Makine Öğrenmesi Yöntemlerini Kullanarak Özellik Kıskançlığının Otomatik Saptanması

Zeynep ÖZKALKAN[1], Kübra AYDİN[2], Hacı Yakup TETİK[3], and Rahime Belen SAĞLAM[4]

[1] Ankara Yıldırım Beyazıt Üniversitesi,Ankara, Türkiye
zeynep.ozkalkan@tiga.com.tr
[2] Ankara Yıldırım Beyazıt Üniversitesi,Ankara, Türkiye kubra.aydin@tiga.com.tr
[3] Ankara Yıldırım Beyazıt Üniversitesi,Ankara, Türkiye haci.tetik@tiga.com.tr
[4] Ankara Yıldırım Beyazıt Üniversitesi,Ankara, Türkiye rbsaglam@ybu.edu.tr

Özet Son yıllarda yazılım sistemlerindeki kusurlu kod ile ilgili çalışmalar kod kalitesinin iyileştirilmesi amacıyla araştırmacıların önemli ölçüde dikkatini çekmiştir. Kusurlu kod, kötü tasarım ve hata eğilimini artıran ve kodun anlaşılmasını zorlaştıran uygulama belirtileridir. Bu çalışmada yaygın olarak ortaya çıkan kusurlu kodlar arasında, özellik kıskançlığını (feature envy) seçtik ve bu kusuru tanımlamak için bir çatı geliştirdik. Özellik kıskançlığı, bir metodun kendi bulunduğu sınıftan çok başka sınıfla ilgilenmesi durumu olarak tanımlanabilir. Aynı sınıfta tanımlanan metodların yüksek uyumlu olması ve aynı sınıfta bulunan üyelerle aktif bir şekilde etkileşim halinde bulunması beklenirken diğer sınıflarla daha az etkileşime sahip olmasının beklenmesi fikrine dayanır. Diğer sınıflarla etkileşimi çok büyük olan metodlar düzgün olmayan uyumluluk ve bağımlılığa neden olurlar. Çalışmamızda problemi bir sınıflandırma görevi olarak modelledik ve kod satır sayısı , dahili ve harici metodlara yapılan çağrıların sayısı gibi yapısal özelliklere farklı makine öğrenimine dayalı sınıflandırma algoritmaları uyguladık. Elde edilen sonuçlara dayanarak makine öğrenmesi tekniklerinin özellik kıskançlığının tahmin etme yeteğine sahip olduğunu ve bu amaçla yazılım geliştiricileri ve araştırmacılar tarafından kullanılabilceğini gözlemledik.

Keywords: Makine öğrenmesi · Kusurlu kod · Özellik Kıskançlığı · Sınıflandırma.

Automatic Detection of Feature Envy using Machine Learning Techniques

Zeynep ÖZKALKAN[1], Kübra AYDİN[2], Hacı Yakup TETİK[3], and Rahime Belen SAĞLAM[4]

[1] Ankara Yıldırım Beyazıt University,Ankara, Turkey `zeynep.ozkalkan@tiga.com.tr`
[2] Ankara Yıldırım Beyazıt University,Ankara, Turkey `kubra.aydin@tiga.com.tr`
[3] Ankara Yıldırım Beyazıt University,Ankara, Turkey `haci.tetik@tiga.com.tr`
[4] Ankara Yıldırım Beyazıt University,Ankara, Turkey `rbsaglam@ybu.edu.tr`

Abstract. In recent years, the studies related to code smells in software systems have received significant attentions from the researchers with the aim of improving the code quality. Code smells are symptoms of poor design and implementation choices that increase fault-proneness and decrease code comprehension. Among the commonly occurring code smells, we have picked feature envy and developed a framework to identify it. Feature envy can be described as a method that is more interested in another class than it is actually in. It is based on the idea that methods defined in the same class must be highly cohesive and expected to actively interact with other members in the same class while having less interactions with other classes. Methods that highly interact with other classes cause improper cohesion and coupling. We have modeled the problem as a classification task and employed different machine learning based classification algorithms on structural features like LOCs, number of calls made to the method internally and externally etc. Based on the results obtained, we observed that the machine learning techniques have the ability for predicting feature envy and can be used by software practitioners and researchers for this purpose.

