A Survey on Machine Learning: Code Smells and Antipatterns

Rodger Byrd
rbyrd2@uccs.edu

I. Abstract

Recently, machine learning methods have been applied to detect and correct code smells, but this is a new field and it is not mature. This paper surveys the current scientific research to review the latest information related to machine learning and code smells. An overview of code smells and antipatterns and machine learning is documented. The survey results are organized by topic and cover the current performance, detection methods, improvements to detection methods, using machine learning to correct identified smells, classifying code smells, and using machine learning to make predictions about source code. Training datasets are also covered in detail as that is a major factor in the performance and results.

II. Introduction

Overviews of current research on code smells are covered in *A Survey on software smells* [1], *A systematic literature review: Refactoring for disclosing code smells in object oriented software*[2]m and *Smells in software test code: A survey of knowledge in industry and academia*[3]. Code smell research is a growing field[4] as demonstrated by the increase in publishing in figure 1. For a survey of machine learning and code smell detection from 2000-2017, Azeem et al. presented a systematic literature review[5]. At the time, they only had 15 related papers to review. Haque et al.[6] performed a survey on the causes, impacts, and detection approaches of code smells but machine learning was not addressed in detail. This paper focuses exclusively on research related to code smells and machine learning. This area of study, which is the intersection of machine learning and code smells is a rapidly growing field. There are almost no relevant papers before 2016.

There are five types of automated code smell detection[7], they are:

1) Metrics based smell detection
2) Rules/heuristic-based smell detection
3) History based smell detection
4) Optimization based smell detection
5) Machine Learning based smell detection

There are also many different tools[8] that can be used for automated smell detection. Deficiencies in automated tools include false-positives and lack of context, limited detection support for known smells, and inconsistent smell detection definitions and detection methods[9]. This paper will focus on the fifth type of smell detection which is machine learning specifically from 2017-2019.

A. *Code Smells and Antipatterns*

What's interesting about anti-patterns is that they have a very human aspect to them. They overlap the way humans think with the way code is written. They connect common ways of human misunderstanding to software development. Most developers expect code to work as they understand it to and in context to their experience, they may not realize they are writing software in a way that will cause sustainment problems later on. Additionally, they don't spend a lot of time thinking about how code will work in ways they don't expect. It is in the nature of most engineers to see things in mathematical/binary ways and ignore the human aspects to what they are working on.

Code smells are similar to antipatterns, but code smells may not always be bad. Code smells can also be referred to as Atoms of Confusion [10], when referring to the smallest methods of definable smells, and nano patterns, which are very small anti-patterns, around the size of a single method. An example of confusing code is shown below in in 1. Some developers will expect the output to be 25 and some will be 13 because they don't understand intuitively how the macro function works (the correct answer is 13).

Listing 1. Example Atom of Confusion

```
#DEFINE M 2+3

int main() {
  int x=5, y;
  y= x * M;
  cout << y << endl;
  return 0;
}
```

Kent Beck coined the term "code smell" [11] and defined as "certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring". They are also sometimes referred to as architecture smells. Some common examples of code smells are as follows:

a) Large Class - Bloaters: methods and classes that have grown so large they become unsustainable, and usually accumulate over time.

b) Lazy Class: methods and classes so small they are pointless or useless.

c) Object-Oriented Abusers: misuse of object-oriented programming principles

d) Change Preventers: where multiple places in code need to be updated for a single change made somewhere else.

