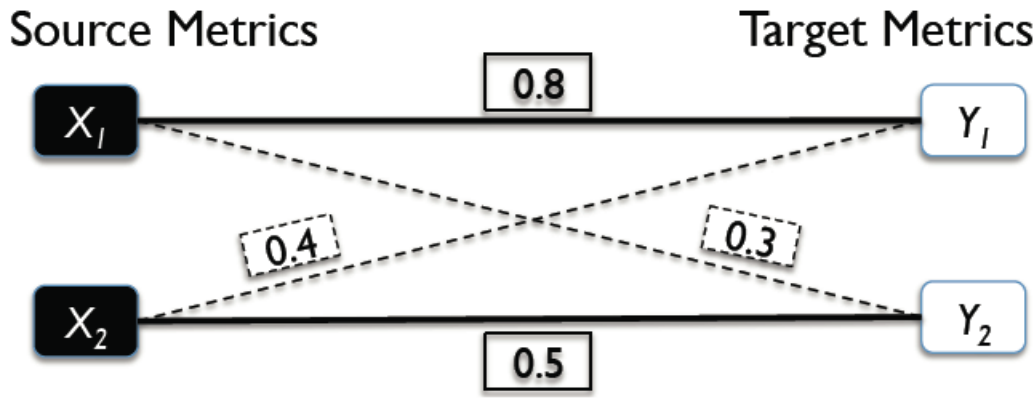

Figure 2 Source and Target Metric Matching

A sample match is shown in Figure 2. Two source measurements (X1 and X2) and two target metrics are available (Y1 and Y2). As a result, four possible matching pairings exist: (X1, Y1), (X1, Y2), (X2, Y1), and (X2, Y2) (X2, Y2). The matching score is shown as the number in rectangles on the edges. The matching score between the metrics X1 and Y2, for example, is 0.3.
We create a bipartite graph using the ***nx.Graph*** data structure from the ***NetworkX*** library. We add the column names of the features that we selected from the source dataset in section 2.2 as the nodes of the graph with the bipartite parameter as 0 and the column names of the target metrics as nodes with the bipartite parameter as 1. Then we add edges between the source and target metric nodes in the graph with their p-values as the weight of edges. Additionally, we don't add the edge between the metrics is the p-value of their KS-test does not satisfy the minimum cut-off criteria. If the cut-off for matching score is 0.35, for example, we only include metric pairs whose matching score is larger than 0.35. When the cut-off criterion is 0.35, the edge (X1, Y2) in matched measurements is omitted (see Figure 2). As a result, in this example, we consider only the following three edges which are (X1, Y1), (X2, Y2), and (X2, Y1).
After adding the nodes and edges in the graph, we used the ***max_weight_matching*** technique from the ***networkx.algorithms.matching*** which returns a set of matched metric pairs, with the highest sum of matching scores, without duplication. In this example, there are two groups of metric pairs without repetition after using the cut-off level of 0.35. The edges (X1, Y1) and (X2, Y2) make up the first group, whereas the edge (X2, Y1) make up the second. The first group's sum of matching scores is 1.3 (=0.8+0.5), whereas the second group's is 0.4. The matching scores of the first group (1.3) is higher than the second (0.4). As a result, in this example, we choose the first metric pair as the set of matched measurements for the specified source and target metrics.

3. Collection of Data

We gathered datasets from past studies that were publicly available. All dataset groups used in our trials are listed in Table 1. As seen in the table, each dataset category has a diverse collection of metrics. Prediction The granularity of instances predicted in the table's last column is referred to as granularity. We used 19 defect datasets from three different groups: ReLink, NASA, and SOFTLAB.