Keywords: Machine Learning · Code Smell · Feature Envy · Classification.

1 Introduction

Today, the need for delivering high quality and maintainable software has increased due to the growing complexity and dependency of the software systems. However, it is a non-trivial task to keep quality of software systems stable. Structural degradation that appear in the form of modules, components and interfaces not acting in accordance to the design-time architecture is inevitable. There are several undesirable characteristics, which are also called code smells, that may appear over time within a software project. Code smells have been defined by Fowler [9] as symptoms of poor design and implementation choices that may originate from activities performed by developers while in a hurry, or by simply making poor design choices. Most common code smells can be given as Blob, Spaghetti Code, Functional Decomposition, Data Class, Shotgun Surgery and Feature Envy. In this study, we focused on Feature Envy (also known as Intensive Coupling or Disperse Coupling) which appears when a method accesses the data of another object more than its own data or calls more methods from other classes than from itself. The method might be heavily using attributes from one or more external classes, directly or via accessor operations. Since such a method is tightly coupled to other classes, it seems to be misplaced in the current one and should be identified and corrected by the development team as early as possible for maintainability and evolution considerations. This smell can be removed via Move Method refactoring operations which is achieved by creating a new method in the class that uses the method the most, then moving code from the old method to there. However, it is hard to identify such a code smell manually. Although human involvement reduces uncertainties in detection process, manual techniques are infeasible for large software systems. In addition to this, it has been argued in the literature that code smells could be subjectively interpreted by humans and even by the automated tools [7]. Even though well-known object-oriented metrics are computed, detection rules may differ or different thresholds can be used for the same metrics. These thresholds can change the number of code smells found and can decrease accuracy by detecting false positive smells. Such rules based approaches do not consider the information related to the domain, the size and the design of the system analyzed. For all these reasons, machine learning offers better targeted solutions subjective to the particular user or community by supporting a learn-by-example process [8].

In this study, we have modeled the problem as a classification task and employed different machine learning based classification algorithms on structural features extracted from the source code. It has been observed that the machine learning based classifiers have the potential to be used for this purpose.

2 Related Work

Code smells have been identified by Fowler et. al in 1999 as low-level design flaws related to the implementation of functionalities within the components [9]. They identified 22 code smells and explained how to fix them. Since then several researchers developed code smell detection techniques that perform static or dynamic analysis to achieve this goal. Static analysis include manual approaches, metrics based, symptom based, probabilistic based, visualization based, cooperative based and search based approaches. Recent studies do not rely on manual techniques since they are error prone and time consuming.

Metric-based approaches are based on defining symptoms that characterize smells and proposing set of metrics to measure these symptoms. Having this information, thresholds are defined to classify the class as affected or not by the defined symptoms. One of those studies was proposed by Marinescu with the aim of finding deviation from good design principles and heuristics [13]. The researchers identified symptoms and proposed some metrics characterizing the smells.

Another metric-based study was conducted by Munro for detection of two smells namely Lazy Class and Temporary Field [14]. A set of thresholds is applied to the measurement of some structural metrics to identify those smells. In the study three metrics have been used: number of methods, line of code, weight methods per class, and coupling between objects.

Kwankamol proposed a metric-based technique to detect Feature Envy based on the idea that methods defined in the same class must be highly cohesive and actively interact with other members in the same class (internal interactions) while having less interactions with other classes (external interactions) [15]. The researcher checked whether external interactions were stronger than internal interactions by a scoring feature envy factor based on number of internal and external method calls and reported promising results. Even though metric-based studies dominate the literature for code smell detection, they have the difficulty to define threshold values for metrics manually. In order to address the limitations of metric-based studies M. Kessentini et al. used heuristic search algorithms such as Harmony Search, Particle Swarm Optimization, and Simulated Annealing to define metrics and thresholds automatically. They leveraged the knowledge from previously manually inspected projects in order to detect design defects to generate new detection rules where the detection rules are automatically derived by an optimization process[11].