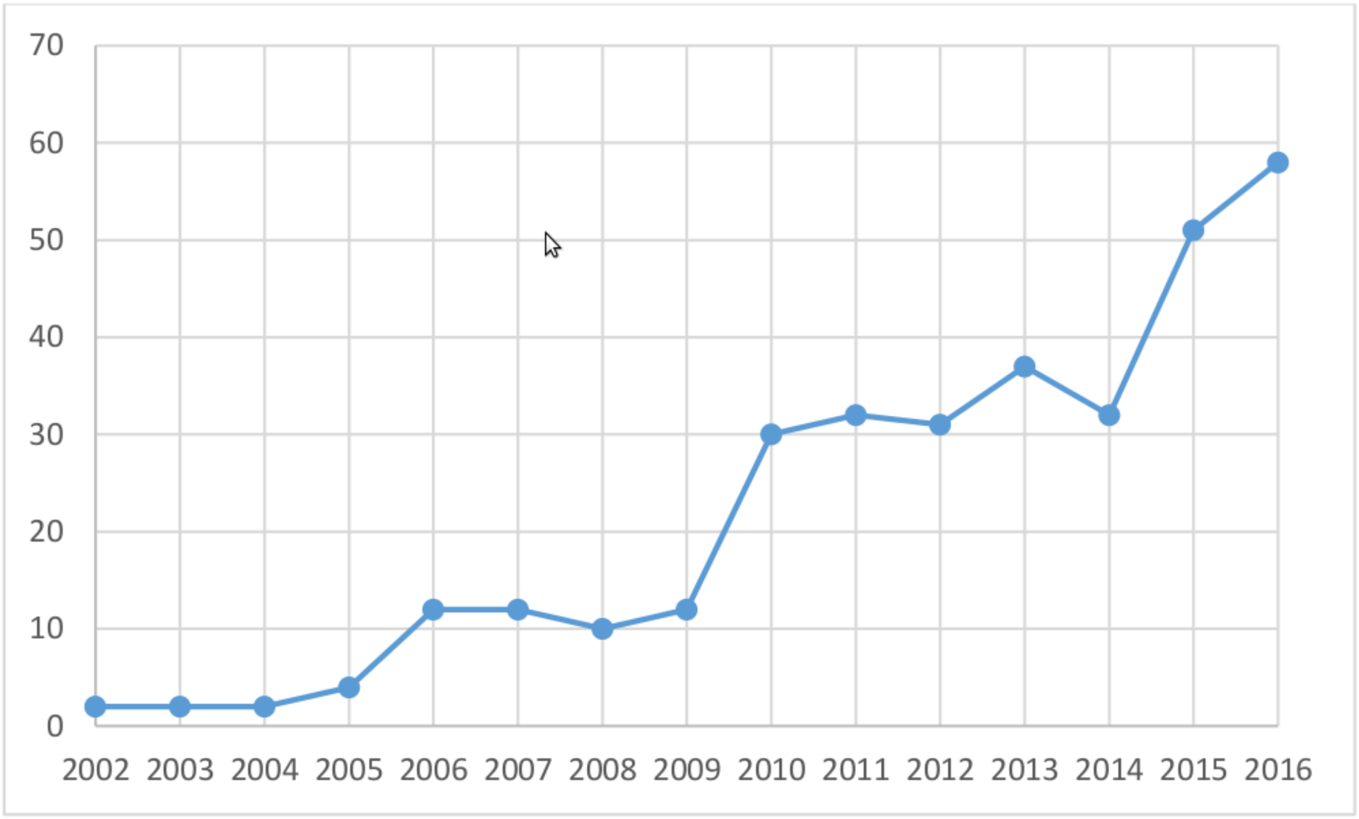

Fig. 1. Code smell research literature production from 2002 to 2016[4]

e) Dispensables: pointless or useless code.

f) Duplicate: code that is copy and pasted in multiple places in the code base.

g) Couplers: excessive coupling between classes.

The term antipattern was coined by Andrew Koenig [12]. It is defined as a commonly reinvented bad solution to a problem. Some common examples of antipatterns are as follows:

h) Singleton Overuse: Overuse of singletons as they violate information hiding

i) Functional Decomposition: functional methods are remnants of procedural languages and conflict with object-oriented practices

j) Poltergeist: short-lived, limited functionality object used to perform initialization or invoke methods in another class.

k) Spaghetti: very long methods and classes with many lines of code.

l) Blob: a class with lots of attributes and methods, which can be unrelated.

m) Copy-Paste: code copied multiple times in the code base.

n) Lava Flow: Ancient code which can't be modified for fear of introducing errors.

Some papers use the terms antipatterns and code smells interchangeably[2]. For the purposes of this paper, smells are precursors to anti-patterns. A code smell signals that code should be refactored and is an indicator of problem, whereas an antipattern is a definitive problem. The existence of code smells and antipatterns implies that there are going to be problems with software sustainment and imply there are issues with the design.

B. Machine Learning

Machine learning is defined as computers learning to solve problems without being explicitly programmed, although they are "trained"[13]. Arthur Samuel coined the term Machine Learning in 1959[14]. Machine learning is a subset of artificial intelligence. It uses algorithms and statistical models to execute a task without explicitly being programmed. Machine learning is used in a wide variety of applications such as email spam filtering, search engines, video surveillance, and image curation.

There are many types of learning algorithms, such as: unsupervised learning, supervised learning, reinforcement learning, self learning, feature learning, sparse dictionary learning, anomaly detection, and association rules.

Models used for machine learning include: Artificial Neural networks, decision trees, support vector machines, Bayesian networks and genetic algorithms.

Machine learning techniques have been used for many different aspects of software engineering[15]. These include

Design Pattern Detection, Code Smell Detection, Bug Prediction, and Recommending Systems among others.

Comment: include info on Training dataset vs test dataset, Variations in how the algorithms are trained, ie developer survey, evolutionary code changes

III. Results

Tian et al.[16] and Tahir et al.[17] performed surveys on stack overflow related to how developers discuss code smells and antipatterns. In [16] they found that developers have difficulty detecting and refactoring code smells due to the lack of available tools and difficulty quantifying the cost.

A. Topic Maps

A large topic map of antipatterns and code smells is included below in figure 2. Machine learning is emphasized with a red circle. Figure 3 focuses on the area of intersection between code smells and machine learning.

comment: Include chart based on source of papers?