Group	Dataset	No. of instances		No. of metrics
		All	Buggy (%)	
ReLink	Apache	194	98 (50.52%)	26
	Safe	56	22 (39.29%)	
	Zxing	399	118 (29.57%)	
NASA	cm1	344	42 (12.21%)	37
	mw1	264	27 (10.23%)	
	pc1	759	61 (8.04%)	
	pc3	1125	140 (12.44%)	
	pc4	1399	178 (12.72%)	
	jm1	9593	1759 (18.34%)	21
	pc2	1585	16 (1.01%)	36
	pc5	17001	503 (2.96%)	38
	mc1	9277	68 (0.73%)	
	mc2	127	44 (34.65%)	39
	kc3	200	36 (18.00%)	
SoftLab	ar1	121	9 (7.44%)	29
	ar3	63	8 (12.70%)	
	ar4	107	20 (18.69%)	
	ar5	36	8 (22.22%)	
	ar6	101	15 (14.85%)	

ReLink datasets were used to improve defect prediction performance by improving defect data quality and extracting 26 code complexity characteristics using the Understand tool.
The **NASA** group contains proprietary datasets from NASA. We used 11 NASA datasets available in the PROMISE repository. As indicated in Table 1, some NASA datasets have multiple metric sets. The PROMISE repository provided us with cleaned NASA datasets (DS′ version).
The **SOFTLAB** group contains proprietary datasets from a Turkish software company. We used all SOFTLAB datasets in the PROMISE repository for the SOFTLAB group. Halstead and McCabe's cyclomatic metrics are used by both NASA and SOFTLAB groups, however NASA has additional complexity measures such as parameter count and proportion of comments.

The prediction of flaws is done using a variety of datasets. For example, in ReLink, we built an Apache prediction model and tested it on ar3 in SoftLab. Some NASA datasets contain distinct metric sets, such as (cm1→jm1), we also did cross prediction between those NASA datasets. From these 19 datasets, we have a total of 342 potential prediction possibilities.

4. Experimental Design

For the machine learning algorithm, we use three commonly used classifiers namely, Logistic regression, Random Forest and Support vector machine. There is no perfect model for every particular problem so we tested the model using all three of these classifiers and the accuracy results shown are the maximum out of these three.

The Logistic Regression implementation is available as the ***LogisticRegression*** function in the ***sklearn.ensemble*** library. The Random Forest implementation is available as the ***RandomForestClassifier*** function in the ***sklearn.linear_model*** library. The Support vector machine implementation is available as the ***SVC*** function in the ***sklearn.svm*** library.

In the implementation of each of these classifiers, we follow the same procedure. First, we split the source dataset into training and testing data by using the ***trian_test_split*** function in the ***sklearn.model_selection*** library. Then, we fit the training data into the model using ***model.fit*** and passing the metric columns and the class column ("Defects" as discussed in section 2.1). Then we predict the classes of the test data and find the ***accuracy_score*** using the ***sklearn.metrics*** library. The accuracy score is the percentage of the correct predictions in the predicted classes.

Similarly, we use our fitted model to predict the classes of the target dataset by using ***model.predict.*** We needed to make sure that the target metrics are in the same order as the source metrics that they are matched with so that their classes can be predicted. Then we find the ***accuracy_score*** just as we did with the test data of the source dataset.

5. Results and Conclusion

Given below are the results of our prediction model in predicting the target defect proneness. The results show that our model produces statistically significant results when transferring defect knowledge to a target dataset of different metrics from the source dataset.

Source Dataset	Target Dataset	Model accuracy on source test data (%)	Model accuracy on target data (%)
cm1	mc1	88	97
cm1	mc2	92	97
cm1	kc3	90	80
cm1	pc5	87	74
cm1	jm1	88	78
cm1	pc2	87	97
mc1	cm1	98	87
mc1	jm1	98	78
mc1	ar1	97	92
mc1	ar3	97	89
mc1	ar4	97	81
mc1	ar5	97	78
mc1	ar6	98	85
pc5	Apache	72	60
pc5	Safe	74	73

pc5	Zxing	75	69
mc2	Apache	60	86
mc2	Safe	78	75
mc2	Zxing	86	67
Zxing	pc5	71	73
Zxing	ar6	70	87
Zxing	kc3	70	81
Safe	jm1	82	75
Safe	pc2	82	83
Safe	ar1	70	91
Apache	mw1	69	65
Apache	ar3	66	73
ar1	cm1	86	87
ar1	jm1	86	78
ar1	Zxing	86	70
ar3	Safe	89	67
ar4	Safe	84	76
ar5	Apache	90	52
ar5	kc3	90	79
ar6	Safe	90	60

