DS&AI

มหาวิทยาลัยขอนแก่น
KHON KAEN UNIVERSITY

Universiteit Leiden

Automated Code Smell Detection

Chitsutha Soomlek, College of Computing, Khon Kaen University

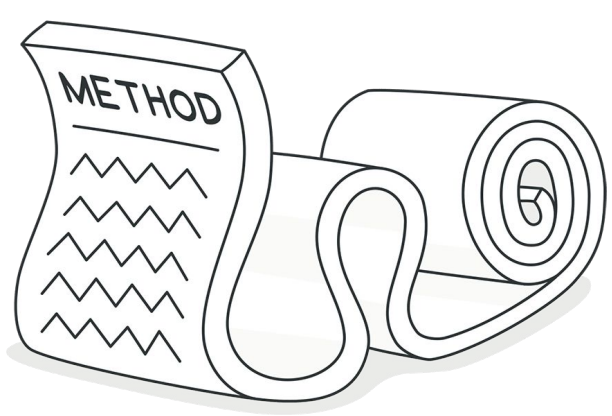
Jan N. van Rijn, LIACS, Leiden University

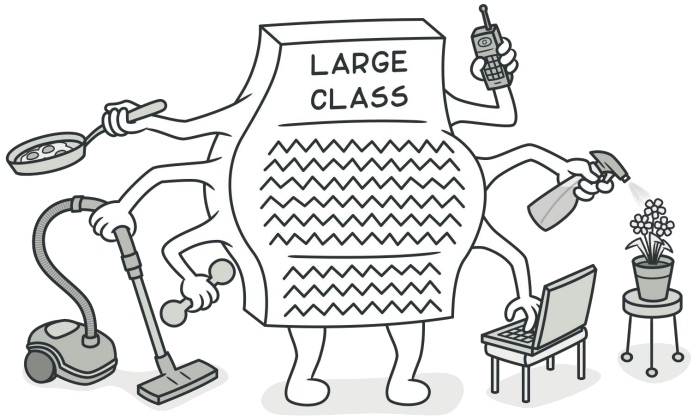
Marcello M. Bonsangue, LIACS, Leiden University

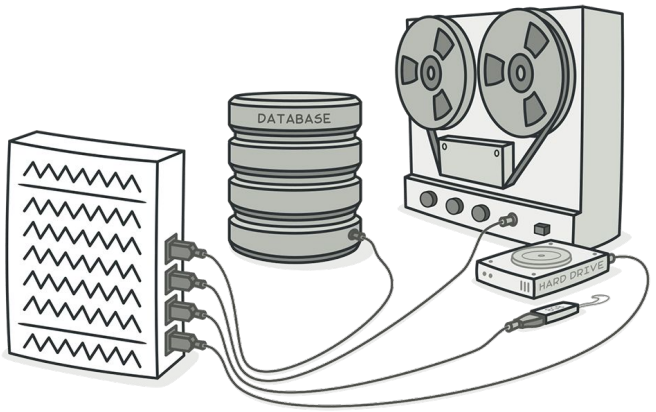
Co-funded by the Erasmus+ Programme of the European Union

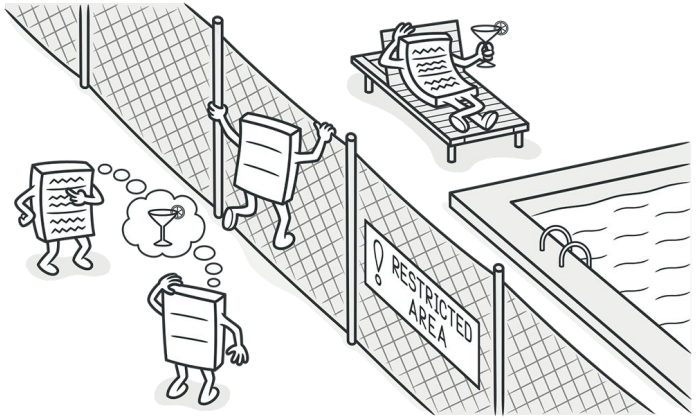
liacs Leiden Institute of Advanced Computer Science

COLLEGE OF COMPUTING KHON KAEN UNIVERSITY









Figures from https://refactoring.guru/

What Exactly is a Code Smell?

A code smell is a surface indication that usually corresponds to a deeper problem in the system… they are often an indicator of a problem rather than the problem themselves".

Kent Beck and Martin Fowler (1999)

Why is it difficult to identify?

- There is no formal definition or standard.
- Developers have different perceptions of code smells.
- Commonly used approaches for code smells detection rely on a fixed set of metrics and corresponding threshold values.
- There are reliability issues of the threshold values.
- Programming languages have been evolving.

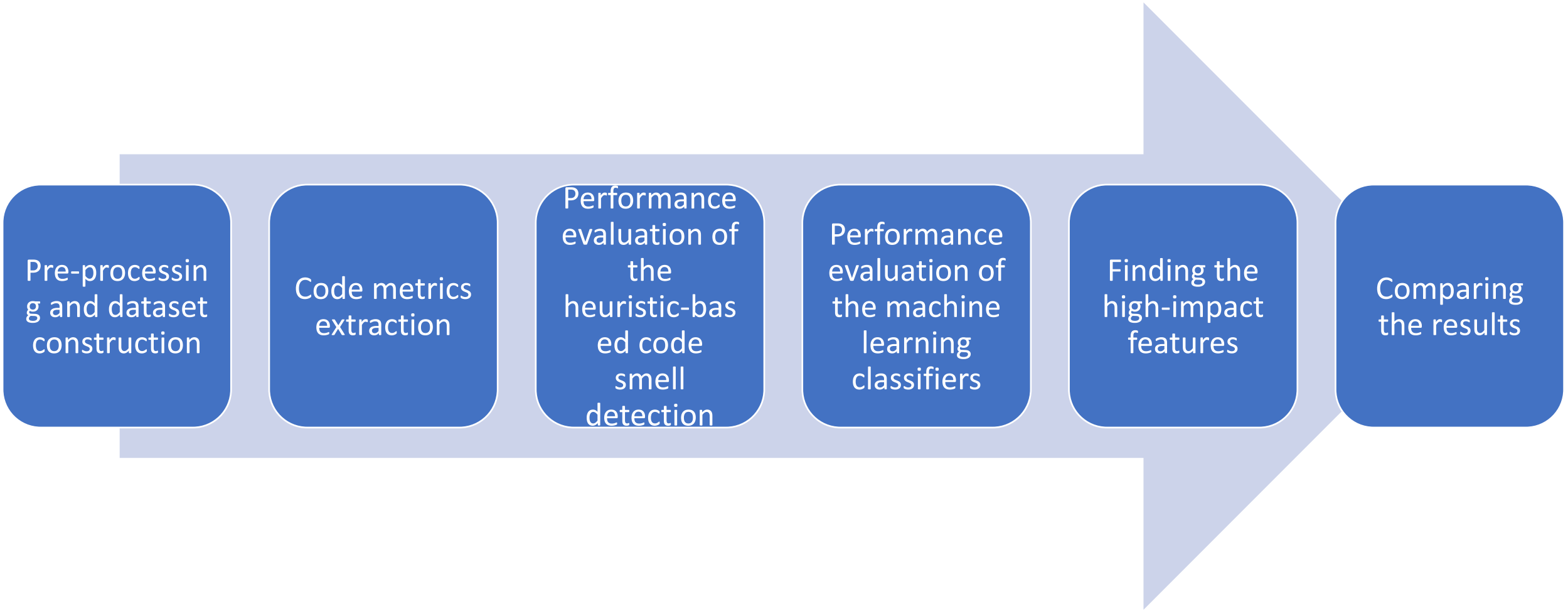
Research Questions

Can we use machine learning to mimic developers' contemporary perception of code smells?

How does machine learning perform when compared to the existing heuristic-based code smell detection?

What are the features contribute most to the classification of code smell instances?

Empirical Study and Evaluation



The MLCQ Dataset

- Madeyski, L., Lewowski, T.: *MLCQ: industry-relevant code smell data set*. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp. 342–347 (2020).

- The characteristics of the published MLCQ dataset
  - No. of project repository = 524 -> Active = 518
  - No. of code smells in Java projects = 14,853
  - No. of reviewers = 26
  - Professional experience in programming of the reviewers = 2-10 years
  - No. reviewed files (path) = 4,610
  - Code smells included = Blob, Data class, Feature envy, and Long method
  - Severity levels = critical, major, minor, and none

Number of reviews on each type of code smells per severity level

Code smells	No. of reviews	Positive			Negative	% of positive instances	No. of code sample with multiple reviews
		Critical	Major	Minor	None		
Blob	4076	129	316	539	3092	24.14	1741
Data class	4078	146	401	510	3021	25.70	2522
Feature Envy	3337	24	142	288	2883	13.61	1030
Long method	3362	78	274	454	2556	23.97	1060

Handling the multiple reviews

Mapping the severity level of a review to a numerical severity score

Calculate the average review score (ARS) for each code sample

Determine the distribution of the number of reviews

Distribution of the number of reviews

No. of Reviews	Blob				Data Class			
	No. of Samples	Average of the average review score	Average standard deviation	Average of the variances	No. of Samples	Average of the average review score	Average standard deviation	Average of the variances
1	1562	0.00000	N/A	N/A	1566	0.00000	N/A	N/A
2	147	0.87755	0.11664	0.01361	147	0.48976	0.00000	0.00000
3	409	0.50448	0.69461	0.48248	411	0.66018	0.69291	0.48013
4	198	0.66162	0.83736	0.70118	196	0.79337	0.78786	0.62075
5	39	0.73333	0.87998	0.77436	39	0.87180	0.88143	0.77692
6	1	2.33333	0.51640	0.26667	1	0.00000	0.00000	0.00000