B. Machine Learning Code Smell Detection Performance

There are somewhat mixed results as far as the performance of machine learning algorithms with respect to code smell detection[18]. In a comparison of the 32 different machine learning algorithms[19] it was determined that the J48 and Random Forest algorithms have the best performance detecting code smells and the support vector machines had the worst performance. They also showed the algorithms had greater than 96% accuracy and only required 100 training examples to get above 95%.

The performance of machine learning code smell detection techniques needs to be compared to heuristic methods to determine whether it has better performance. Heuristic methods use detection rules based on software metrics. (Pecorelli et al.) showed[20] that their Naive Bayesian machine learning code smell detection algorithm performed at the same level or below the DECOR heuristic method. It can't be said for certain that all machine learning code smell detection will always perform at that level as it may be due to the datasets used for training or testing and the variety of algorithms.

C. Code Smell Detection

One of the larger experiments on machine learning for code smell detection was done by Fontana et al.[19]. In this study they used 16 different machine learning algorithms on four code smells (Data Class, Large Class, Feature Envy, Long Method). It included 74 software systems and 1986 manually validated code smell samples. Many other studies use the code smell samples to perform other research and validate this research. In[19] they rank the 10 best algorithms and the top two have a large effect size meaning they had a large advantage over the remaining algorithms. The top 10 algorithms are as follows:

1) B-J48 Pruned
2) Random Forest
3) B-JRip
4) B-J48 Unpruned
5) B-Random Forest
6) J48 Pruned
7) J48 Unpruned
8) JRip
9) B-J48 Reduced Error Pruning
10) J48 Reduced Error Pruning

As shown above there are many Machine learning algorithms that can be used to detect code smells[21][22][23][18][22][24][25][26]. In[21][23] (Reshi and Singh) showed that those algorithms could be used to identify code smells implying that it could be used for preventative maintenance and to document defects or their absence in software. They performed the assessment on the open source Eclipse software so their work could be verifiable. Others have created novel algorithms[27] like a hybrid J48 algorithm with optimization. Other papers refer to code smells as "Evolvability Defects"[28] but use similar methodologies such as training machine learning algorithms to detect them. In[28], they refer to code smells as availability defects and antipatterns as functional defects.

The scale of the machine learning studies cited above varies widely. From [19] where they tested 16 different algorithms to [24] where the tested a single SVM algorithm. Machine Learning algorithms consistently show they are able to detect code smells. Additionally, they vary widely in what is considered a code smell and how many they are testing.

a) Vulnerability Detection: In a slightly different field but related effort, Machine learning was used to look for software vulnerabilities[29]. In this research, they used machine learning with code metrics "trivial features" as the training dataset. These included properties like character count, entropy, max nesting depth, if-complexity and other features. It did not result in good vulnerability detection results.

b) Mobile Applications: Code smells in mobile applications could lead to poor performance. Algorithms have been developed[30][31] to generate detection rules for Android applications. In[31], algorithms have been designed to detect and refactor code smells for mobile applications.

c) Techniques and Tools: Tian et al.[16] performed a survey on stack overflow related to how developers discuss code smells and antipatterns. They found that developers have difficulty detecting and refactoring code smells due to the lack of available tools and difficulty quantifying the cost. In[2] they test tools for performance in recognizing design smells using iPlasma with the J48 Decision Tree algorithm against open source software.

WekaNose is a code smell detection machine learning tool[32]. It has many of the same challenges of any machine learning algorithm that is used. It is dependent on code smell input definitions and characterization feedback on the likelihood of smell instances.

d) Deep Learning - Neural Networks: The biggest challenge for deep learning based code smell detection[33] is that it requires larger training datasets than typical machine learning algorithms. In [33] Liu et al. showed that an automated

