Identificação dos Cenários que Influenciam a Acurácia do Algoritmo de *Naive Bayes* em Classificar o *Code Smell Large Class*

Laercio Nazareno Neto[1]

[1] Bacharelado em Engenharia de Software
Instituto de Ciências Exatas e Informática – PUC Minas
Ed. Fernanda. Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

laercio.nazareno@sga.pucminas.br

Abstract. *This article presents a study related to the identification of characteristics that affect the accuracy of the Naive Bayes algorithm in detecting code smells. The implementation of machine learning algorithms reveals a variation in the results presented due to the imbalance of the data, which affects the accuracy of the classification, the problem that this article seeks to solve is:* ***What are the scenarios that increase the accuracy of Naive Bayes algorithm in classifying Large Class?****. Based on the presented arguments, the objective of this work is to* ***identify which scenarios increase the classification accuracy of the Naive Bayes algorithm in detecting Large Class****.*

Resumo. *Este artigo apresenta um estudo relacionado a identificação das características que afetam a acurácia do algoritmo de Naive Bayes em detectar code smells. A implementação dos algoritmos de machine learning revelam uma variação nos resultados apresentados devido ao desequilíbrio dos dados, o que afeta a acurácia da classificação, o problema que este artigo busca solucionar é:* ***Quais são os cenários que aumentam a acurácia do algoritmo de Naive Bayes em classificar Large Class?*** *Com base nos argumentos apresentado o objetivo deste trabalho é* ***identificar quais cenários aumentam a acurácia de classificação do algoritmo de Naive Bayes em detectar Large Class****.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientador acadêmico (TCC I): Lesandro Ponciano - lesandrop@pucminas.br
Orientador do TCC II: Johnatan Alves OLIVEIRA

Belo Horizonte, 15 de Maio de 2022.

1. Introdução

O termo *code smell* foi utilizado por Martin Fowler (1999) para caracterizar trechos de códigos que manifestam possíveis problemas de manutenção. A degradação da qualidade do software está ligada diretamente a *code smells* [Oliveira et al. 2020]. Com a finalidade de evitá-la, é necessário detectar e solucionar as implementações inadequadas como *Code*

Duplicate, Feature Envy, Large Class, Large Function, entre outras. Existem maneiras automáticas de identificar *code smells* que são baseadas em métricas, heurísticas ou por algoritmos de *machine learning* [Mhawish and Gupta 2020], este artigo foca em *machine learning*. O uso de técnicas de *machine learning* apontam resultados promissores em identificar determinados tipos de implementações inadequadas [Oliveira et al. 2020], e, além disso, possibilita relacionar os *code smells* entre si e identificar a correlação entre eles [Fontana and Zanoni 2011a].

A implementação dos algoritmos *J48*, *Random Forest*, *Naive Bayes*, *JRip* revelam uma variação nos resultados apresentados devido ao desequilíbrio dos dados [Pecorelli et al. 2019a], o que afeta a acurácia da classificação. Pode-se observar tal comportamento ao comparar a acurácia de diferentes implementações do algoritmo de *Naive Bayes* em identificar *Large Class* [Pecorelli et al. 2019a]. Esse algoritmo supervisionado consiste no cálculo probabilístico que classifica de acordo com a multiplicação das probabilidades de ocorrência de cada evento [Katti et al. 2021]. Com base nos argumentos apresentados, o problema que este artigo busca solucionar é: **Quais são os cenários que aumentam a acurácia do algoritmo de *Naive Bayes* em classificar *Large Class*?**

Resolver este problema é relevante, porque a literatura aponta uma lacuna de conhecimento na aplicação de *machine learning* na detecção de *code smells* [Pecorelli et al. 2019b]. Outros pontos de relevância do estudo são: ampliar os conhecimentos sobre os cenários que apresentam o *code smell Large Class* e aumentar a acurácia do algoritmo de *Naive Bayes* para detectá-lo, assim, auxiliando na identificação desse *code smell*.

O objetivo deste trabalho é **identificar quais cenários aumentam a acurácia de classificação do algoritmo de *Naive Bayes* em detectar *Large Class***. Para alcançar o objetivo geral dividimos entre os seguintes objetivos específicos: 1) Identificar os possíveis cenários que afetam o algoritmo; 2) Variar os cenários em diferentes execuções e 3) Identificar os cenários que tiveram o melhor resultado.

Com este estudo, espera-se identificar quais são os cenários que constituem o *code smell Large Class*, a fim de determinar quais são os atributos que aumentam a acurácia do algoritmo de *Naive Bayes*. Entregando uma lista com os cenários e suas respectivas acurácias e identificando qual grupo teve melhor resultado.