Distribution of the number of reviews

No. of Reviews	Feature Envy				Long Method			
	No. of Samples	Average of the average review score	Average standard deviation	Average of the variances	No. of Samples	Average of the average review score	Average standard deviation	Average of the variances
1	1954	0.00102	N/A	N/A	1967	0.00051	0.02255	0.00051
2	82	0.19512	0.15617	0.02439	162	0.67593	1.16269	1.35185
3	303	0.42904	0.72894	0.53135	302	0.00331	0.30988	0.09603
4	70	0.72857	0.97101	0.94286	73	0.96233	1.96197	3.84932
5	6	0.53333	0.86603	0.75000	7	0.97143	2.20389	4.85714
6	0	N/A	N/A	N/A	0	N/A	N/A	N/A

**There is considerable disagreement on the reviews of feature envy and long method.

Heuristic-based classifiers

Code smell	Specification	Reference
Blob	$ATFD > 2 \,\&\, WMC \geq 47 \,\&\, TCC < 0.33$	Trifu and Marinescu [56]
	$ATFD > 5 \,\&\, WMC \geq 47 \,\&\, TCC < 0.33$	Olbrich et al. [33] Schumacher et al. [45] Macia et al. [29] and PMD [40][41]
	$ATFD > 3 \,\&\, WMC \geq 47 \,\&\, TCC < 0.33$	Lanza and Marinescu [25]
	$NOM > 15 \,\vert\, NOF > 15$	Kiefer et al. [22]
	$NOM + NOF > 20$	Moha et al. [32]
	$CLOC > 100 \,\vert\, VG > 20$	Liu et al. [28]
	$NOM > 20 \,\vert\, NOF > 9 \,\vert\, CLOC > 750$	Danphitsanuphan and Suwantada [11]
	$LCOM \geq 0.725 \,\&\, WMC \geq 34 \,\&\, NOF \geq 8 \,\&\, NOM \geq 14$	Souza et al. [53]
	$CLOC \geq CLOCV \,\vert\, NOM \geq NOMV$	Rasool et al. [44]
	$(WMC > 47 \,\&\, ATFD > 5 \,\&\, TCC < 0.33) \,\vert\, CLOC > 100 \,\vert\, NOM > 14 \,\vert\, NOF > 8$	Mayvan et al. [4]
	$ATFD > 2 \,\&\, TCC < 0.333 \,\&\, WMC \geq 29.25$	JSpIRIT [17,58]
	1. $LCOM \geq 0.8 \,\&\, NOF \geq 7 \,\&\, NOM \geq 7$ 2. $NOPF \geq 20 \,\vert\, NOM \geq 30 \,\vert\, WMC \geq 100$	DesigniteJava [47,54]

- ATFD = Access to foreign data
- WMC = Weighted method count
- TCC = Tight class cohesion
- NOM = Number of methods in a class
- NOF = Number of fields in a class
- CLOC =Line of code in a class
- VG = McCabe's complexity
- LCOM = Lack of cohesion in methods

Heuristic-based classifiers

Code smell	Specification	Reference
Data class	$(WOC < 0.33) \,\&\, ((NOPA + NOAM > 4) \,\&\, (WMC < 47) \,\vert\, (NOPA + NOAM > 2) \,\&\, (WMC < 31))$	Trifu and Marinescu [51]
	$WOC < 0.33 \,\&\, ((NOAP + NOAM > 3) \,\&\, WMC < 31) \,\vert\, ((NOAP + NOAM > 4) \,\&\, WMC < 47)$	Lanza and Marinescu [24]
	$(WMC < 50) \,\vert\, (LCOM < 0.8)$	Danphitsanuphan and Suwantada [11]
	$LCOM > 2 \,\vert\, NOAM > 10$	Kaur et al. [20]
	$nMc(nf) \geq nMc(nf)V \,\vert\, nMc = 0$	Rasool et al. [40]
	$NSC \leq 1 \,\&\, DIT \leq 2 \,\&\, NOF > 3$	Souza et al. [48]
	$LCOM < 0.8 \,\vert\, NOAM > 2 \,\vert\, WOC < 0.33 \,\vert\, NOAP > 3$	Mayvan et al. [4]
	$(NOPA + NOAM > 3 \,\&\, WMC < 31) \,\vert\, (NOPA + NOAM > 5 \,\&\, WMC < 47)$	PMD [36,37] and DesigniteJava [42,49]
	$WOC < 0.333 \,\&\, ((NOAM + NOPA > 2) \,\&\, WMC \geq 29.25) \,\vert\, ((NOAM + NOPA > 4) \,\&\, WMC < 43.875)$	JSpIRIT [17,53]

- WOC = Weight of Class
- NOPA = Number of public attributes
- NOAM = Number of accessor methods
- nMc(nf) = Number of nonfunctional methods in a class
- WMC = Weighted method count
- LCOM = Lack of cohesion in methods
- NSC = Number of children
- DIT = Depth of Inheritance Tree

Heuristic-based classifiers

Code smell	Specification	Reference
Feature envy	$CBO > 5 \mid LCOM > 2$	Kaur et al. [20]
	$LCOM \geq 0.725$	Souza et al. [47]
Long method	$MLOC > 50 \mid VG > 10$	Liu et al. [27]
	$MLOC > 50$	Danphitsanuphan and Suwantada [11]
	$MLOC \geq 100$	PMD [36,37] and DesigniteJava [41,48]
	$MLOC > 40$	JSpIRIT [17,51]
	$MLOC > 150$	Checkstyle [7]

- CBO = Coupling between objects
- MLOC = Line of code in a method
- VG = McCabe's complexity

Code metric extraction

- SciTools Understand 6.0 (build 1055) (https://www.scitools.com)
- PMD 6.35.0 (https://pmd.github.io)
- DesigniteJava 1.8.6 (https://www.designite-tools.com/designitejava/)

- Total = 72 code metrics
  - Class-level
  - Function-level

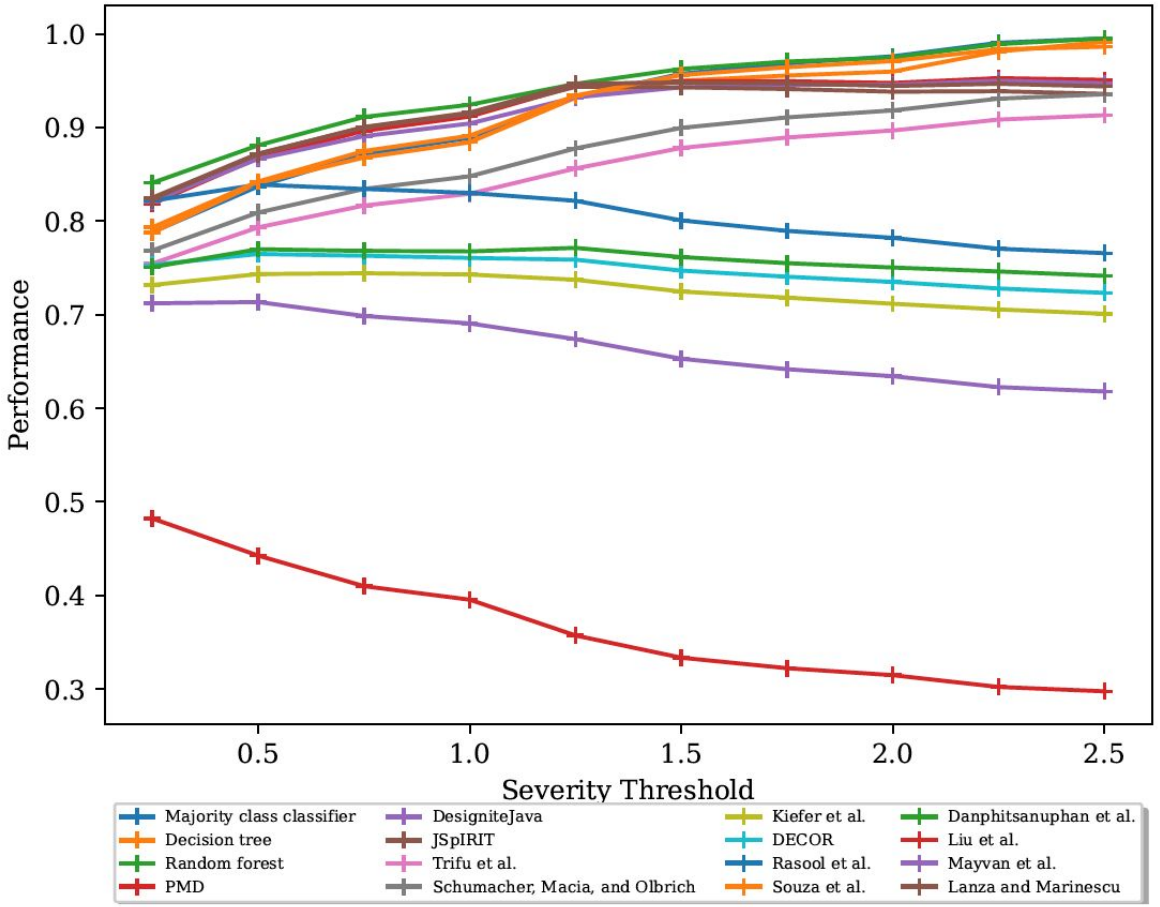
Machine learning based code smell detection

- **Goal:** To mimic human perception of code smell detection.
- **Requirements**:
  - Labelled examples and counterexamples of code smells.
  - Descriptive attributes.
- We built separate models for *blob*, *data class*, *feature envy*, and *long method*.
- The models were evaluated and compared against the heuristic-based classifiers.

