1 Data Preprocessing

The datasets are usually in tabular format, with rows representing software instances (e.g., code files or bug reports) and columns representing different features or attributes. The features in the PROMISE repository datasets include Static code metrics, Dynamic code metrics, Software change metrics, Complexity metrics, and Other software-related metrics capture different aspects of software quality and maintainability.

Table 1
Attribute Information of JM1 Dataset

No.	Feature Name	Description
1.	loc	: numeric % McCabe's line count of code
2.	v(g)	: numeric % McCabe "cyclomatic complexity"
3.	ev(g)	: numeric % McCabe "essential complexity"
4.	iv(g)	: numeric % McCabe "design complexity"
5.	n	: numeric % Halstead total operators + operands
6.	v	: numeric % Halstead "volume"
7.	l	: numeric % Halstead "program length"
8.	d	: numeric % Halstead "difficulty"
9.	i	: numeric % Halstead "intelligence"
10.	e	: numeric % Halstead "effort"
11.	b	: numeric % Halstead
12.	t	: numeric % Halstead's time estimator
13.	lOCode	: numeric % Halstead's line count
14.	lOComment	: numeric % Halstead's count of lines of comments
15.	lOBlank	: numeric % Halstead's count of blank lines
16.	lOCodeAndComment	: numeric
17.	uniq_Op	: numeric % unique operators
18.	uniq_Opnd	: numeric % unique operands
19.	total_Op	: numeric % total operators
20.	total_Opnd	: numeric % total operands
21.	branchCount	: numeric % of the flow graph
22.	defects	: {false, true} % module has/has not one or more % reported defects

The PROMISE repository datasets are extensively used to train and evaluate ML models for software defect prediction, identify patterns and trends in software issues, analyze software quality metrics, and benchmark different software engineering techniques and tools. The datasets are publicly available and can be accessed by researchers and practitioners for academic and non-commercial purposes.

We examined 33 datasets from two publicly accessible corpora. One (JM1) is from NASA's creators (Sayyad Shirabad and Menzies, 2005), and another is the Jira repository (Yatish et al., 2019), which contains different versions of the Java project. JM1 is written in "C," a real-time predictive ground system that uses simulations to generate predictions. The data comes from McCabe and Halstead's extractors of source code. These features were defined

Table 2
Before Data Balancing

Dataset	#Files	Model	ABC	P	R	F1	AUC	AAC	SM
jruby-1.1	511	XGB	0.95	0.39	0.41	0.40	0.77	0.95	XGB
jruby-1.4.0	684	RF	0.91	0.19	0.21	0.19	0.84	0.94	XGB
jruby-1.5.0	791	DT	0.94	0.15	0.16	0.15	0.86	0.96	XGB
jruby-1.7.0.preview1	**1129**	**RF**	**0.96**	**0.52**	**0.39**	**0.44**	**0.72**	**0.96**	**RF**
hive-0.9.0	991	RF	0.96	0.34	0.29	0.30	0.80	0.95	RF
hive-0.10.0	1092	DT	0.95	0.48	0.54	0.50	0.85	0.94	RF
hive-0.12.0	**1863**	**RF**	**0.97**	**0.40**	**0.43**	**0.42**	**0.78**	**0.97**	**RF**
hbase-0.94.0	741	RF	0.83	0.12	0.13	0.12	0.82	0.85	RF
hbase-0.95.0	**1168**	**RF**	**0.94**	**0.16**	**0.15**	**0.14**	**0.84**	**0.92**	**RF**
hbase-0.95.2	1283	XGB	0.84	0.21	0.16	0.16	0.82	0.85	XGB
groovy-1_5_7	**529**	**SVM**	**0.96**	**0.96**	**0.96**	**0.96**	**0.50**	**0.97**	**SVM, RF**
groovy-1_6_BETA_1	574	RF	0.98	0.62	0.70	0.64	0.73	0.96	DT
groovy-1_6_BETA_2	618	RF	0.96	0.29	0.27	0.27	0.82	0.96	XGB
derby-10.2.1.6	1374	DT	0.75	0.12	0.13	0.12	0.85	0.79	XGB
derby-10.3.1.4	1544	RF	0.79	0.13	0.11	0.11	0.87	0.80	RF
derby-10.5.1.1	**1893**	**RF**	**0.89**	**0.16**	**0.15**	**0.14**	**0.86**	**0.89**	**RF**
camel-1.4.0	1060	RF	0.94	0.28	0.30	0.29	0.78	0.94	RF
camel-2.9.0	4984	XGB	0.98	0.28	0.29	0.28	0.84	0.98	XGB
camel-2.10.0	5539	XGB	0.98	0.37	0.35	0.34	0.86	0.98	RF, XGB
camel-2.11.0	**6192**	**DT**	**0.99**	**0.44**	**0.41**	**0.42**	**0.85**	**0.99**	**RF, DT**
activemq-5.0.0	1318	RF	0.94	0.27	0.24	0.25	0.83	0.92	RF
activemq-5.1.0	1379	XGB	0.97	0.34	0.29	0.29	0.82	0.97	XGB
activemq-5.2.0	1428	RF	0.95	0.19	0.18	0.17	0.87	0.95	RF
activemq-5.3.0	1656	RF	0.93	0.24	0.25	0.24	0.79	0.92	RF
activemq-5.8.0	**2394**	**RF**	**0.97**	**0.28**	**0.28**	**0.27**	**0.84**	**0.97**	**RF**
wicket-1.3.0-beta2	1234	RF	0.97	0.34	0.36	0.35	0.77	0.97	RF, XGB
wicket-1.3.0-beta-1	1170	XGB	0.97	0.28	0.31	0.29	0.81	0.97	XGB
wicket-1.5.3	**1804**	**DT**	**0.98**	**0.67**	**0.48**	**0.49**	**0.68**	**0.98**	**RF**
lucene-2.3.0	563	XGB	0.92	0.36	0.34	0.34	0.79	0.90	XGB
lucene-2.9.0	957	RF	0.90	0.35	0.26	0.26	0.82	0.91	RF
lucene-3.0.0	935	RF	0.95	0.48	0.42	0.43	0.78	0.95	RF, XGB
lucene-3.1	**1964**	**XGB**	**0.98**	**0.45**	**0.48**	**0.46**	**0.78**	**0.98**	**XGB**