Esse trabalho está organizado da seguinte forma: na Seção 2 apresenta os conceitos relevantes ao estudo; Seção 3 apresenta os estudos relacionados e Seção 4 apresenta materiais, métodos e cronograma de atividades.

2. Referencial Teórico

Nesta seção apresentamos os principais conceitos abordados pelo estudo, sendo 3 subseções que explicam: 1) o conceito de *code smell* e identifica qual será estudado; 2) o conceito e aplicação do algoritmo de classificação *Naive Bayes* e 3) a definição de acurácia no contexto dos algoritmos de *machine learning*.

2.1. *Code Smells - Large Class*

O termo *code smell* foi utilizado por Martin Fowler (1999) que define estruturas que apresentam possibilidade de refatoração. Este trabalho estuda o *code smell Large Class*,

também conhecido como *God Class*, que consiste em classes que têm mais de um comportamento. Esta característica pode ser observada na quantidade métodos e atributos que ela tem, que diminui a coesão do sistema, aumenta a dependência do sistema com outras classes e fere o conceito de *Single Responsibility Principle* [dos Reis et al. 2020]. Além disso, pode carregar outros *code smells* que são relacionados a ele, como o *Feature Envy* e *Data Class* [Fontana and Zanoni 2011b].

2.2. Algoritmo de classificação supervisionado *Naive Bayes*

Os modelos preditivos, também conhecidos como algoritmos supervisionados, são modelos que estimam um resultado baseado em um conjunto de dados. O funcionamento ocorre a partir da análise de um conjunto de dados previamente rotulados, assim, o algoritmo é capaz de rotular um novo dado seguindo o modelo $D = \{(x_i,f(x_i)), i = 1, ..., n\}$. Como a função f é desconhecida, o algoritmo molda uma função f' que se aproxima da função original [Katti et al. 2021]. Dentre os algoritmos de classificação supervisionados, *K-NN, Naive Bayes, Support Vector Machines (SVM),* Redes Neurais Artificiais e Árvore de Decisão, o selecionado foi o de *Naive bayes*. A escolha ocorreu, porque o algoritmo não precisa de uma regra geral para classificar os dados e permite identificar quais características foram consideradas para a classificação.

O algoritmo de *Naive bayes* é a representação computacional do teorema probabilístico de Bayes que calcula a probabilidade a posteriori de um evento B acontecer dado a ocorrência, a priori, do evento A, assim encontrando a verossimilhança do novo dado [Katti et al. 2021]. Dessa forma, partindo da certeza que ocorreu um evento A, é possível calcular a probabilidade do evento B acontecer pela lei total da probabilidade, que, seguindo os axiomas:[Katti et al. 2021]

- $P(A) > 0$
- Se Ω é o espaço amostral, então $P(\Omega) = 1$
- Se A e B são eventos disjuntos, então $P(A \cup B) = P(A) + P(B)$

  podemos aplicar a lei total da probabilidade (1), que é:

$$P(B) = \sum_{i=1}^{m} P(B|A_i) \times P(A_i) \tag{1}$$

O classificador de bayes apresenta incapacidade em lidar com a interdependência entre atributos, mesmo conhecendo o valor de uma variável não traz informações sobre a outra [Katti et al. 2021]. Para resolver essa situação foi construída a Rede Bayesiana, que monta um grafo de dependência entre as variáveis [Katti et al. 2021], o que facilita a identificação dos pontos que constroem o rótulo. A Rede Bayesiana é implementada seguindo o algoritmo de indução Classificador Bayesiano com k-dependências (k-CBDs) proposto por Sahami (1996), que, dado um conjunto de treinamento $D = \{(x_i, y_i), i = 1, ..., n\}$, com *k* dependências admissíveis, o algoritmo é capaz de classificar um dado e apontar o grau de certeza.

2.3. Acurácia

Com a necessidade de identificar o nível de confiança do modelo é fundamental calcular métricas que o validem. A métrica escolhida para a avaliação do modelo é a acurácia, sendo assim, podemos considerar que a taxa de predições corretas feitas

[Katti et al. 2021]. No cenário dos algoritmos supervisionados, quanto mais dados forem utilizados maior será a taxa de acerto dos modelos, contudo, quanto maior a quantidade de dados maior será o tempo de processamento [Katti et al. 2021]. As bases de dados são estratificadas formando amostras que têm um baixo custo computacional, assim, encontrando um equilíbrio entre o tempo de processamento e a acurácia.