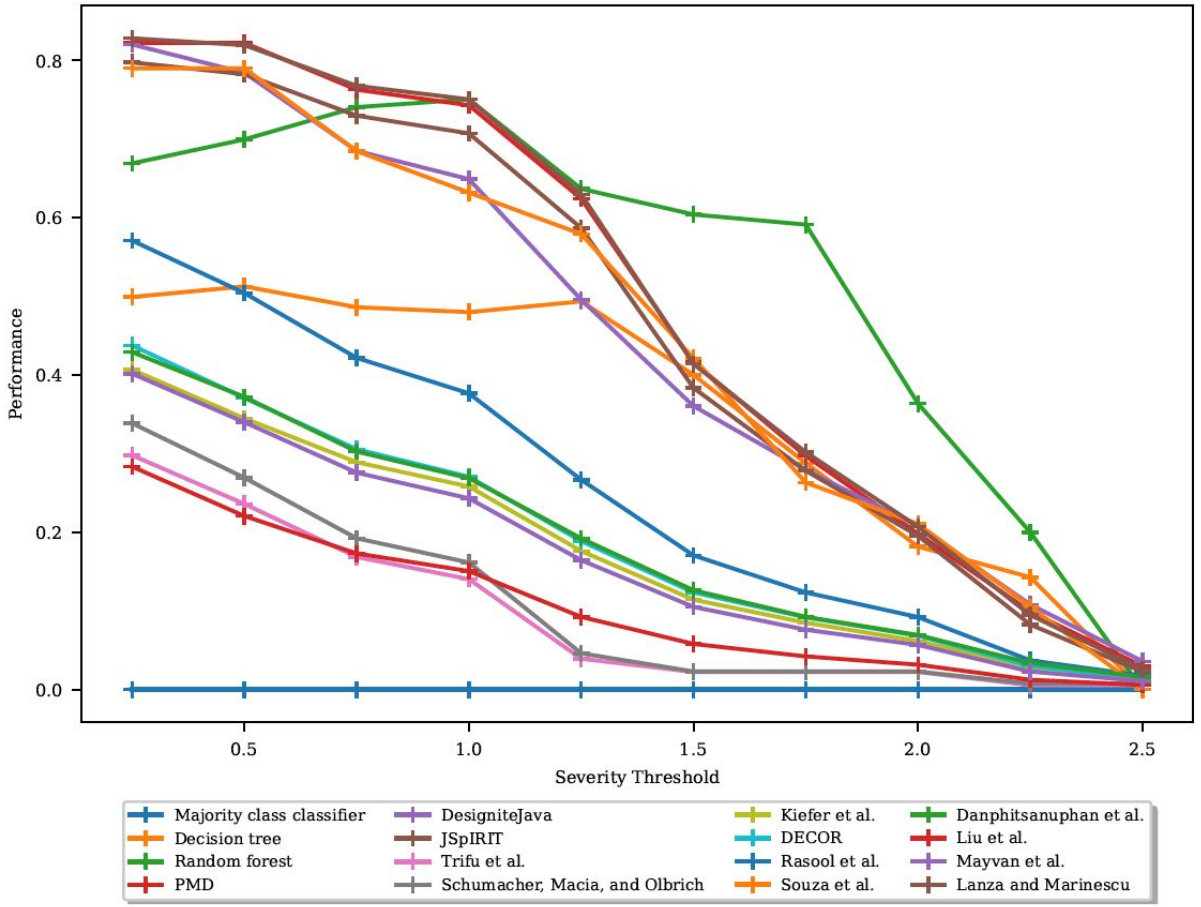
Labelling of code smells

- Human Perception of Code Smells
- Mapping from MLCQ dataset
  - *none* (not a code smell): 0
  - *minor*: 1
  - *major*: 2
  - *critical*: 3
- This allows us to average over multiple reviewers.
- We train a binary classifier
  - Random forest classifier
  - Decision tree classifier
- Question: how to determine a good threshold value?