Accuracy (Before CV): ABC, Accuracy (After CV): AAC, Precision: P, Recall: R, F1 Score: F1, wicket-1.3.0-incubating-beta-1: wicket-1.3.0-beta-1, Selected Model: SM

in the 70s in an attempt to characterize code features that are associated with software quality objectively. The JM1 has 22 attributes (5 different lines of code measure, 3 McCabe metrics, 4 base Halstead measures, 8 derived Halstead measures, a branch count, and 1 goal field). In addition, we obtained 32 defect prediction datasets from a publicly available corpus containing various versions of Java projects. This corpus includes several metrics for individual files from any project: Static code analysis tools were used to extract 54 code metrics, while the Git version control system was used to extract five process metrics and six ownership metrics. Furthermore, labels `Clean` or `Defect` are assigned to each file based on whether or not they were affected by an issue-fixing commit in the subsequent release.

We used two approaches to feature selection to identify the most relevant metrics for bug prediction: automatic and manual. The process of selecting features using an algorithm is called "automatic feature selection." As in previous studies (Yatish et al., 2019), the AutoSpearman method was used in this study. **Manual** feature selection means that we chose a few features (i.e., three, five, seven, and ten) that we thought were the most interpretable based on our prior knowledge. This AutoSpearman alternative was considered because au-

tomatic selection frequently results in many features (20 to 30), which could impede interpretability. Nonetheless, the methodology described in model selection ensures that simple manual selection will be considered only if it results in better cross-validation performance.

The data considered in this work is highly imbalanced, with a few software modules exhibiting defects and many modules being defect-free. Imbalanced datasets can produce suboptimal classification models that perform well in the majority class. Examples from the minority class, on the other hand, are frequently misclassified, if not entirely dismissed, as noise (He and Garcia, 2009; López et al., 2013). As a result, we could only naively evaluate our models using the NASA dataset if we conducted extensive data exploration (Tan et al., 2015). As a result, we used a method called Synthetic Minority Oversampling Technique (SMOTE) (Tantithamthavorn et al., 2018). As a result, we used the SMOTE technique to balance the data 50/50. (i.e., half of the modules have defects, and the other half represents clean modules.)

Data pre-processing is an important ML step involving cleaning, transforming, and preparing the raw data before applying data balancing techniques. Data balancing techniques are used to address the issue of class imbalance, where one class is significantly under-represented compared to others in the dataset. Before applying data balancing techniques in Table 2, it is crucial to pre-process the data to ensure accurate and reliable results properly.

Table 3
Summary: Before Data Balancing

Dataset	#Files	Model	ABC	P	R	F1	AUC	AAC	SM
jruby-1.7.0.preview1	**1129**	**RF**	**0.96**	**0.52**	**0.39**	**0.44**	**0.72**	**0.96**	**RF**
hive-0.12.0	**1863**	**RF**	**0.97**	**0.40**	**0.43**	**0.42**	**0.78**	**0.97**	**RF**
hbase-0.95.0	**1168**	**RF**	**0.94**	**0.16**	**0.15**	**0.14**	**0.84**	**0.92**	**RF**
groovy-1_5_7	**529**	**SVM**	**0.96**	**0.96**	**0.96**	**0.96**	**0.50**	**0.97**	**SVM, RF**
derby-10.5.1.1	**1893**	**RF**	**0.89**	**0.16**	**0.15**	**0.14**	**0.86**	**0.89**	**RF**
camel-2.11.0	**6192**	**DT**	**0.99**	**0.44**	**0.41**	**0.42**	**0.85**	**0.99**	**RF, DT**
activemq-5.8.0	**2394**	**RF**	**0.97**	**0.28**	**0.28**	**0.27**	**0.84**	**0.97**	**RF**
wicket-1.5.3	**1804**	**DT**	**0.98**	**0.67**	**0.48**	**0.49**	**0.68**	**0.98**	**RF**
lucene-3.1	**1964**	**XGB**	**0.98**	**0.45**	**0.48**	**0.46**	**0.78**	**0.98**	**XGB**

Accuracy (Before CV): ABC, Accuracy (After CV): AAC, Precision: P, Recall: R, F1 Score: F1, Selected Model: SM

Properly pre-processing the data before applying data balancing techniques helps to ensure that the data is clean, transformed, and properly split for model training and evaluation, leading to more reliable and accurate results in software defect prediction or any other ML task. The summarized Table 3 shows the accurate result regarding the best-performing ML model.