Para calcular a acurácia é necessário entender que o modelo pode apresentar predições incoerentes que podem ser verdadeiro positivo (VP), verdadeiro negativo (VN), falso positivo (FP), falso negativo (FN), que significa que o modelo pode classificar de maneira incoerente as informações para rotulagem, prejudicando a sua acurácia. Para identificar esses casos é calculada a precisão (2), que caracteriza a porcentagem de acerto do modelo e o *recall* (3), que identifica quais instâncias que realmente são identificadas, sendo definidas por:

$$p = \frac{VP}{VP + FP} \tag{2}$$

$$r = \frac{VP}{VP + FN} \tag{3}$$

Com base nesses conceitos podemos definir o cálculo da acurácia (4), que é:

$$a = \frac{VP + VN}{VP + VN + FP + FN} \tag{4}$$

3. Trabalhos Relacionados

Nesta seção são expostos os trabalhos relacionados à identificação de *code smell* com *machine learning*, em que é apresentada a importância do estudo, seus desafios e diferentes abordagens de avaliação dos algoritmos de *machine learning*. Assim, os trabalhos divulgados nesta seção corroboram para a construção deste trabalho.

O trabalho realizado por Guggulothu et. al. (2019) auxilia na fundamentação da importância deste estudo, pois realiza uma comparação entre os algoritmos *J48*, *Random Forest*, *Naive Bayes*, *JRip*, *Sequential minimal optimization (SMO)* e *K-nearest neighbours* (k=2) para identificar os *code smells Shotgun Surgery* e *Message Chain* [Guggulothu and Moiz. 2019]. O estudo apresenta resultados positivos para a aplicação dos algoritmos de *machine learning* para detecção de *code smells*. Para isso, os autores estudaram métricas referente a tamanho, complexidade, coesão, acoplamento, encapsulamento e herança em um nicho de 74 projetos java.

Em relação aos trabalhos pesquisados, o estudo que mais se assemelha ao problema deste trabalho foi realizado por Pecorelli et. al. (2019) que apresentou o desafio que o desequilíbrio dos dados gera para a identificação de *code smell* com algoritmos de *machine learning*. Os autores estudaram os *code smells Large Class, Spaghetti Code, Class Data Should Be Private, Complex Class, Long Method* aplicando as técnicas de balanceamento *ClassBalancer, SMOTE, Resample, CostSensitiveClassifier, OneClassClassifier*. O que auxilia na construção da avaliação do algoritmo de *Naive Bayes* em encontrar o *code smell Large Class*.

Foram identificados três trabalhos que têm a execução do experimento semelhante a este. O primeiro trabalho realizado por Reis et. al (2020), que teve duas etapas. A

primeira foi a classificação manual dos *code smells: God Class, Long Method e Feature Envy* por diversos desenvolvedores, constituindo uma base de dados sólida para a aplicação de algoritmos de *machine learning* na identificação dos *code smells*. A segunda etapa foi a aplicação dos algoritmos de *machine learning* para a previsão dos *code smells*. Obtendo os resultados:, *God Class* (ROC=0.896 com *Naive Bayes*) e *Long Method* (ROC=0.870 com *AdaBoostM1*), *Feature Envy* (ROC=0.570 com *Random Forrest*) [dos Reis et al. 2020]. O trabalho se relaciona a este, pois aplica o algoritmo de *Naive Bayes* para identificar o *code smell Large Class*, e construiu uma base de dados robusta.

O segundo trabalho, realizado por Mohammad et. al. (2020), é um estudo comparativo entre os algoritmos de *machine learning* para detectar os *code smells: Large Class, Long Method* e *Feature Envy*[Mhawish and Gupta 2020]. Para isso, os autores levantam questões de pesquisa referente à eficácia e desempenho dos algoritmos, chegando à conclusão de que os diferentes algoritmos apresentam diferentes acurácias para os mesmos *code smells*. Para chegar a essa conclusão os autores construíram duas bases de dados a partir de 74 projetos, assim, puderam analisar algoritmos supervisionados e não supervisionados [Mhawish and Gupta 2020]. A relevância do trabalho está nos aspectos que a avaliação dos algoritmos supervisionados.