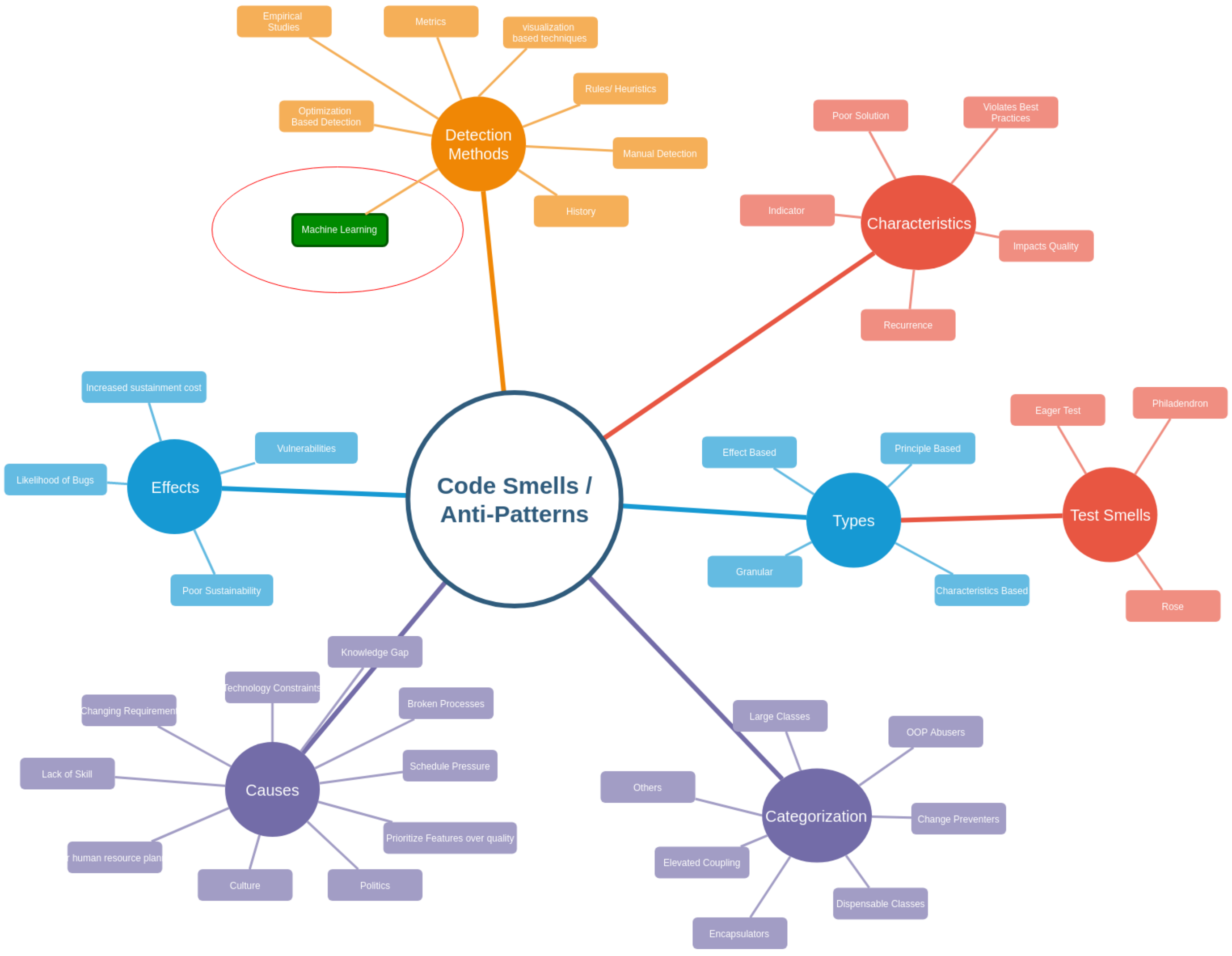

Fig. 2. High level topic map on code smells and antipatterns

approach for building training datasets was possible and that Deep Learning could be used to detect code smells. In a related area, Ban et al. showed that a deep learning model could be used to identify potential vulnerabilities[34], but had poor performance when facing cross-project vulnerability detection. Liu et al. focus solely on the "feature envy" code smell and used deep learning to identify it in code. They also proposed automation to create their training dataset.

Other research[35],where they refer to code smells as "linguistic antipatterns", has shown that deep learning did not outperform typical machine learning algorithms. In[35], they showed that correctly tuned, traditional machine learning classifiers can outperform deep learning for smell detection. Hardware constraints such as memory requirements resulted in much lower performance for the deep learning model against all metrics that were measured.

e) Novel Approaches to Training Datasets: Machine learning algorithms need to have a training dataset. One novel approach for that is to use the same codebase over time and use the deltas in code as an evolutionary training dataset[36]. The assumptions for this are that the refactoring that happens over time will correct code smells and the algorithm can train on that dataset. In[36] Wang et al. showed that they could identify code smells in open source datasets based on this model. This evolutionary approach is contrary to the idea of software entropy[37], which proposes that code will grow more disorderly over time, not less disorderly. The evolutionary approach may not produce consistent results for all code bases.

f) Detection on Web Services: It is possible to predict the occurrence of web-service antipatterns using source code metrics and machine learning[38]. In their test Kumar and Sureka demonstrate that Random Forest was the highest performing algorithm. While a lot of time has been spent on defects in object-oriented software development, less has been spent on web service design defects[39]. In[39] Ouni et al. use two different machine learning techniques, Support Vector Machine and Simulated Annealing, on eight different defect