Accuracy and Precision of Predicting Blob

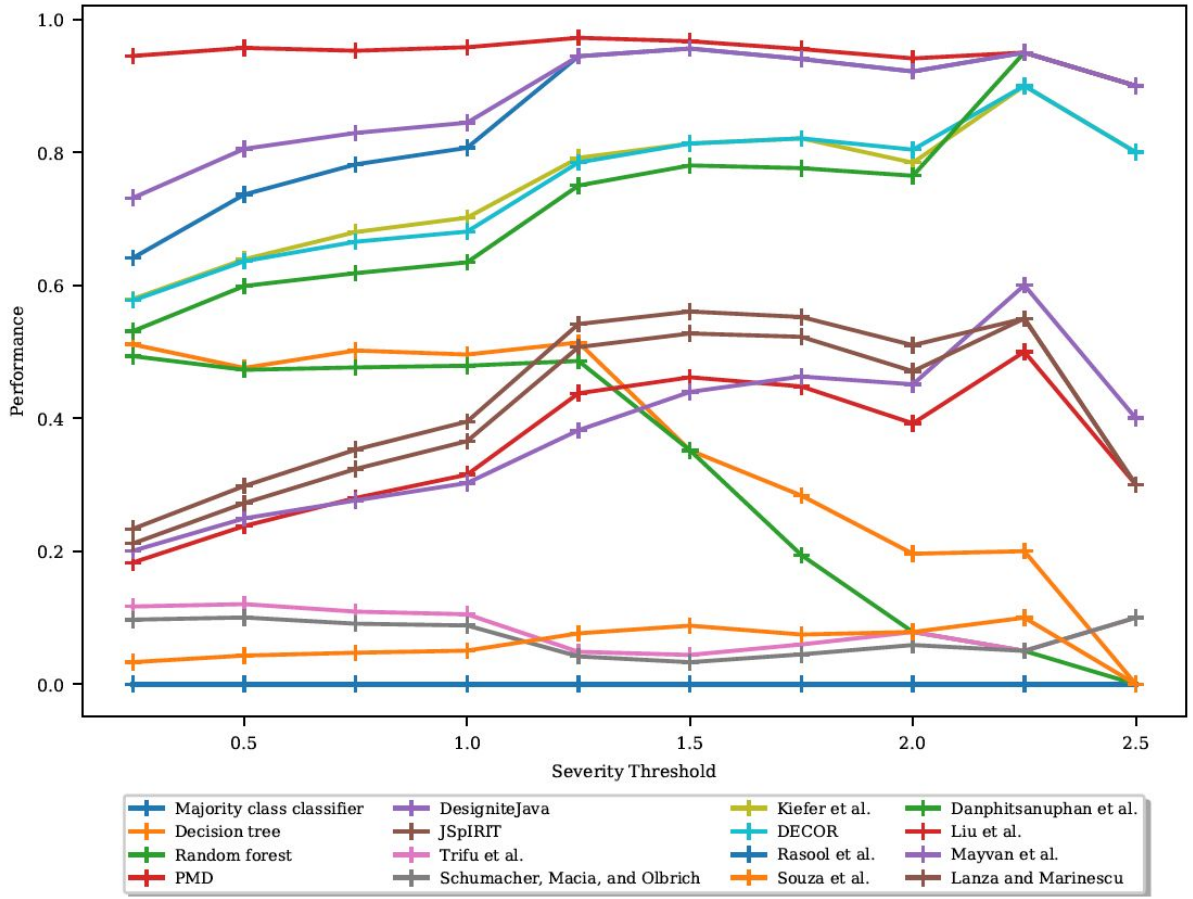


(a) Accuracy

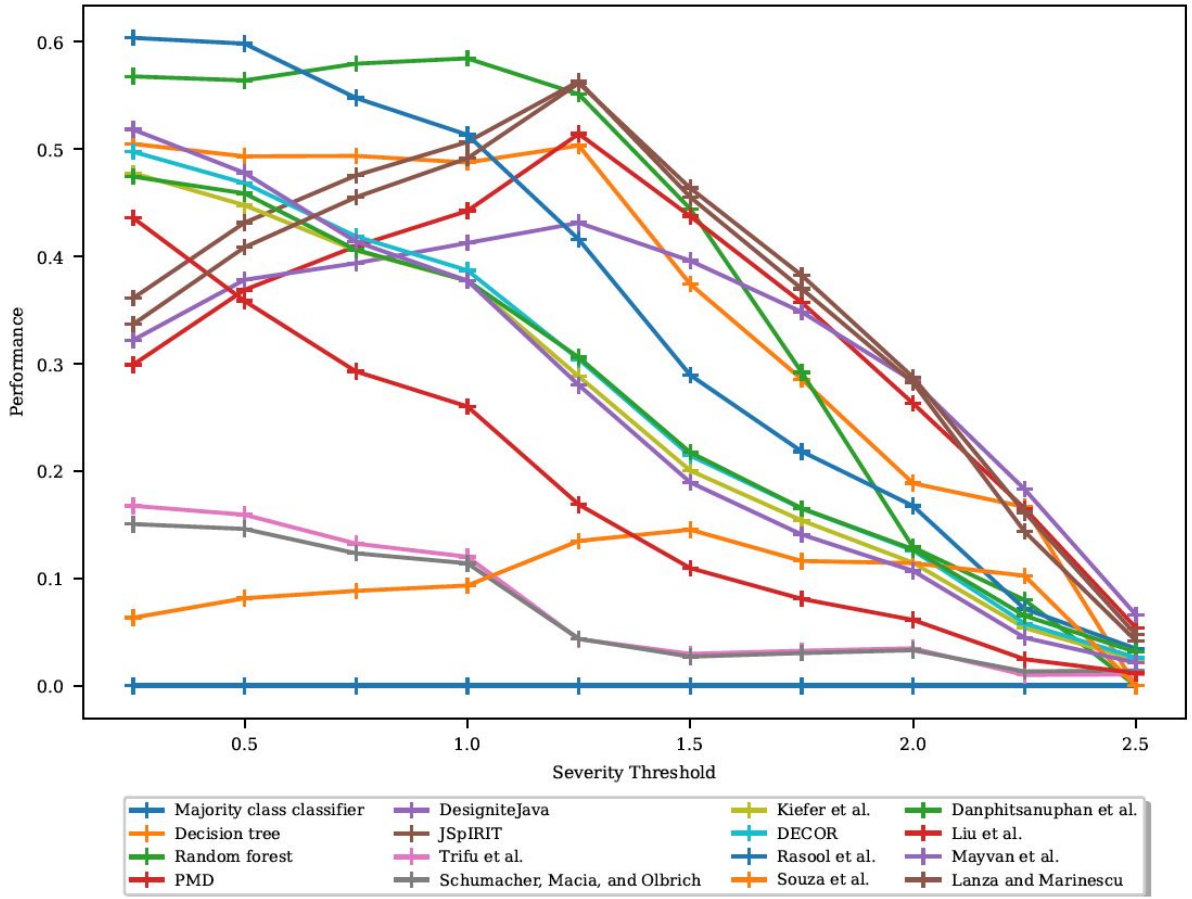


(b) Precision

Recall and F1-Score of Predicting Blob

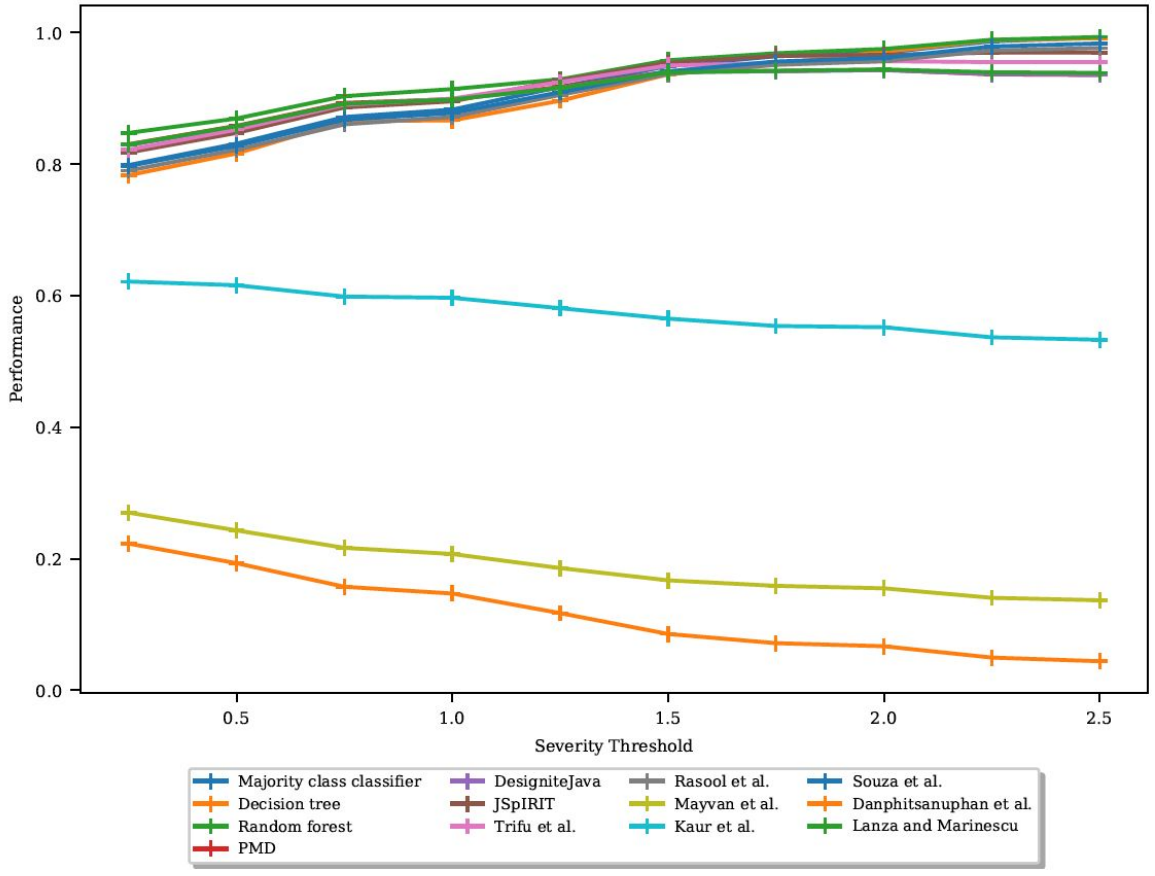


(c) Recall

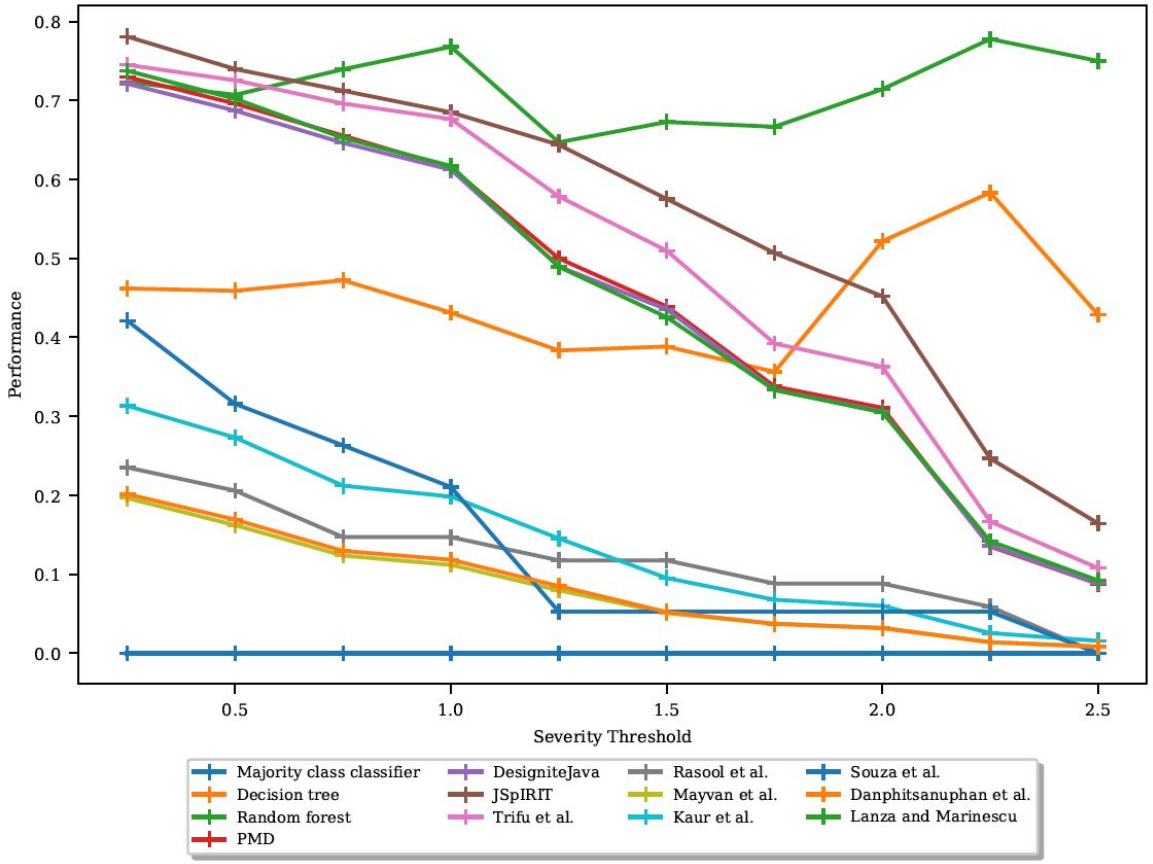


(d) F1-Score

Accuracy and Precision of Predicting Data Class

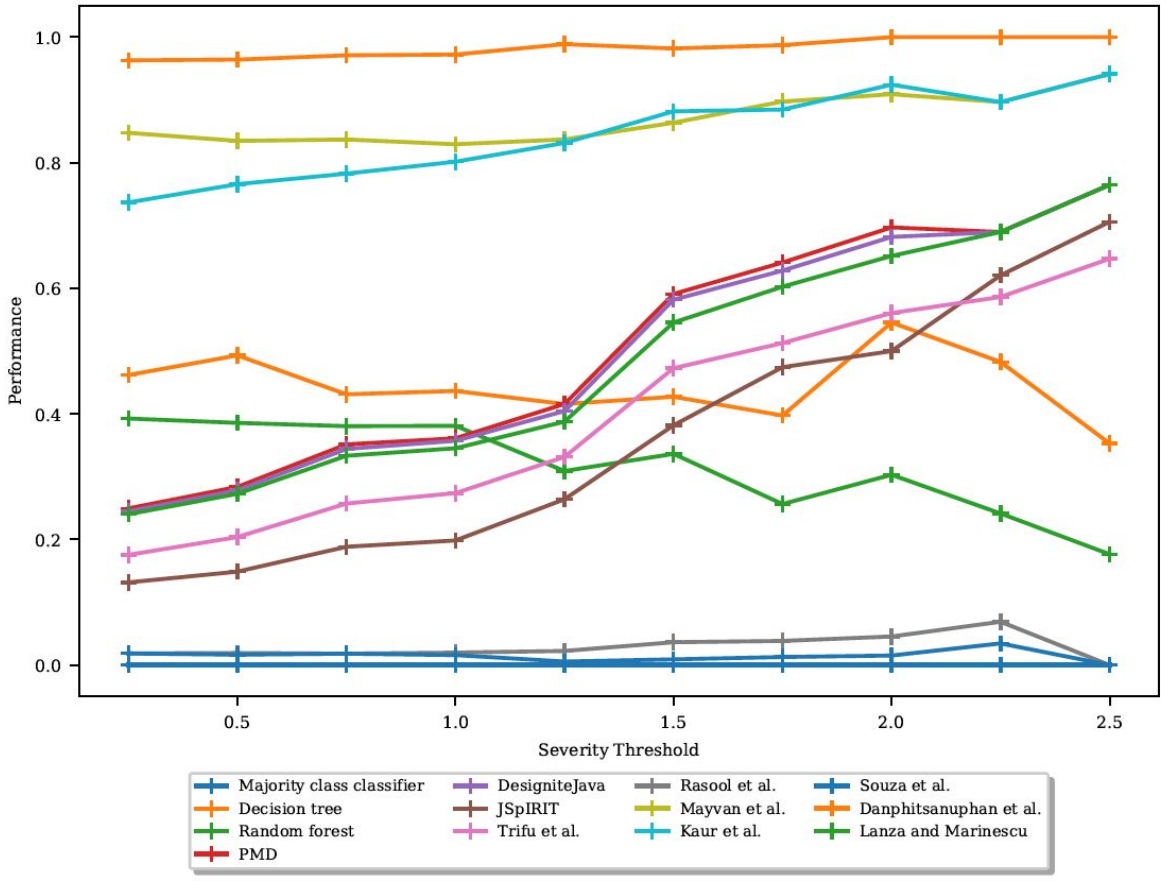