Instead of structural features that are characterized by source code metrics, Palomba et. al. made use of features characterized by how source code changes over time [17]. In this context, they defined Feature Envy as a method of a class that tends to change more frequently with methods of other classes rather than with those of the same class. They proposed an approach to detect smells based on change history information mined from versioning systems.

Kumar and Chhabra proposed a dynamic approach for feature envy detection by considering the actual execution performance instead of the static behavior [12]. It has been concluded that use of dynamic analysis was advantageous in detection of feature envy.

Machine learning based approaches for code smell detection are highly limited. Kaur et al. used support vector machine algorithm for detecting different four types of code smells which are God Class, Feature Envy, Data Class and Long Method [10]. They used polynomial kernel because it works with non - linearly separable data. They reported high precision and recall values. In our study, in addition to SVM, we have also applied naive bayes and decision tree for feature envy detection. Some other machine learning based classifiers used for detecting code smells are random forest, J48, support vector machine etc [8]. The researchers worked on four common code smells which are Data Class, Large Class, Feature Envy and Long Method. They analyzed 74 software systems, with several manually validated code smell samples. It has been reported that the highest performances were obtained by J48 and Random Forest, while the worst performance was achieved by support vector machines.

3 Dataset

We have conducted our experiments on two open source projects that are given in Table II. Fiware is framework of open source platform components which can be assembled together and with other third-party platform components to accelerate the development of Smart Solutions in multiple industries[2]. We have covered the Authorization PDP Generic Enabler (formerly called Access Control GE) which provides an API to get authorization decisions based on authorization policies [1].

Apache Tomcat is the second project covered in this study .It is an open source implementation of some specifications of Java Community Process and has been widely used as a dataset for several similar studies. We applied JSpirit[3] and JDeodorant[6] which are publicly available Eclipse plug-ins that have been widely used to identify code smells in the literature [17] [16]. We have validated code smells detected by the tools manually and labeled the data for the classification task accordingly.

We worked on a limited dataset in which we have 20 methods 10 of which are labeled as feature envy. We have covered 15 methods in the training set and 5 methods in the test set. In order to overcome the limitation of the data size, we tested each model using 5-fold cross validation.

Table 1. Dataset properties

System	Version	Line of Codes	Number of class
Apache Tomcat	7.0.2	283289	1538
Fiware AuthZForce	5.4.2	11078	42

Table 2. Project Analyze Tools

Project	Tool Name
Apache Tomcat	JDeodarant
AuthZForce	JDeodarant, JSpIRIT

4 Proposed Work

This paper proposes a technique to detect Feature Envy for Java projects using machine learning based classifiers. For this purpose we have implemented a framework to parse the source code and transform it into abstract syntax tree (AST) using Plyj[4], a library written in Python which is used as a Java 7 parser for parsing the source code. AST is tree version of the syntactical structure source code in any programming language which enables to calculate some structural metrics such as line of code (LOC) or number of method calls. As mentioned above, Feature envy is a problem of improper coupling between classes and low cohesion within the class. Consequently the number of external and internal method calls and the number of external and internal attribute access have been calculated to uncover relationships between methods. We have also computed complexity of the method with the consideration that external method calls are more acceptable for the methods with high complexity whereas they can be considered as feature envy for low complex methods. We made use of a library called Lizard[5] for this purpose which is a free open source library for complexity analysis supporting many programming languages. LOC is another feature we cover in our models.

We have applied three classification algorithms which are naive bayes, decision tree and support vector machines and evaluated the results using recall, precision, and accuracy.