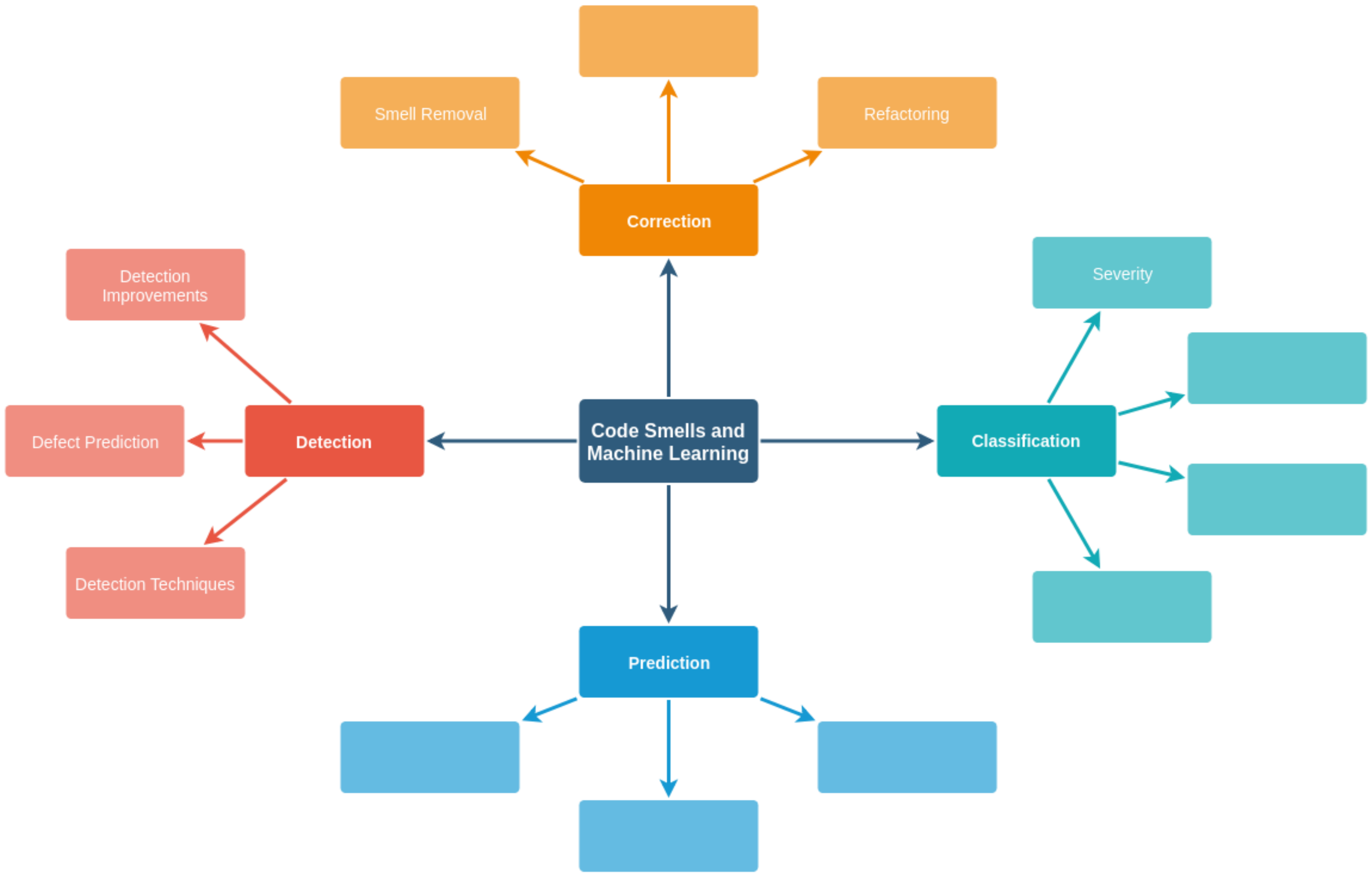

Fig. 3. Detail of topic map on machine learning related to code smells and antipatterns

types and achieved an average of 91% precision.

D. *Code Smell Detection Improvements*

Sharma et al.[40] look at the feasibility of using Transfer-learning to improve the performance of machine learning algorithms when detecting code smells. This opens the opportunity to use transfer learning to do code smell detection on programming languages where smell detection tools are not yet available.

Using machine learning code smell detection has shown that code smells are a strong predictor of class change proneness[41] with greater than 70% probability. Software change proneness is related to its quality and sustainability.

Foidl and Felderer have proposed[42] a risk-based approach to weight the input data quality of the machine learning algorithm with the impact of the "badness" of the identified code smell. This method could help to optimize and prioritize software maintenance efforts for large scale sustainment projects. Barbez et al. propose[43] an ensemble method based deep learning algorithm for anti pattern detection which had a precision of 48% vs the heuristic DECOR algorithm which is 35%. Ensemble learning combines the predictions from multiple neural networks.

E. *Code Smell Correction and Refactoring*

It has been shown that machine learning can be used to automatically fix or propose patches for software[44]. For if-statement repair Xiong et al. showed that they could use machine learning to predict code repair with 43.5% precision.