(a) Accuracy

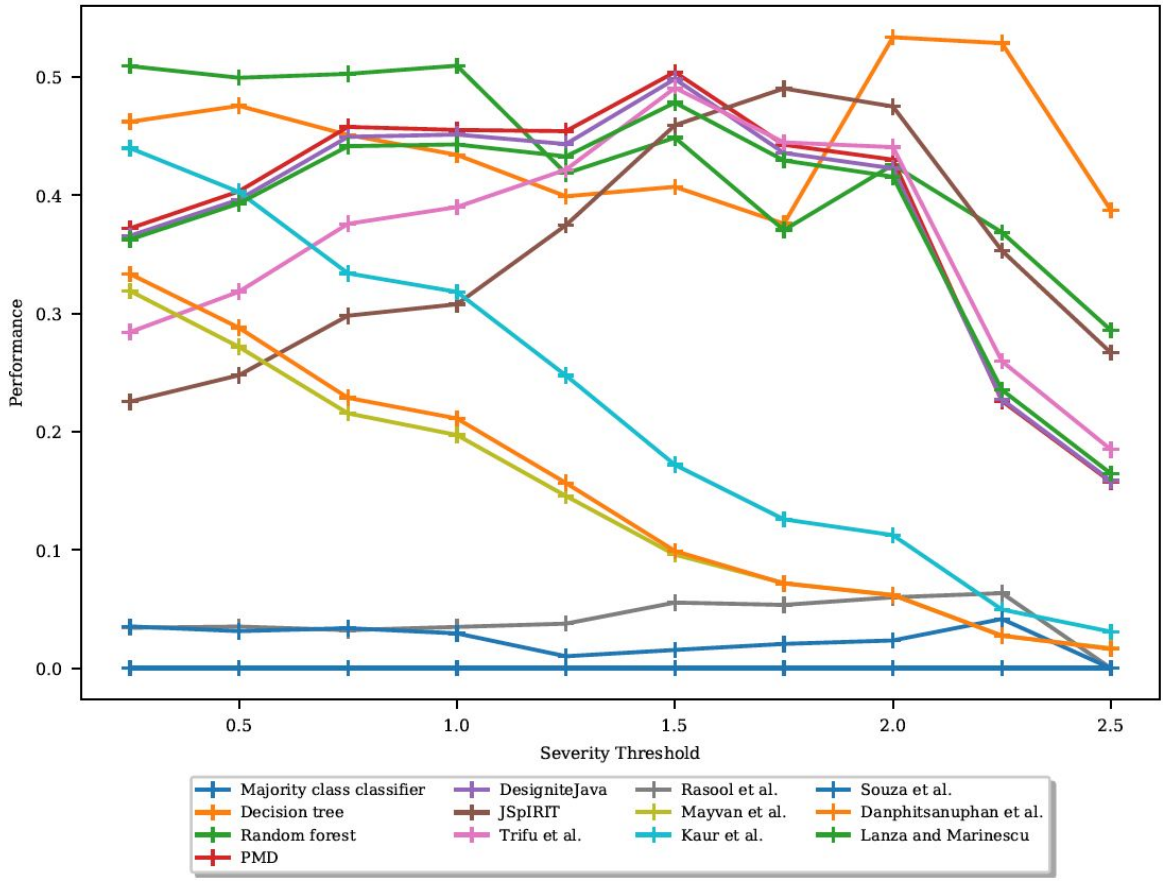


(b) Precision

Recall and F1-Score of Predicting Data Class

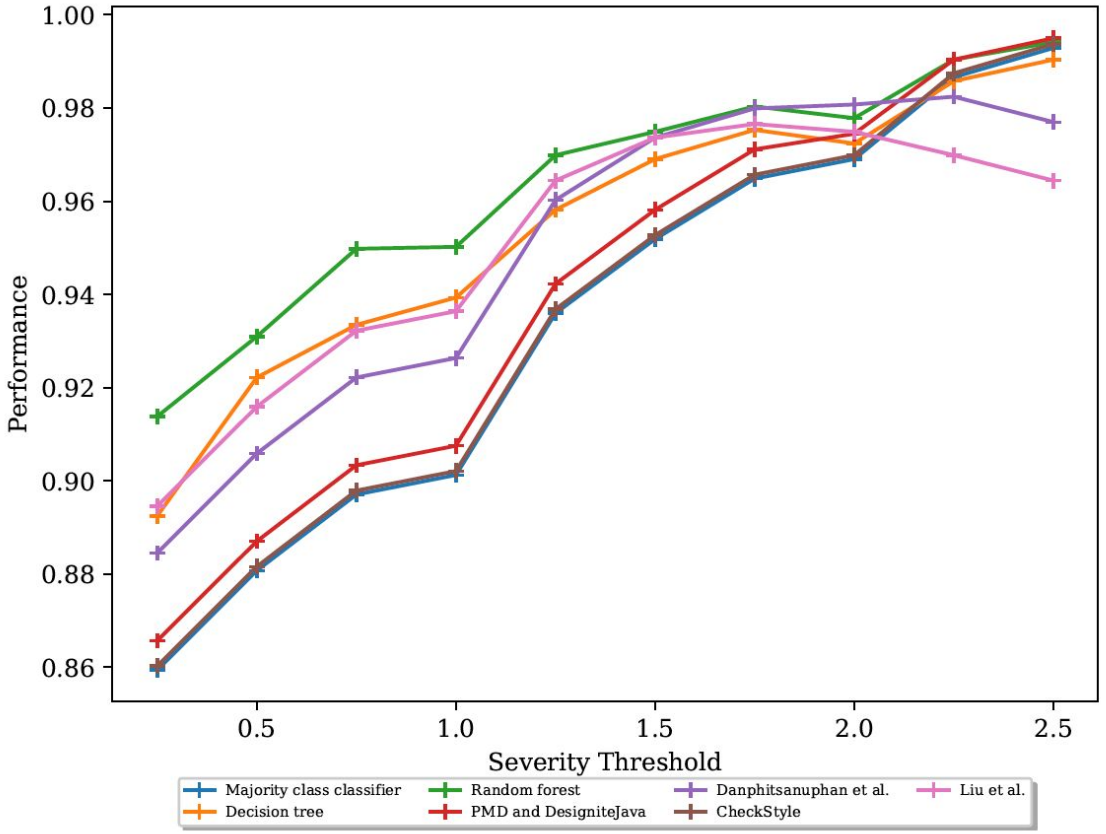


(c) Recall

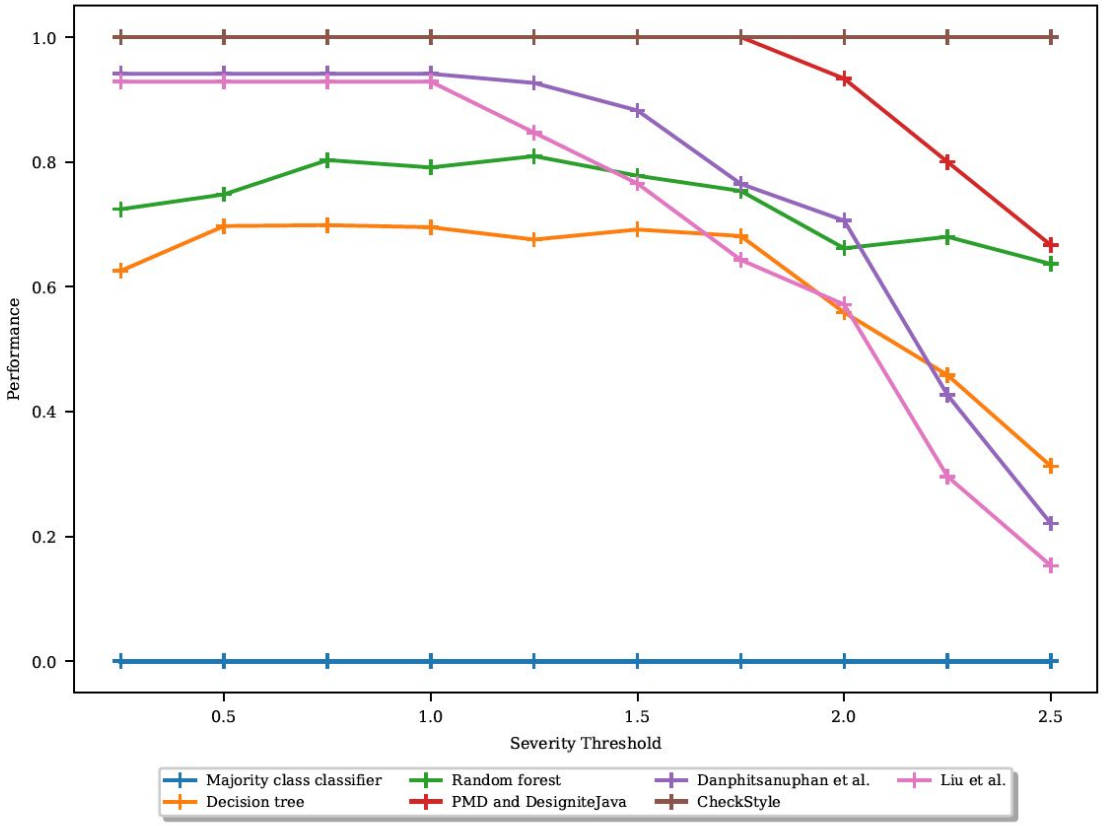


(d) F1-Score

Accuracy and Precision of Predicting Long Method

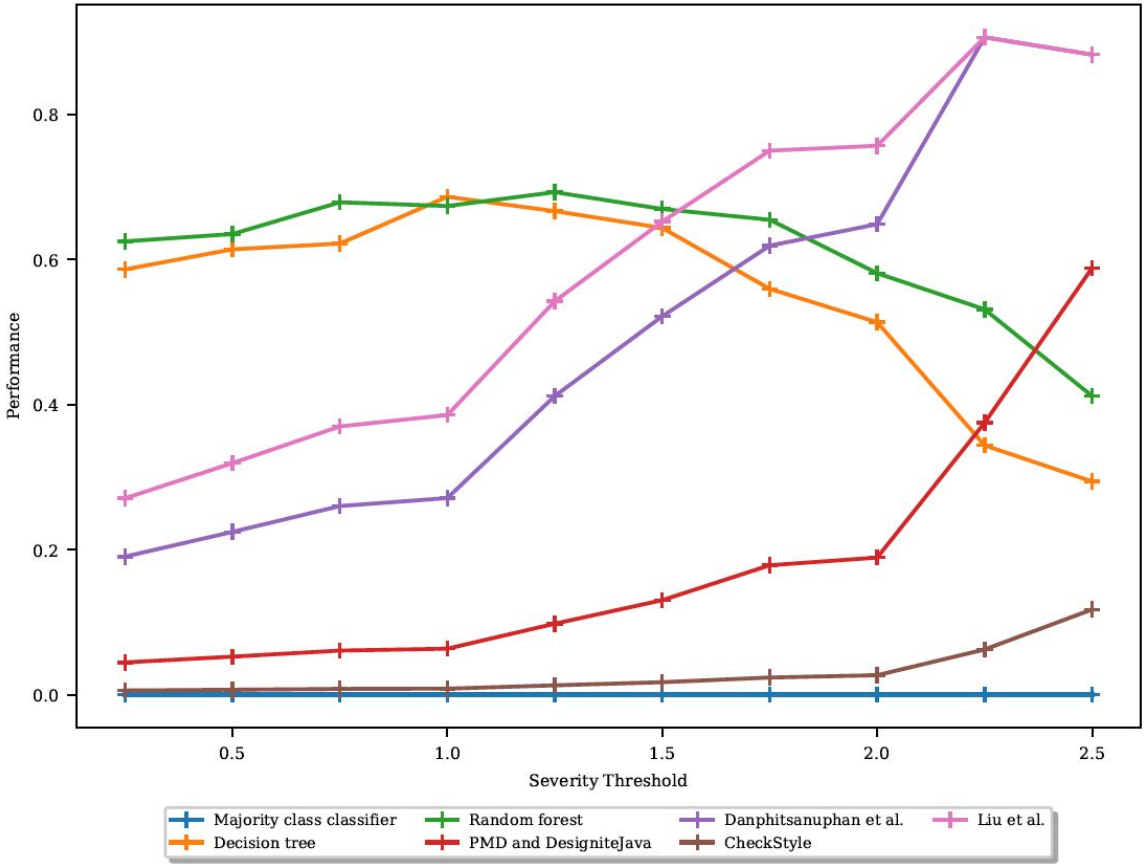


(a) Accuracy

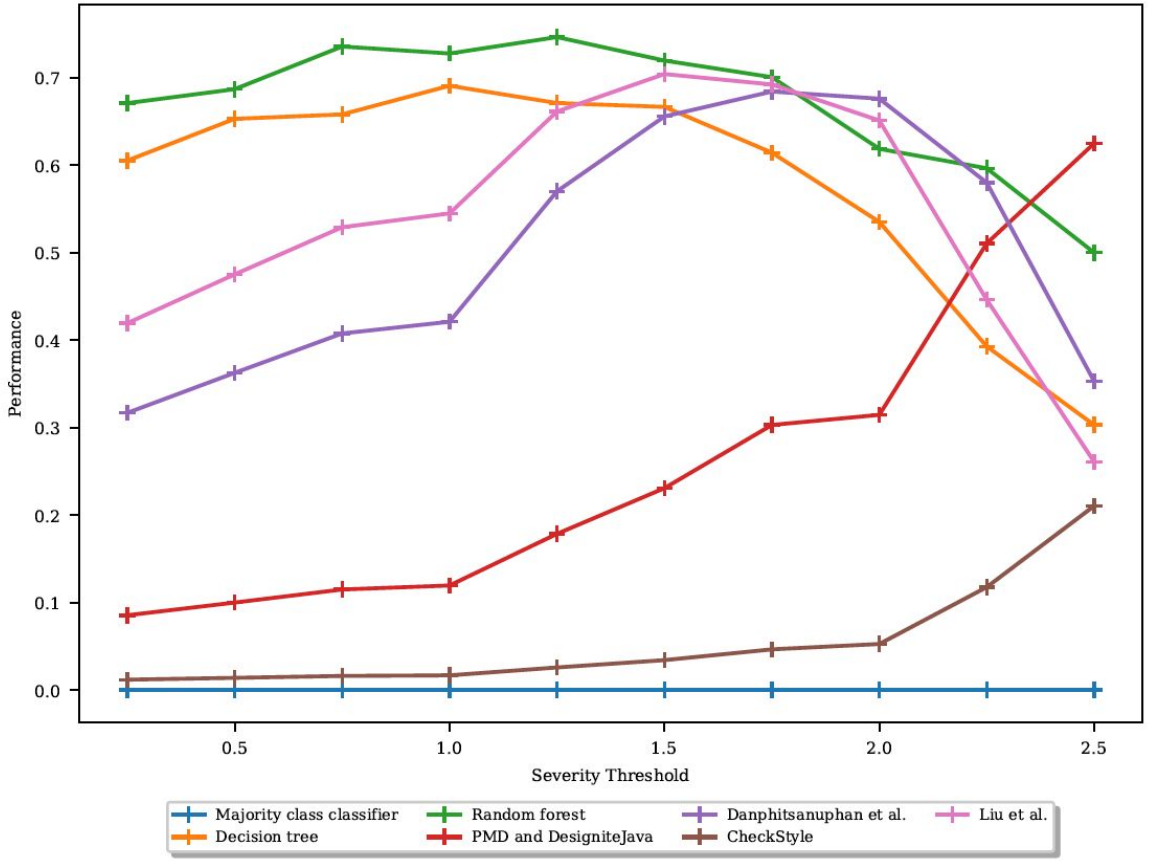


(b) Precision