5 Experimental Results

Four structural features have been used to detect detect feature envy methods; code complexity, line of code (LOC), number of internal method call and the number of external method call. Each method has been represented by a vector with those four features. We also checked the number of internal and external attribute accesses, however, we did not cover it since the values of this feature were zero for all the methods in the dataset disregarding the classes they belong to.

We computed the average accuracy, precision and recall values for the experiments conducted using 5-fold cross validation. The results are displayed in Table 3.

Table 3. Experimental Results

	SVM	Decision Tree	Naive Bayes
ACCURACY	0.733	0.467	0.6
PRECISION	0.8	0.633	0.667
RECALL	0.867	0.767	0.867

As observed from Table 3. SVM outperforms the other classifiers which is in line with the findings in [10]. The results of SVM show that proposed approach has a potential to systematically and efficiently detect feature envy methods in an object oriented Java based systems.

6 Conclusion

Detection of code smells is a hard task for large software projects, however they need to be identified to improve quality of the source code and to simplify maintenance. In this paper, we proposed a machine learning technique to detect Feature Envy. Several experiments has been conducted on two open source projects to show how classifiers work for the detection of Feature Envy. The results show that SVM outperforms naive bayes and decision tree. In future, we are planning to generalize this technique on different code smells using different features and different machine learning algorithms on projects which are bigger in terms of size and number of developers. We are also planning to extend the model to suggest appropriate refactoring techniques for the detected code smells.

References

1. Authorization pdp - authzforce | fiware catalogue. https://catalogue-server.fiware.org/enablers/authorization-pdp-authzforce. (Accessed on 06/18/2018).
2. Home - fiware. https://www.fiware.org/. (Accessed on 06/18/2018).
3. Jspirit - santiago vidal. https://sites.google.com/site/santiagoavidal/projects/jspirit. (Accessed on 06/18/2018).
4. musikk/plyj: A java parser written in python using ply. https://github.com/musiKk/plyj. (Accessed on 06/18/2018).
5. terryyin/lizard: A simple code complexity analyser without caring about the c/c++ header files or java imports, supports most of the popular languages. https://github.com/terryyin/lizard. (Accessed on 06/18/2018).
6. Welcome to jdeodorant. https://users.encs.concordia.ca/ nikolaos/jdeodorant/. (Accessed on 06/18/2018).
7. Francesca Arcelli Fontana, Pietro Braione, and Marco Zanoni. Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, 11(2):5–1, 2012.
8. Francesca Arcelli Fontana, Mika V Mäntylä, Marco Zanoni, and Alessandro Marino. Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3):1143–1191, 2016.
9. Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.
10. Amandeep Kaur, Sushma Jain, and Shivani Goel. A support vector machine based approach for code smell detection. In *Machine Learning and Data Science (MLDS), 2017 International Conference on*, pages 9–14. IEEE, 2017.
11. Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Manuel Wimmer. Search-based design defects detection by example. In *International Conference on Fundamental Approaches to Software Engineering*, pages 401–415. Springer, 2011.
12. Swati Kumar and Jitender Kumar Chhabra. Two level dynamic approach for feature envy detection. In *Computer and Communication Technology (ICCCT), 2014 International Conference on*, pages 41–46. IEEE, 2014.
13. Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pages 350–359. IEEE, 2004.

14. Matthew James Munro. Product metrics for automatic identification of" bad smell" design problems in java source-code. In *Software Metrics, 2005. 11th IEEE International Symposium*, pages 15–15. IEEE, 2005.
15. Kwankamol Nongpong. Feature envy factor: A metric for automatic feature envy detection. In *Knowledge and Smart Technology (KST), 2015 7th International Conference on*, pages 7–12. IEEE, 2015.
16. Ali Ouni, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Kalyanmoy Deb. Multi-criteria code refactoring using search-based software engineering: An industrial case study. *ACM Trans. Softw. Eng. Methodol.*, 25(3):23:1–23:53, June 2016.
17. Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, 41(5):462–489, 2015.