O terceiro trabalho, realizado por Luiz et. al. (2019), estudou diferentes formas de aplicar técnicas de *machine learning* para identificar os *code smells: Large Class, Long Method e Feature Envy*. Os autores separam o estudo em dois experimentos, um que aborda os algoritmos supervisionados e outro com os algoritmos não supervisionados. Com essa divisão houve a necessidade de construir duas bases de dados com os 74 sistemas para aplicar nos diferentes cenários encontrados. O artigo utiliza algoritmos genéticos para avaliar os algoritmos supervisionados, assim, encontrando um modelo mais preciso em detectar os *code smells*. O algoritmo que teve uma acurácia superior aos outros foi o *Random Forest* com a acurácia de 95.31% [Luiz et al. 2019]. A relevância do trabalho está no modo que foi realizada a avaliação do *code smell Large Class*

4. Materiais e Métodos

A pesquisa apresentada neste trabalho é do tipo explicativa quantitativa, a qual busca medir e compreender quais fatores possibilitam o aumento da acurácia do algoritmo de *Naive Bayes*. Para executar a avaliação da acurácia é necessário analisar o comportamento do algoritmo em realizar a classificação com diferentes cenários da base de dados.

4.1. Ambiente

O ambiente de avaliação é composto por uma base de dados construída avaliando 12 projetos *Java open source*, com as ferramentas *Decore*, *PMD*, *JDeodorant* para detectar o *code smell Large Class* e a ferramenta *CkMetrics* para avaliar as métricas, as ferramentas foram escolhidas levando em consideração a sua relevância no cenário acadêmico e a viabilidade na avaliação o projetos java. O particionamento da base de dados é 75% para o treinamento e 25% para verificar a acurácia do algoritmo.

4.2. Questões de Pesquisa

Para alcançar o objetivo foram levantadas 3 questões de pesquisa que seguem pontos de interseção entre os dados encontrados pelas ferramentas e pelo algoritmo de *Naive*

Bayes. **RQ1)** O nível de correlação das métricas afeta na acurácia do algoritmo? **RQ2)** A detecção realizada pelo algoritmo de *Naive Bayes* encontra a mesma quantidade de *Large Class* que as ferramentas? **RQ3)** Os falsos positivos e falsos negativos que foram encontrados pelo algoritmo de *Naive Bayes* foram detectados por alguma das ferramentas?.

5. Procedimentos

Para responder às questões de pesquisa foram abordadas 4 estratégias para a avaliação do algoritmo e como foi realizada a construção da base de dados, que serão discutidas nesta Seção.

5.1. Definição da base de dados

Para a construção da base de dados, foram avaliados 12 projetos *Java open source* disponibilizados no *Qualita Corpus* [Tempero et al. 2010], a escolha do *Qualita Corpus* é ferente a sua relevância acadêmica. Os projetos foram selecionados de acordo com o sucesso de execução das ferramentas *Decore*, *PMD*, *Jdeodorant*, chegando a um número de 3164 arquivos analisados. Para definir se a classe analisada é um *Large Class* ou não, foi realizado um sistema de votação, que considera se existe o *code smell* se pelo menos 2 ferramentas apontarem a existência.

A ferramenta *CK Metrics* possibilita a avaliação da seguintes métricas: *CBO (Coupling between objects), FAN-IN, FAN-OUT, DIT (Depth Inheritance Tree), NOC (Number of Children), Number of fields, Number of methods, Number of visible methods, NOSI (Number of static invocations), RFC (Response for a Class), WMC (Weight Method Class), LOC (Lines of code), LCOM (Lack of Cohesion of Methods),LCOM* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion), LCC (Loose Class Cohesion)* [Aniche 2015]. As métricas foram escolhidas de acordo com que a ferramenta *CkMetrics* disponibiliza.

5.2. Avaliação do algoritmo *Naive Bayes*

A avaliação do algoritmo ocorreu variando a sua execução com 3 versões da base de dados, além da versão completa. As versões foram montadas de acordo com o nível de correlação entre as métricas, as versões tiveram as seguintes características: a primeira com as métricas que obtiveram uma correlação abaixo de 0.2; a segunda versão acima de 0.2; e a terceira acima de 0.3. Assim a versão que obtiver a acurácia mais alta é realizada a avaliação dos Falsos Positivos e Falsos Negativos.

6. Análise dos Resultados

A análise de resultado foi dividida em duas etapas: análise geral dos dados e análise do algoritmo de *Naive Bayes*. Para a análise geral dos dados foram utilizadas duas técnicas: matriz de correlação para identificar a correlação entre as métricas que fazem mais sentido dentro da avaliação do *code smell Large class* e o algoritmo de *K-Means* para obter os grupos de classificação. Em relação a análise do algoritmo de *Naive Bayes* foram utilizadas contas matemáticas de porcentagem e média aritmética.