Recall and F1-Score of Predicting Long Method

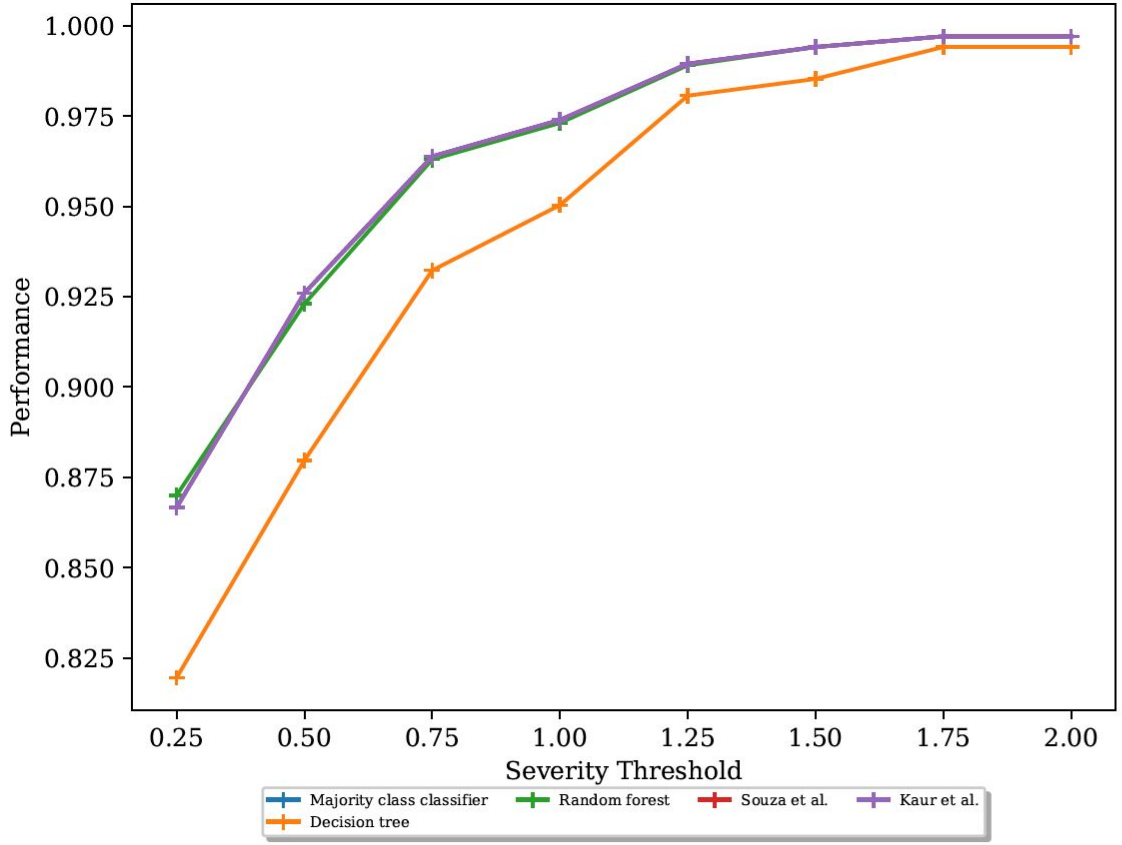


(c) Recall

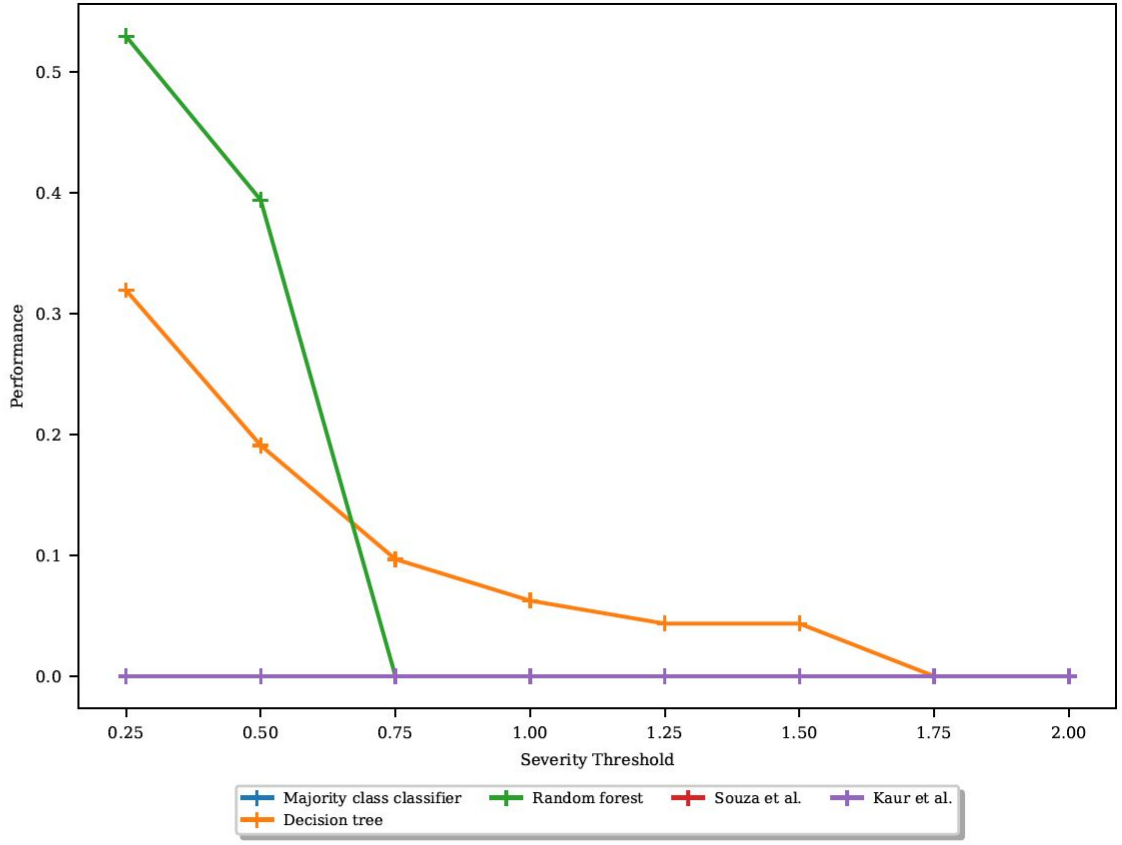


(d) F1-Score

Accuracy and Precision of Predicting Feature Envy

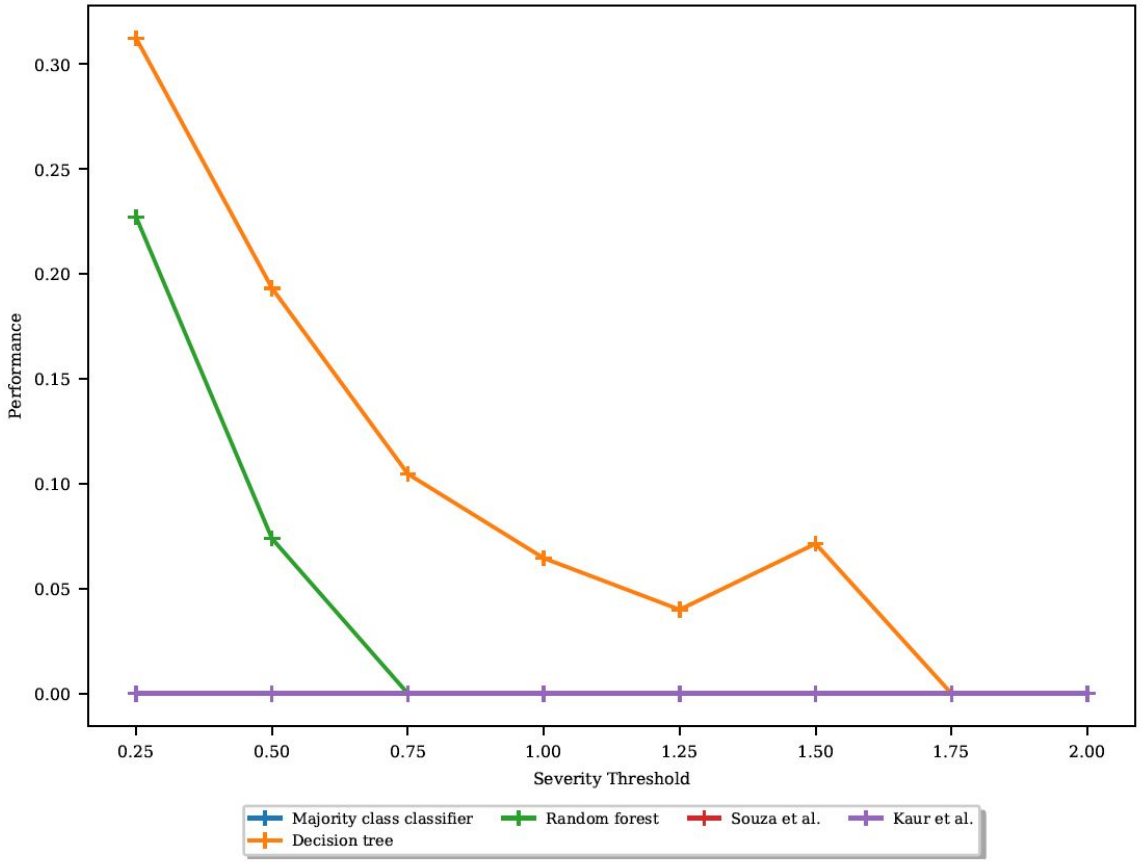


(a) Accuracy

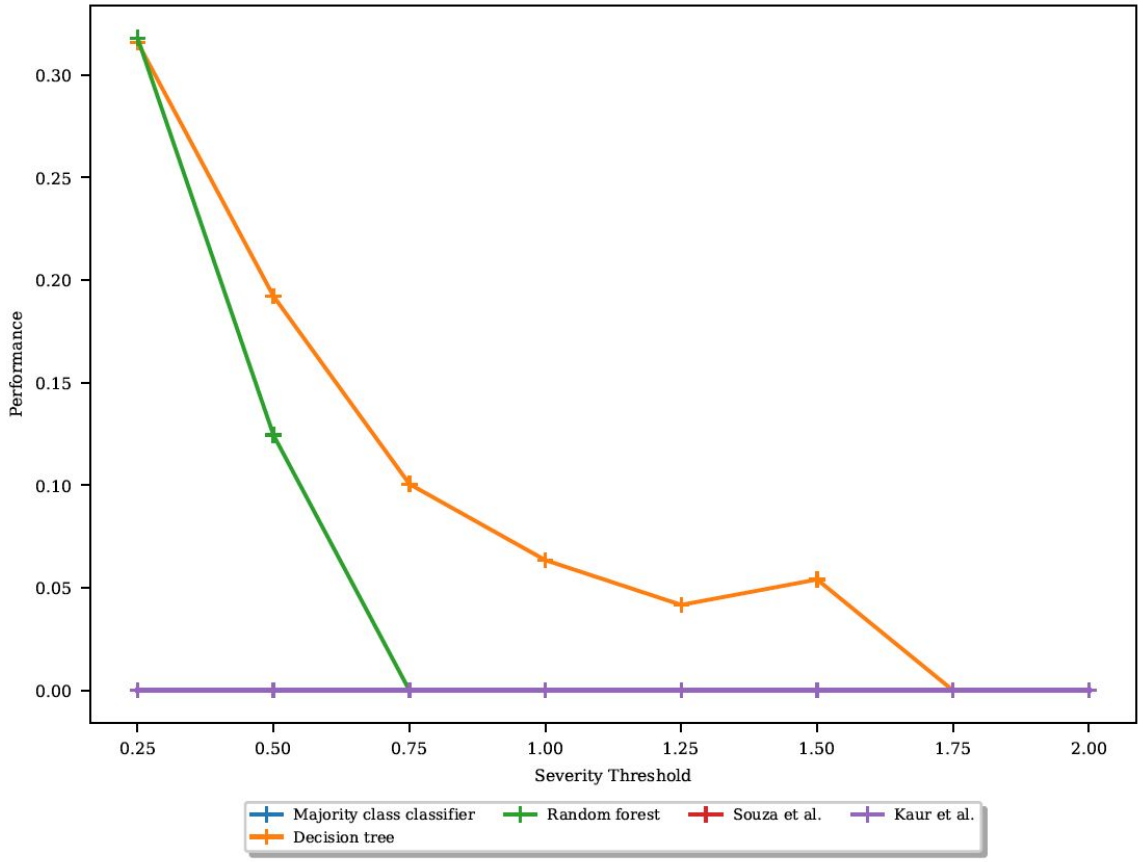


(b) Precision

Recall and F1-Score of Predicting Feature Envy



(c) Recall