F. *Code Smell Classification Severity*

Most code smells are treated with equal severity in the works cited in this paper. Fontana and Zanoni[15] create a code smell severity classification using machine learning. They classify four code smells, Data Class, God Class, Feature Envy, and Long Method on a scale of 0-3 ranging from no smell to severe smell. In this experiment, they followed many of the same methods as other studies, but included a severity aspect.

G. *Code Smell Predictions*

Machine learning has been used to detect code smells and link those findings to bug predictions[45] using Naive Bayesian, Random Forest and Logistic Regression algorithms. There are also related studies on defect prediction and software reliability prediction. For a survey on the current state of software defect prediction[46]. Li et al. have done a thorough survey including publicly available datasets and machine learning. Additionally, software reliability can be predicted using machine learning techniques[47]. A software defect prediction model using machine learning and code smells was proposed by Soltanifar et al.[48]. In[48] to decrease maintenance time and improve developer efficiency, they used machine learning to obtain actionable information from code smells. Using Naive Bayes and logistic regression algorithms they showed that code smells outperformed code metrics.

H. Training Datasets

comment: refer to detection datasets paragraph and consolidate

In[46], they document many publicly available datasets for software defect prediction. This is important to compare the results of research. A common public dataset used in many of the papers is from [15]. It is used in [22] among others. Not all training material is equal[35], and for some problems like bug prediction there are abundant datasets, but other problems may require significant effort to build relevant labeled datasets.

In Kaur and Kaur[49], they try to address the problem of poor performance of machine learning due to imbalanced datasets. In their experiments they evaluated 23 machine learning algorithms and showed that code smells and metrics together outperformed testing against software metrics alone which agrees with the results from[29].

IV. Discussion

Most confusion is in training datasets and identifying code smells because they can be subjectively interpreted. Additionally it isn't enough to just use expert advice/knowledge unless it has been experimentally shown to be correct code smells. There must be a way to mathematically demonstrate or by experimentation that particular code patterns cause problems. The big challenge in this field seems to be the best way to identify code smells and how training datasets are labeled and created.

Beyond just sustainment and refactoring code, code smells and antipatterns need to be understood and identified. Once code smell taxonomies are standardized and identified developers can change their best practices to prevent the introduction of these problems. The best practices should be created in such a way that the typical misunderstandings of the developer can be avoided.

All the aforementioned software patterns can cause confusion to the reader of the code. Developers spend more time reading than writing (citation?) code. If these smells and antipatterns can be detected and corrected, it will lead to more sustainable, more stable code with less time spent by the developer so less cost as well.

V. Conclusion

Most developers think that a compiler is well-defined and bugs in code must be due to lack of understanding not code structure that is good at confusing human nature. Manual analysis to detect code smells would take so much time it is completely impractical from a cost perspective. The problem is developers don't realize they are implementing anti-patterns at the time they are writing code. The interesting things about it are how do we find them, detect them, and what is the fix when we identify the problem.

Because we know that the anti-patterns cause confusion to the developer, they create risk because they create unstable code. This is risk for the owner of the software and the customer of the developer who uses the code. For real world examples of the problem, in Temp Az, a person hit by self driving car[50]. Another example is the Boeing 737 MAX. Code developers expected pilots to respond with typical emergency procedure, but when the errors occurred the pilots were overwhelmed by the amount of feedback being given to them by the systems in the cockpit[51] causing the planes to crash.

Machine learning algorithms have been clearly shown to be able to detect code smells, but there is no current standard approach. Performance varies due to different algorithms, and different training datasets. J48 and Random Forest algorithms come up often as having the best performance detecting code smells. Beyond the specific algorithms, the training approaches and datasets make a large difference in the performance of the algorithms. There are typical approaches to creating training data such as surveying developers and more interesting approaches such as evolutionary approach which look at the deltas in refactored code. This points to the fact that exactly how to identify code smells and consensus on what they are specifically in code are not well defined. Multiple studies showed similar performance to the other non-machine learning models.

The papers reference above varied widely in the number of code smells they searched for, which algorithms they used, and how they came up with training datasets. Software sustainment and assurance is one of the most expensive parts of the software life-cycle[46] and this subject can have a direct impact on lowering that effort and cost.

The files for this latex document are in the github repository located at `https://github.com/rodger79/CS6000`

References

[1] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158 – 173, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303114

[2] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software," *Ain Shams Engineering Journal*, vol. 9, no. 4, pp. 2129 – 2151, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2090447917300412

[3] V. Garousi and B. Kk, "Smells in software test code: A survey of knowledge in industry and academia," *Journal of Systems and Software*, vol. 138, pp. 52 – 81, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303060

[4] P. Kokol, M. Zorman, G. Zlahtic, and B. Zlahtic, *Code smells*, 2018.