We have calculated all model performances across all datasets before and after data balancing techniques. However, it's common in research to summarize or aggregate results from multiple experiments or data points to provide a clearer overview or a comparative analysis. For instance, Table 3 presents a summary of statistic derived from the individual results in Table 2, providing a higher-level understanding. Table 2 presents 32 dataset performances, where

every dataset version shows its performance in the corresponding ML model. In Table 3, we just picked the best-performed dataset for each version of the dataset to summarize overall dataset performance. For example, the dataset "jruby" has four versions or releases (jruby-1.1, jruby-1.4.0, jruby-1.5.0, and jruby-1.7.0.preview1). Among four releases, "jruby-1.7.0.preview1" shows the best performance than others. We select this one in Table 3 to show the summarized view. The same thing happens for all other datasets. By this way, we got hive-0.12.0, hbase-0.95.0, groovy-1.5.7, derby-10.5.1.1, camel-2.11.0, activemq-5.8.0, wicket-1.5.3, and lucene-3.1 in Table 3. We did the same thing for Table 4 and Table 5 in the case of the after-data balancing technique.

Table 4
After Data Balancing

Dataset	#Files	Model	ABC	P	R	F1	AUC	AAC	SM
jruby-1.1	4005	RF	0.95	0.43	0.31	0.33	0.77	0.99	RF
jruby-1.4.0	5040	RF	0.93	0.18	0.19	0.18	0.83	1.0	RF, DT
jruby-1.5.0	5912	XGB	0.93	0.15	0.16	0.15	0.87	1.0	XGB
jruby-1.7.0.preview1	**9603**	**RF**	**0.96**	**0.52**	**0.41**	**0.45**	**0.73**	**1.0**	**RF**
hive-0.9.0	7020	RF	0.96	0.34	0.29	0.30	0.80	1.0	RF
hive-0.10.0	9640	RF	0.95	0.24	0.26	0.25	0.75	0.99	RF
hive-0.12.0	**8575**	**XGB**	**0.97**	**0.38**	**0.41**	**0.40**	**0.80**	**1.0**	**RF, XGB**
hbase-0.94.0	10030	XGB	0.85	0.13	0.18	0.14	0.86	1.0	RF, XGB
hbase-0.95.0	**10800**	**RF**	**0.94**	**0.15**	**0.14**	**0.13**	**0.83**	**0.99**	**RF**
hbase-0.95.2	16439	XGB	0.84	0.20	0.24	0.21	0.84	0.99	XGB
groovy-1_5_7	**4608**	**DT**	**0.98**	**0.15**	**0.12**	**0.13**	**0.82**	**0.98**	**DT, MLP**
groovy-1_6_BETA_1	5190	XGB	0.97	0.59	0.67	0.59	0.73	1.0	XGB
groovy-1_6_BETA_2	5085	RF	0.96	0.31	0.30	0.31	0.79	1.0	RF
derby-10.2.1.6	19173	RF	0.77	0.10	0.11	0.11	0.85	1.0	RF
derby-10.3.1.4	22512	RF	0.78	0.009	0.10	0.009	0.86	1.0	RF
derby-10.5.1.1	**27693**	**RF**	**0.89**	**0.14**	**0.13**	**0.13**	**0.87**	**0.99**	**RF**
camel-1.4.0	6888	RF	0.93	0.34	0.31	0.32	0.77	0.99	RF
camel-2.9.0	48510	XGB	0.98	0.26	0.27	0.26	0.85	1.0	RF
camel-2.10.0	43024	RF	0.98	0.48	0.29	0.34	0.67	0.99	RF
camel-2.11.0	**48488**	**RF**	**0.99**	**0.44**	**0.43**	**0.43**	**0.80**	**1.0**	**RF**
activemq-5.0.0	14495	RF	0.93	0.17	0.18	0.17	0.83	0.99	RF
activemq-5.1.0	12700	RF	0.97	0.41	0.43	0.32	0.82	1.0	XGB
activemq-5.2.0	15204	XGB	0.96	0.26	0.24	0.23	0.88	1.0	XGB
activemq-5.3.0	25024	RF	0.93	0.23	0.24	0.22	0.83	0.98	RF
activemq-5.8.0	**24750**	**RF**	**0.98**	**0.31**	**0.29**	**0.28**	**0.84**	**1.0**	**RF**
wicket-1.3.0-beta2	8022	RF	0.96	0.34	0.34	0.34	0.76	1.0	RF
wicket-1.3.0-beta-1	8800	XGB	0.97	0.27	0.29	0.28	0.80	1.0	XGB
wicket-1.5.3	**15615**	**RF**	**0.98**	**0.67**	**0.50**	**0.49**	**0.67**	**0.98**	**RF**
lucene-2.3.0	4620	RF	0.92	0.45	0.37	0.38	0.79	1.0	RF
lucene-2.9.0	7670	RF	0.90	0.37	0.31	0.32	0.82	0.99	RF
lucene-3.0.0	7443	RF	0.95	0.48	0.43	0.44	0.76	1.0	RF
lucene-3.1	**15144**	**XGB**	**0.97**	**0.35**	**0.30**	**0.32**	**0.66**	**1.0**	**RF**