(d) F1-Score

Performance of PMD vs. Human Experts

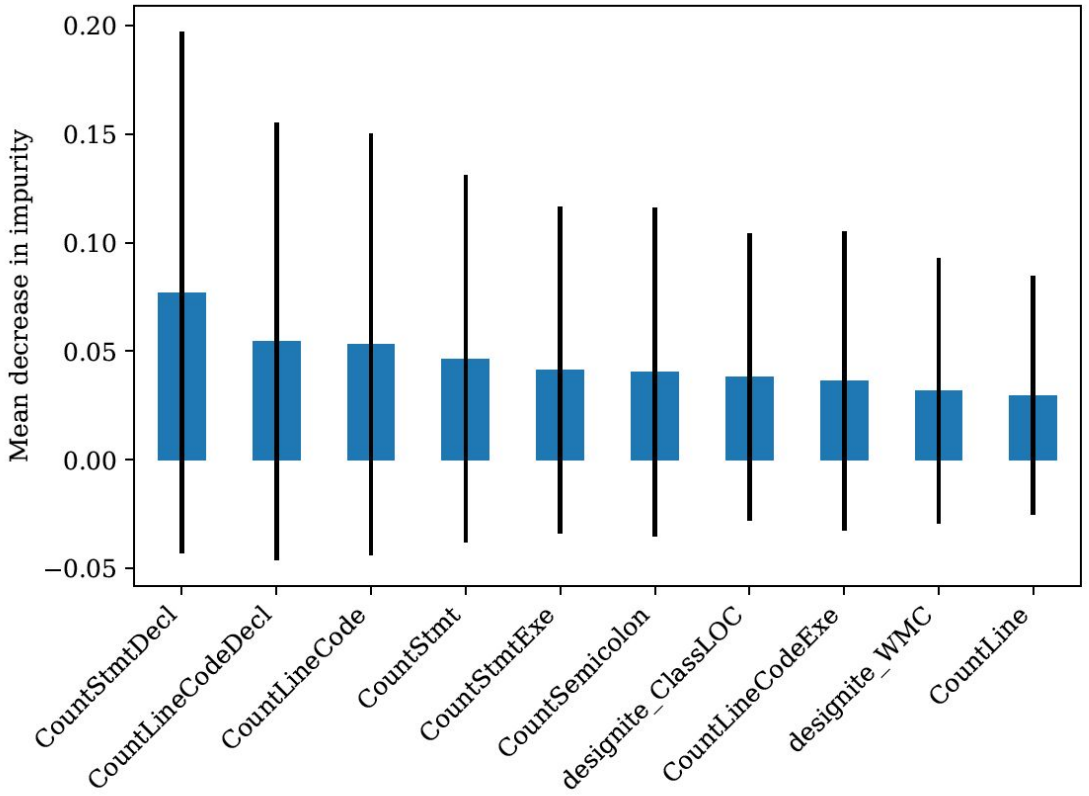
Code smells	TP	TN	FP	FN	Precision	Recall	F1-Score
Blob	111	1822	207	186	0.34906	0.37374	0.36098
Data class	80	1987	46	217	0.63492	0.26936	0.37825
Long method	80	1830	326	166	0.19704	0.32520	0.24540

The assessment of the human expert was labelled as code smell if the average review was higher than 0.75.