[5] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115 – 138, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584918302623

[6] M. S. Haque, J. Carver, and T. Atkison, "Causes, Impacts, and Detection Approaches of Code Smell: A Survey," in *Proceedings of the ACMSE 2018 Conference*, ser. ACMSE '18. New York, NY, USA: ACM, 2018, pp. 25:1–25:8, event-place: Richmond, Kentucky. [Online]. Available: http://doi.acm.org/10.1145/3190645.3190697

[7] M. Lafi, J. W. Botros, H. Kafaween, A. B. Al-Dasoqi, and A. Al-Tamimi, "Code Smells Analysis Mechanisms, Detection Issues, and Effect on Software Maintainability," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Apr. 2019, pp. 663–666.

[8] B. Walter, F. A. Fontana, and V. Ferme, "Code smells and their collocations: A large-scale experiment on open-source systems," *Journal of Systems and Software*, vol. 144, pp. 1 – 21, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301109

[9] T. Sharma, "Detecting and Managing Code Smells: Research and Practice," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 546–547, event-place: Gothenburg, Sweden. [Online]. Available: http://doi.acm.org/10.1145/3183440.3183460

[10] D. Gopstein, J. Iannacone, Y. Yan, L. DeLong, Y. Zhuang, M. K.-C. Yeh, and J. Cappos, "Understanding Misunderstandings in Source Code," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 129–139, event-place: Paderborn, Germany. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106264

[11] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[12] A. Koenig, "Patterns and antipatterns," *The patterns handbook: techniques, strategies, and applications*, vol. 13, p. 383, 1998.

[13] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[14] A. L. Samuel, "Some studies in machine learning using the game of checkers. IIrecent progress," in *Computer Games I*. Springer, 1988, pp. 366–400.

[15] F. A. Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowledge-Based Systems*, vol. 128, pp. 43 – 58, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705117301880

[16] F. Tian, P. Liang, and M. A. Babar, "How Developers Discuss Architecture Smells? An Exploratory Study on Stack Overflow," in *2019 IEEE International Conference on Software Architecture (ICSA)*, Mar. 2019, pp. 91–100.

[17] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, "Can You Tell Me if It Smells?: A Study on How Developers Discuss Code Smells and Anti-patterns in Stack Overflow," in *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018*, ser. EASE'18. New York, NY, USA: ACM, 2018, pp. 68–78, event-place: Christchurch, New Zealand. [Online]. Available: http://doi.acm.org/10.1145/3210459.3210466

[18] D. D. Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. D. Lucia, "Detecting code smells using machine learning techniques: Are we there yet?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2018, pp. 612–621.

[19] F. Arcelli Fontana, M. V. Mntyl, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1143–1191, Jun. 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9378-4

[20] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, "Comparing Heuristic and Machine Learning Approaches for Metric-based Code Smell Detection," in *Proceedings of the 27th International Conference on Program Comprehension*, ser. ICPC '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 93–104, event-place: Montreal, Quebec, Canada. [Online]. Available: https://doi.org/10.1109/ICPC.2019.00023

[21] J. A. Reshi and S. Singh, "Investigating the Role of Code Smells in Preventive Maintenance," *Journal of Information Technology Management*, vol. 10, no. 4, pp. 41–63, 2019.

[22] K. Karauzovi-Hadiabdi and R. Spahi, "Comparison of Machine Learning Methods for Code Smell Detection Using Reduced Features," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, Sep. 2018, pp. 670–672.

[23] J. A. Reshi and S. Singh, *Predicting Software Defects through SVM: An Empirical Approach*, 2018.

[24] A. Kaur, S. Jain, and S. Goel, "A Support Vector Machine Based Approach for Code Smell Detection," in *2017 International Conference on Machine Learning and Data Science (MLDS)*, Dec. 2017, pp. 9–14.

[25] M. Hozano, N. Antunes, B. Fonseca, and E. Costa, "Evaluating the Accuracy of Machine Learning Algorithms on Detecting Code Smells for Different Developers." in *ICEIS (2)*, 2017, pp. 474–482.

[26] D. Le and N. Medvidovic, "Architectural-based Speculative Analysis to Predict Bugs in a Software System," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 807–810, event-place: Austin, Texas. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889260

[27] A. Kaur, S. Jain, and S. Goel, "SP-J48: a novel optimization and machine-learning-based approach for solving complex problems: special application in software engineering for detecting code smells," *Neural Computing and Applications*, Apr. 2019. [Online]. Available: https://doi.org/10.1007/s00521-019-04175-z

[28] N. Tsuda, H. Washizaki, Y. Fukazawa, Y. Yasuda, and S. Sugimura, "Machine Learning to Evaluate Evolvability Defects: Code Metrics Thresholds for a Given Context," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Jul. 2018, pp. 83–94.

[29] B. Chernis and R. Verma, "Machine Learning Methods for Software Vulnerability Detection," in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, ser. IWSPA '18. New York, NY, USA: ACM, 2018, pp. 31–39, event-place: Tempe, AZ, USA. [Online]. Available: http://doi.acm.org/10.1145/3180445.3180453

[30] J. Rubin, A. N. Henniche, N. Moha, M. Bouguessa, and N. Bousbia, "Sniffing Android Code Smells: An Association Rules Mining-Based Approach," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2019, pp. 123–127.