Accuracy (Before CV): ABC, Accuracy (After CV): AAC, Precision: P, Recall: R, F1 Score: F1, wicket-1.3.0-incubating-beta-1: wicket-1.3.0-beta-1, Selected Model: SM

2 Experimental Result for all dataset

The Chi-square test and the Correlation matrix with heatmap are applied to all datasets in the Jira repositories as part of the defect prediction scenario analysis. The results indicate varying levels of disagreement between these two methods across different datasets. In general, the Chi-square test

Table 5
Summary: After Data Balancing

Dataset	#Files	Model	ABC	P	R	F1	AUC	AAC	SM
jruby-1.7.0.preview1	**9603**	**RF**	**0.96**	**0.52**	**0.41**	**0.45**	**0.73**	**1.0**	**RF**
hive-0.12.0	**8575**	**XGB**	**0.97**	**0.38**	**0.41**	**0.40**	**0.80**	**1.0**	**RF, XGB**
hbase-0.95.0	**10800**	**RF**	**0.94**	**0.15**	**0.14**	**0.13**	**0.83**	**0.99**	**RF**
groovy-1_5_7	**4608**	**DT**	**0.98**	**0.15**	**0.12**	**0.13**	**0.82**	**0.98**	**DT, MLP**
derby-10.5.1.1	**27693**	**RF**	**0.89**	**0.14**	**0.13**	**0.13**	**0.87**	**0.99**	**RF**
camel-2.11.0	**48488**	**RF**	**0.99**	**0.44**	**0.43**	**0.43**	**0.80**	**1.0**	**RF**
activemq-5.8.0	**24750**	**RF**	**0.98**	**0.31**	**0.29**	**0.28**	**0.84**	**1.0**	**RF**
wicket-1.5.3	**15615**	**RF**	**0.98**	**0.67**	**0.50**	**0.49**	**0.67**	**0.98**	**RF**
lucene-3.1	**15144**	**XGB**	**0.97**	**0.35**	**0.30**	**0.32**	**0.66**	**1.0**	**RF**

Accuracy (Before CV): ABC, Accuracy (After CV): AAC, Precision: P, Recall: R, F1 Score: F1, Selected Model: SM

showed better results (i.e., less disagreement) than the correlation matrix with heatmap for most datasets. For most datasets from Jira repositories, feature selection using the Chi-square test yielded better performance in defect prediction scenarios. Specific datasets such as camel-2.10.0, camel-2.11.0, derby-10.2.1.6, derby-10.5.1.1, hbase-0.94.0, groovy-1.5.7, hive (all versions), jruby-1.5.0, and wicket-1.5.3 exhibited less disagreement in the correlation matrix with heatmap when considering different values of k (e.g., k=7, k=10) for both sign and rank agreement. Although the correlation matrix with heatmap generally showed more disagreement compared to the Chi-square test, there are specific datasets where it performed better. Notably, all versions of the hive dataset consistently showed better results with less disagreement in the correlation matrix with heatmap than the Chi-square test. All experimental results, including those for the Chi-square test and the correlation matrix with heatmap, are available in the replication package. The experimental result (PDF) for both methods can be found inside the replication package in the folder named "Buggy file testing for all dataset". Overall, while the Chi-square test is generally more effective in feature selection for defect prediction across most datasets, the correlation matrix with heatmap yielded better results for specific datasets, such as all versions of the hive dataset. These findings are detailed in the experimental results PDF within the replication package.

Some sample experimental result for The Chi-square test are shown in Figures 1, 2, and 3. And some for The Correlation matrix with Heatmap are shown in Figures 4, 5, and 6. Detailed experimental result are available in replication package.