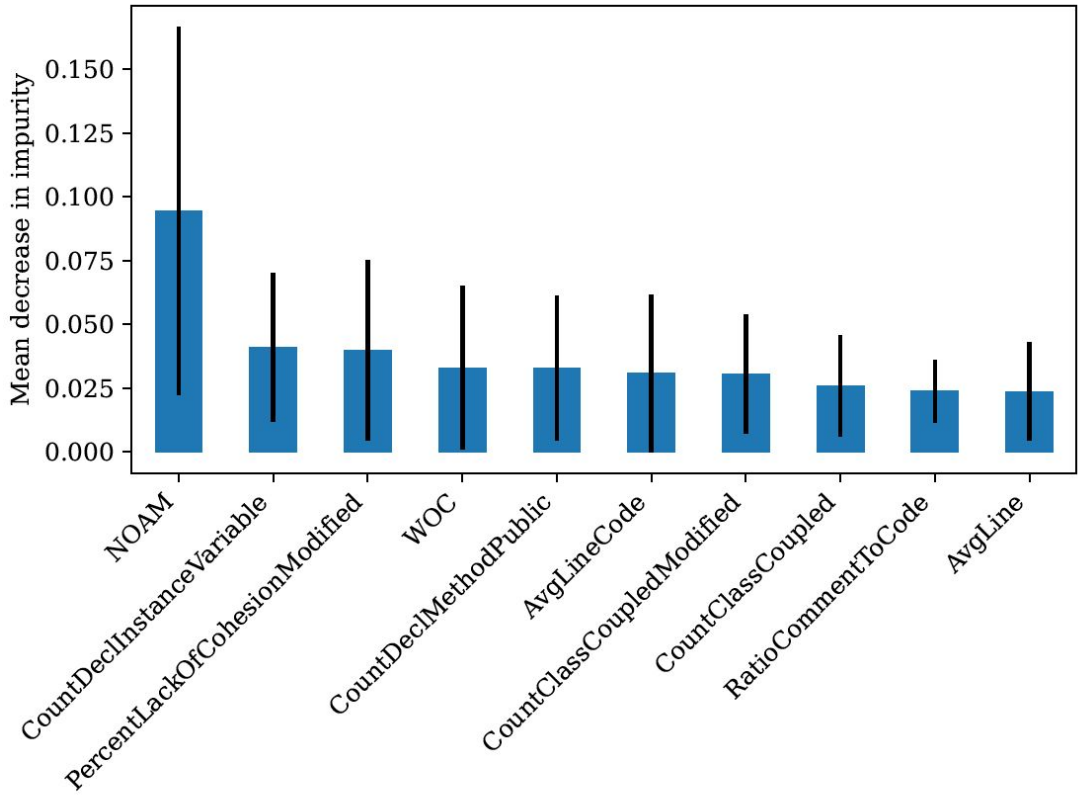
**There is a large discrepancy between what the human perceives as a code-smell, and what static analysis tools like PMD do.

Note: PMD use 100 LOC as the default threshold value to identify long method. According to MLCQ dataset, the average length of code samples identified as long method is 20.7 lines.

High Impact Features

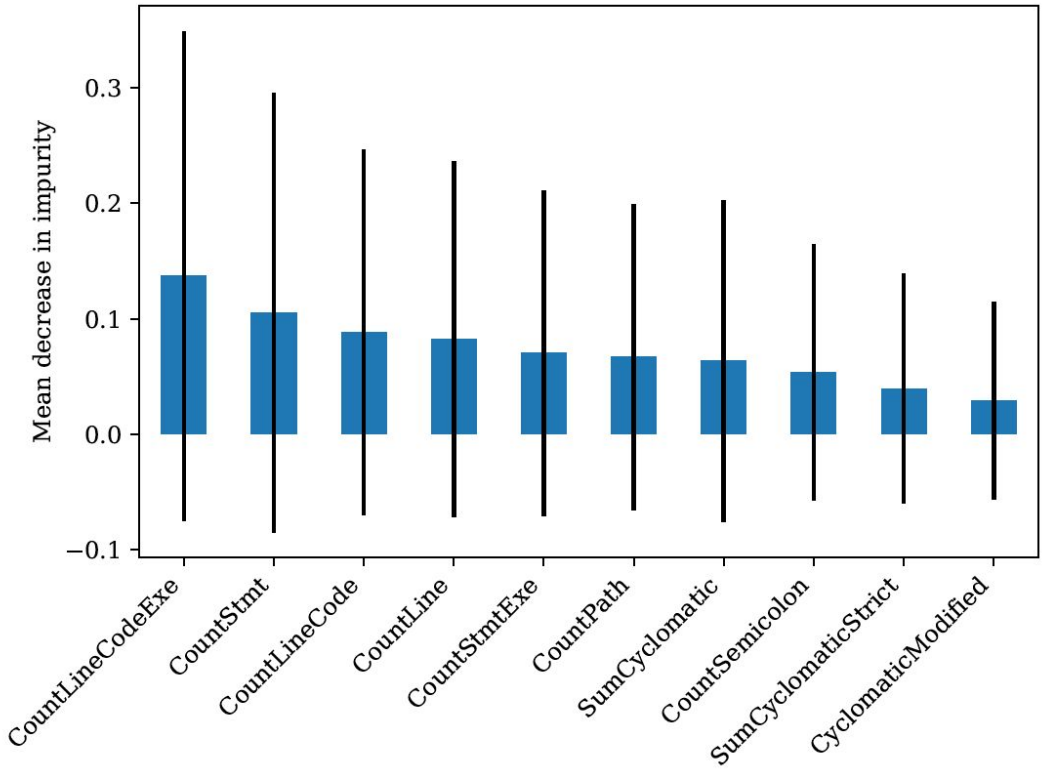


(a) Blob

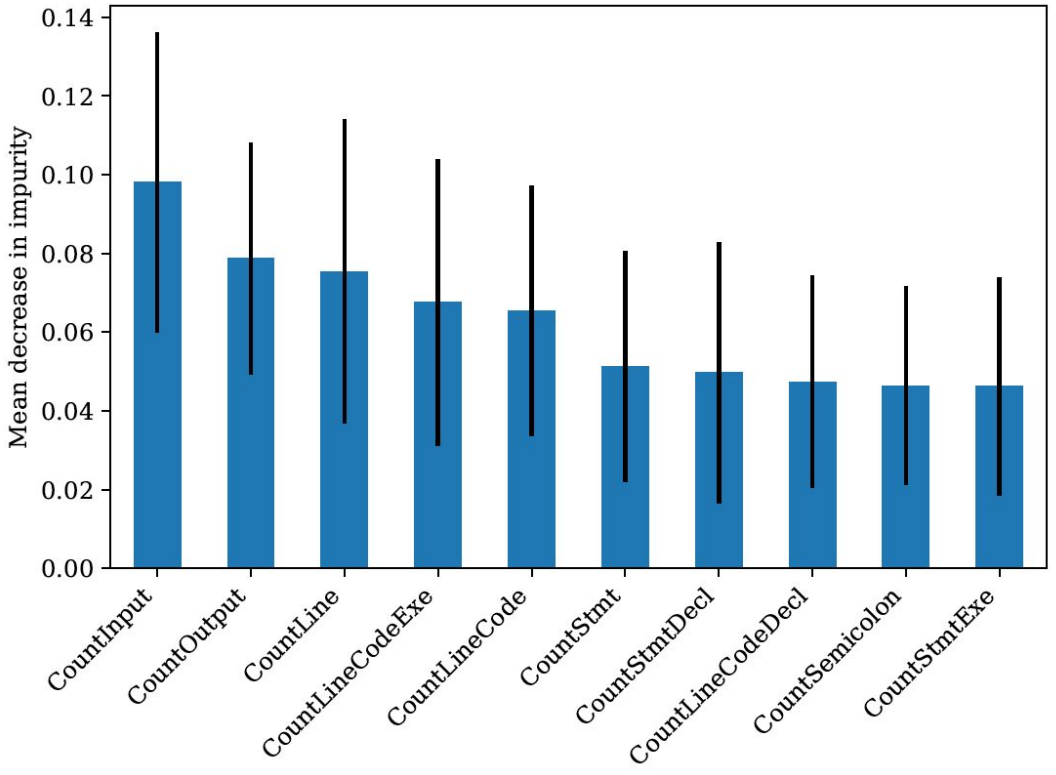


(b) Data Class

High Impact Features



(c) Long Method



(d) Feature Envy

DS&AI

Conclusions

Can we use machine learning to mimic developers' contemporary perception of code smells?

How does machine learning perform when compared to the existing heuristic-based code smell detection?

What are the features contribute most to the classification of code smell instances?