From the above results we can conclude that the performance of our model in predicting defects is comparable to other existing defect prediction techniques. Our model is working effectively in predicting faults in datasets with distinct or heterogeneous datasets.
However, there are some source and target pairs that do not produce even a single matched metric pair with the cut-off of 0.05 in the KSAnalyser for example in pc3→ Apache the maximum p-value in the KSAnalyser is 0.013 which is less than the cut-off of 0.05 so no source and target metrics are matched. We choose the cut-off as 0.05 because it is one of the most commonly accepted significance level in statistical tests.
The NASA dataset cm1 proved effective in predicting the SoftLab group whereas, the NASA dataset mc2 was the most effective in predicting all the projects in the ReLink group.
The SoftLab group datasets proved effecting in predicting the datasets in the NASA group but ineffective in predicting the ReLink group.
The ReLink datasets predict all the datasets in the SoftLab group effectively but the accuracy score for predicting NASA datasets is average.

This demonstrates the scalability of our model by allowing us to easily exceed the target coverage limitation by fitting prediction models to different existing defect datasets as source until we reach 100% coverage.
A training dataset with a greater defect ratio may cause bias since its prediction performance may be better than that of a dataset with a lower defect ratio. We analysed the best and worst source datasets that led to the best and worst ***accuracy_score*** values, respectively, to see if our model is affected by the training dataset defect ratio and what creates better prediction performance.
In our experiments, we discovered that our model does not suffer from the defect ratios of datasets, and that prediction performance is largely dependent on the defect-proneness tendency of matched metrics. This statement is backed up by the fact that mc1 is the best source dataset, despite its low defect ratio of 0.73%.

ar1 (0.320), with a defect percentage of 7.44 percent, is the worst source dataset. We discovered that the matched metrics of ar1 had an uneven defect-proneness between source and target, resulting in noisy metric matching. The constant defect proneness tendency of matched measurements between source and target datasets is most significant to lead to superior prediction performance, as evidenced by these best and worst situations.

Previously, cross-project defect prediction was not possible due to the varied metric sets used in the projects. We presented heterogeneous defect prediction in our model based on metric matching utilising statistical analysis to alleviate this constraint. Our tests revealed that the proposed models are viable and produce good outcomes. In addition, we looked at the bottom bounds of the size of the source and target datasets for successful defect prediction transfer learning. Based on our empirical and analytical investigations, we can show that for certain types of data sets, 50 examples are sufficient to create a defect predictor and apply our model.

Our model has a lot of promise because it allows all heterogeneous software project datasets to be utilised for defect prediction on new projects or projects with no defect data. Furthermore, it might not be restricted to defect prediction. This strategy has the potential to be used in all prediction and recommendation-based software engineering challenges.

References

J. Nam, W. Fu, S. Kim, T. Menzies & L. Tan, “Heterogeneous Defect Prediction”, *IEEE Transactions on Software Engineering*, 2017.

K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, “Choosing software metrics for defect prediction: An investigation on feature selection techniques,” Softw. Pract. Exper., vol. 41, no. 5, pp. 579–606, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.1043

IEEE-1012, “IEEE standard 1012-2004 for software verification and validation,” 1998.

https://scikit-learn.org/stable/modules/feature_selection.html

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html

https://networkx.org/documentation/stable/reference/algorithms/bipartite.html

http://promise.site.uottawa.ca/SERepository/datasets-page.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html