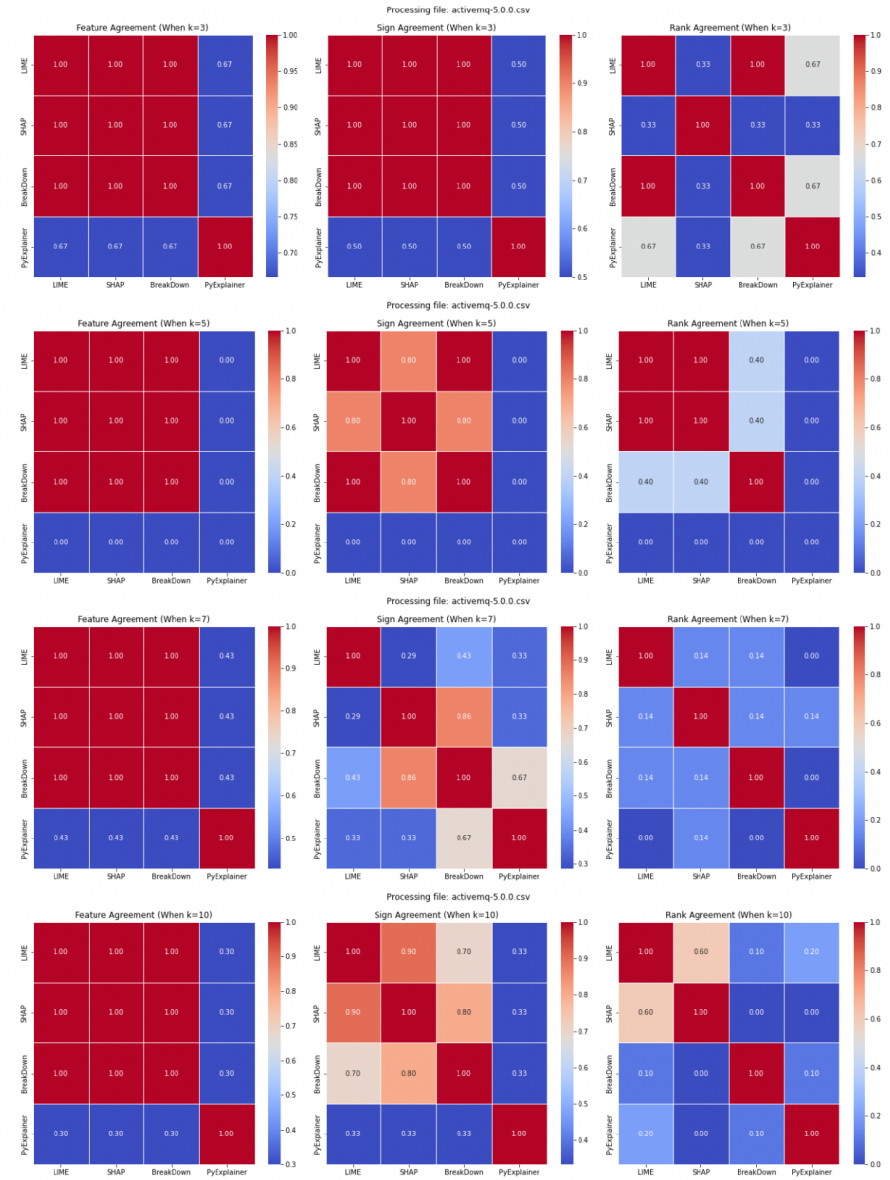

Fig. 1: Chi-square test result for activeMQ 5.0.0 dataset

References

Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.