[31] R. Ibrahim, M. Ahmed, R. Nayak, and S. Jamel, "Reducing redundancy of test cases generation using code smell detection and refactoring," *Journal of King Saud University - Computer and Information Sciences*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157818300296

[32] U. Azadi, F. A. Fontana, and M. Zanoni, "Poster: machine learning based code smell detection through WekaNose," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2018, pp. 288–289.

[33] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang, "Deep Learning Based Code Smell Detection," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[34] X. Ban, S. Liu, C. Chen, and C. Chua, "A performance evaluation of deep-learnt features for software vulnerability detection," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 19, p. e5103, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5103

[35] S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, and G. Antoniol, "Keep it simple: Is deep learning good for linguistic smell detection?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2018, pp. 602–611.

[36] Y. Wang, S. Hu, L. Yin, and X. Zhou, "Using Code Evolution Information to Improve the Quality of Labels in Code Smell Datasets," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 01, Jul. 2018, pp. 48–53.

[37] A. Gupta, B. Suri, V. Kumar, S. Misra, T. Blaauskas, and R. Damaeviius, "Software code smell prediction model using Shannon, Rnyi and Tsallis entropies," *Entropy*, vol. 20, no. 5, p. 372, 2018.

[38] L. Kumar and A. Sureka, "An Empirical Analysis on Web Service Anti-pattern Detection Using a Machine Learning Framework," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 01, Jul. 2018, pp. 2–11.

[39] A. Ouni, M. Daagi, M. Kessentini, S. Bouktif, and M. M. Gammoudi, "A Machine Learning-Based Approach to Detect Web Service Design Defects," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 532–539.

[40] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, *On the Feasibility of Transfer-learning Code Smells using Deep Learning*, 2019.

[41] N. Pritam, M. Khari, L. H. Son, R. Kumar, S. Jha, I. Priyadarshini, M. Abdel-Basset, and H. V. Long, "Assessment of Code Smell for Predicting Class Change Proneness Using Machine Learning," *IEEE Access*, vol. 7, pp. 37 414–37 425, 2019.

[42] H. Foidl and M. Felderer, "Risk-based Data Validation in Machine Learning-based Software Systems," in *Proceedings of the 3rd ACM*

SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, ser. MaLTeSQuE 2019. New York, NY, USA: ACM, 2019, pp. 13–18, event-place: Tallinn, Estonia. [Online]. Available: http://doi.acm.org/10.1145/3340482.3342743

[43] A. Barbez, F. Khomh, and Y.-G. Guhneuc, "A Machine-learning Based Ensemble Method For Anti-patterns Detection," *CoRR*, vol. abs/1903.01899, 2019. [Online]. Available: http://arxiv.org/abs/1903.01899

[44] Y. Xiong, B. Wang, G. Fu, and L. Zang, "Learning to Synthesize," in *Proceedings of the 4th International Workshop on Genetic Improvement Workshop*, ser. GI '18. New York, NY, USA: ACM, 2018, pp. 37–44, event-place: Gothenburg, Sweden. [Online]. Available: http://doi.acm.org/10.1145/3194810.3194816

[45] G. M. Ubayawardana and D. D. Karunaratna, "Bug Prediction Model using Code Smells," in *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*, Sep. 2018, pp. 70–77.

[46] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175(14), Jun. 2018. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2017.0148

[47] V. K. Kulamala, A. S. C. Teja, A. Maru, Y. Singla, and D. P. Mohapatra, "Predicting Software Reliability using Computational Intelligence Techniques: A Review," in *2018 International Conference on Information Technology (ICIT)*, Dec. 2018, pp. 114–119.

[48] B. Soltanifar, S. Akbarinasaji, B. Caglayan, A. B. Bener, A. Filiz, and B. M. Kramer, "Software Analytics in Practice: A Defect Prediction Model Using Code Smells," in *Proceedings of the 20th International Database Engineering &#38; Applications Symposium*, ser. IDEAS '16. New York, NY, USA: ACM, 2016, pp. 148–155, event-place: Montreal, QC, Canada. [Online]. Available: http://doi.acm.org/10.1145/2938503.2938553

[49] K. Kaur and P. Kaur, "Evaluation of sampling techniques in software fault prediction using metrics and code smells," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2017, pp. 1377–1387.

[50] "How a Self-Driving Uber Killed a Pedestrian in Arizona - The New York Times." [Online]. Available: https://www.nytimes.com/interactive/2018/03/20/us/self-driving-uber-pedestrian-killed.html

[51] "Boeing Underestimated Cockpit Chaos on 737 Max, N.T.S.B. Says - The New York Times." [Online]. Available: https://www.nytimes.com/2019/09/26/business/boeing-737-max-ntsb-mcas.html