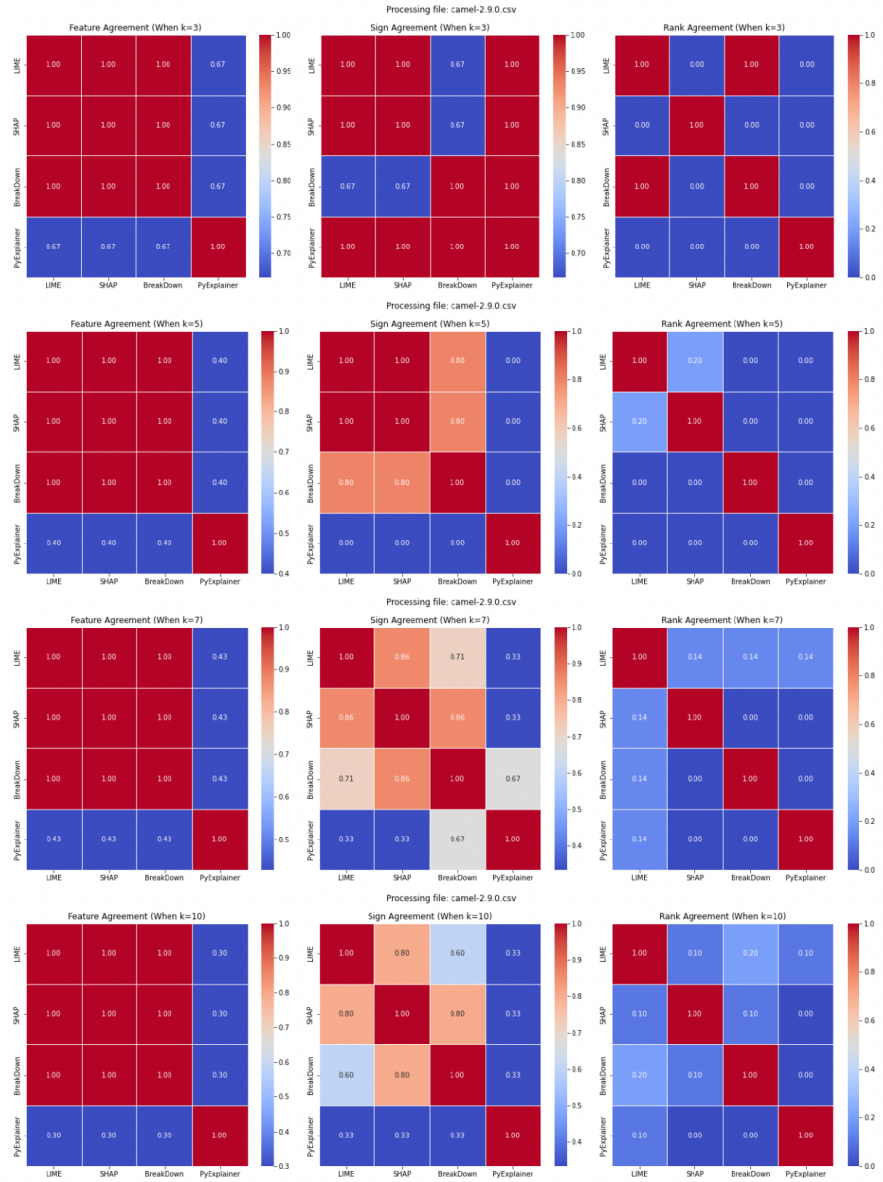

Fig. 2: Chi-square test result for camel 2.9.0 dataset

Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. 2013. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information sciences* 250 (2013), 113–141.

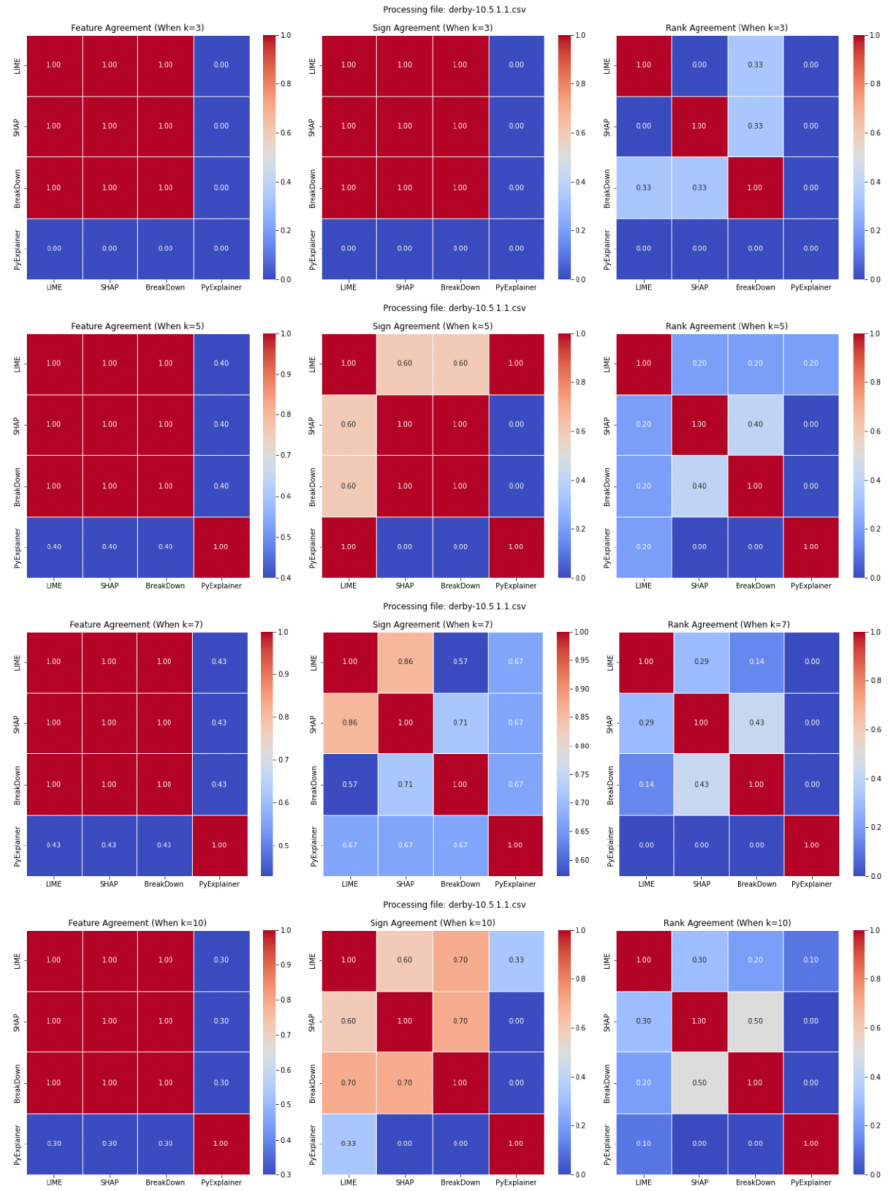

Fig. 3: Chi-square test result for derby 10.5.1.1 dataset

J. Sayyad Shirabad and T.J. Menzies. 2005. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. `http://promise.site.uottawa.ca/SERepository`

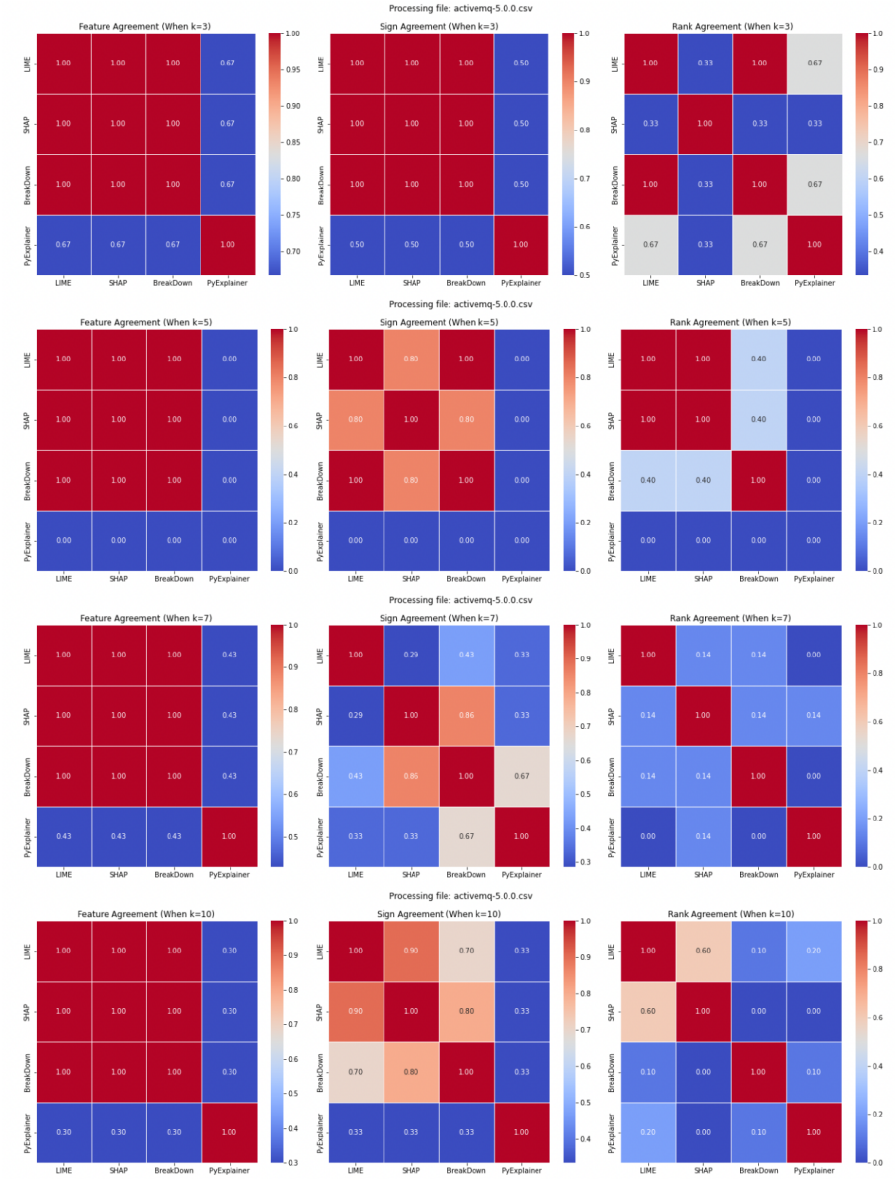

Fig. 4: Correlation matrix with Heatmap test result for activemq 5.0.0 dataset

Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online defect prediction for imbalanced data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 99–108.

Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of class rebalancing techniques on the performance and inter-

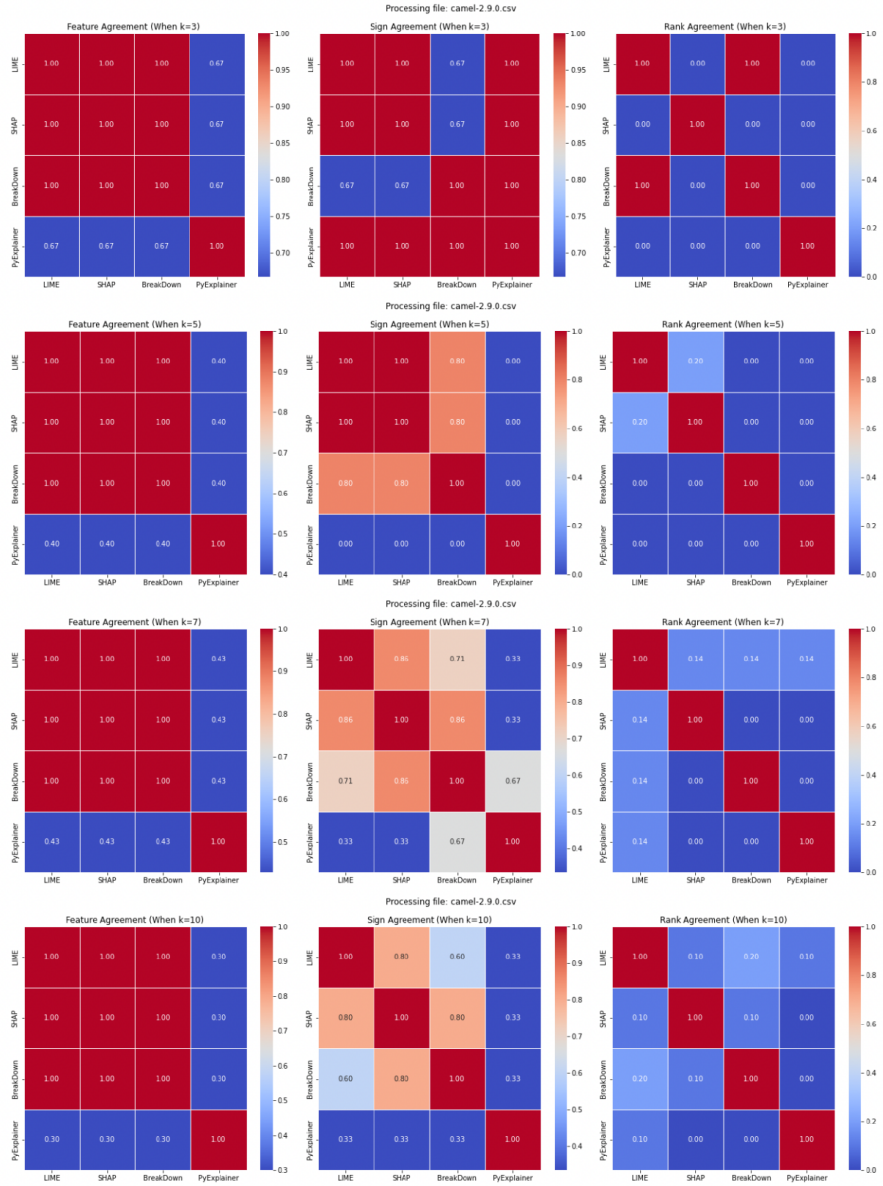

Fig. 5: Correlation matrix with Heatmap test result for camel 2.9.0 dataset

pretation of defect prediction models. *IEEE Transactions on Software Engineering* 46, 11 (2018), 1200–1219.

Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining Software Defects: Should We Consider Affected Releases?. In *2019 IEEE/ACM 41st International Conference on*

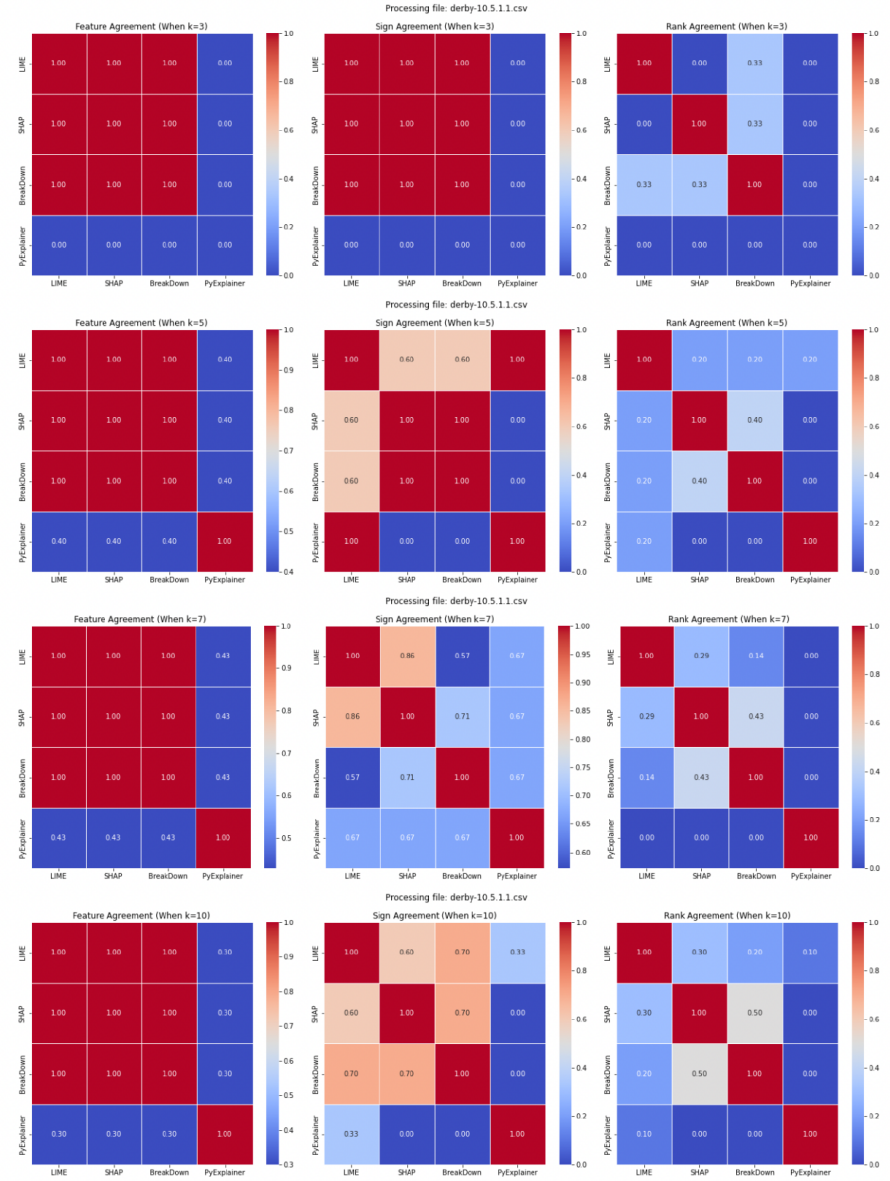

Fig. 6: Correlation matrix with Heatmap test result for derby 10.5.1.1 dataset

Software Engineering (ICSE). 654–665. `https://doi.org/10.1109/ICSE.2019.00075`