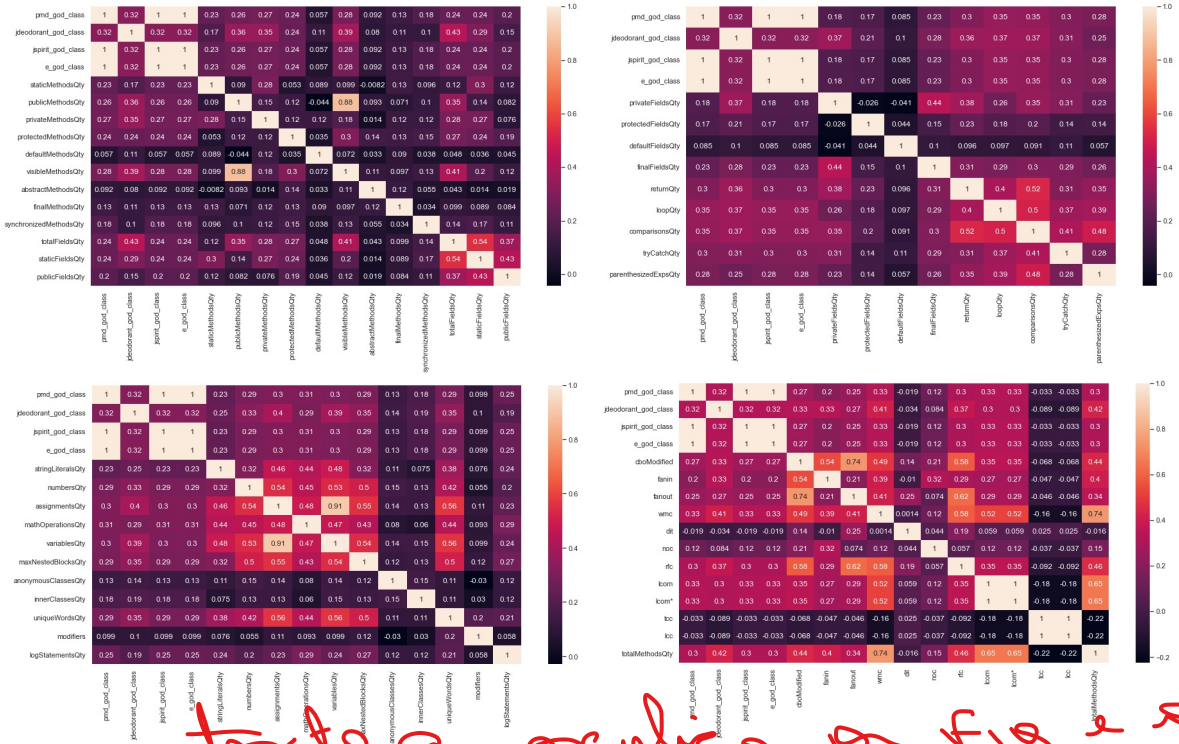

Figura 1. Gráfico de correlação das métricas

6.1. Análise Geral dos Dados

Para realizar a análise geral dos dados foi verificada a correlação das métricas com o resultado das ferramentas e do sistema de votação como demonstrado na figura 1.

É possível observar que as métricas que tiveram uma correlação abaixo de 0.2 com o resultado avaliação do sistema de votação e quando essas métricas com baixa correlação são removidas há uma redução na acurácia do algoritmo de *Naive Bayes*; a redução varia para cada implementação mas em geral reduz como mostra na tabela 1, cenário que será melhor detalhado na Seção 6.2. É interessante comentar que as correlações das métricas relacionadas à avaliação do *JDeodorant* tiveram um contraste em relação a das outras ferramentas, o que significa que a análise dessa ferramenta ocorre de maneira diferente das outras.

Ao filtrar as métricas com uma correlação acima de 0.3 figura 2 podemos observar que são métricas referentes a coesão das classes. Além de moldar quais métricas tem mais relevância para cada ferramenta, um achado é que a quantidade de palavras únicas tem uma correlação forte com a com o *code smell Large Class*

6.2. Análise do Algoritmo de *Naive Bayes*

Levando em consideração que existem diferentes implementações do algoritmo de *Naive Bayes*, foram avaliados os diferentes cenários para a avaliação da acurácia de cada implementação: *Gaussian Naive Bayes(GNB)*, Multivariate Bernoulli(MB), *ComplementNB(CNB)* e *MultinomialNB(MNB)*. Podemos observar na tabela 1, que quanto maior e mais diversa é a correlação entre as métricas maior é a acurácia do algoritmo. Nos casos

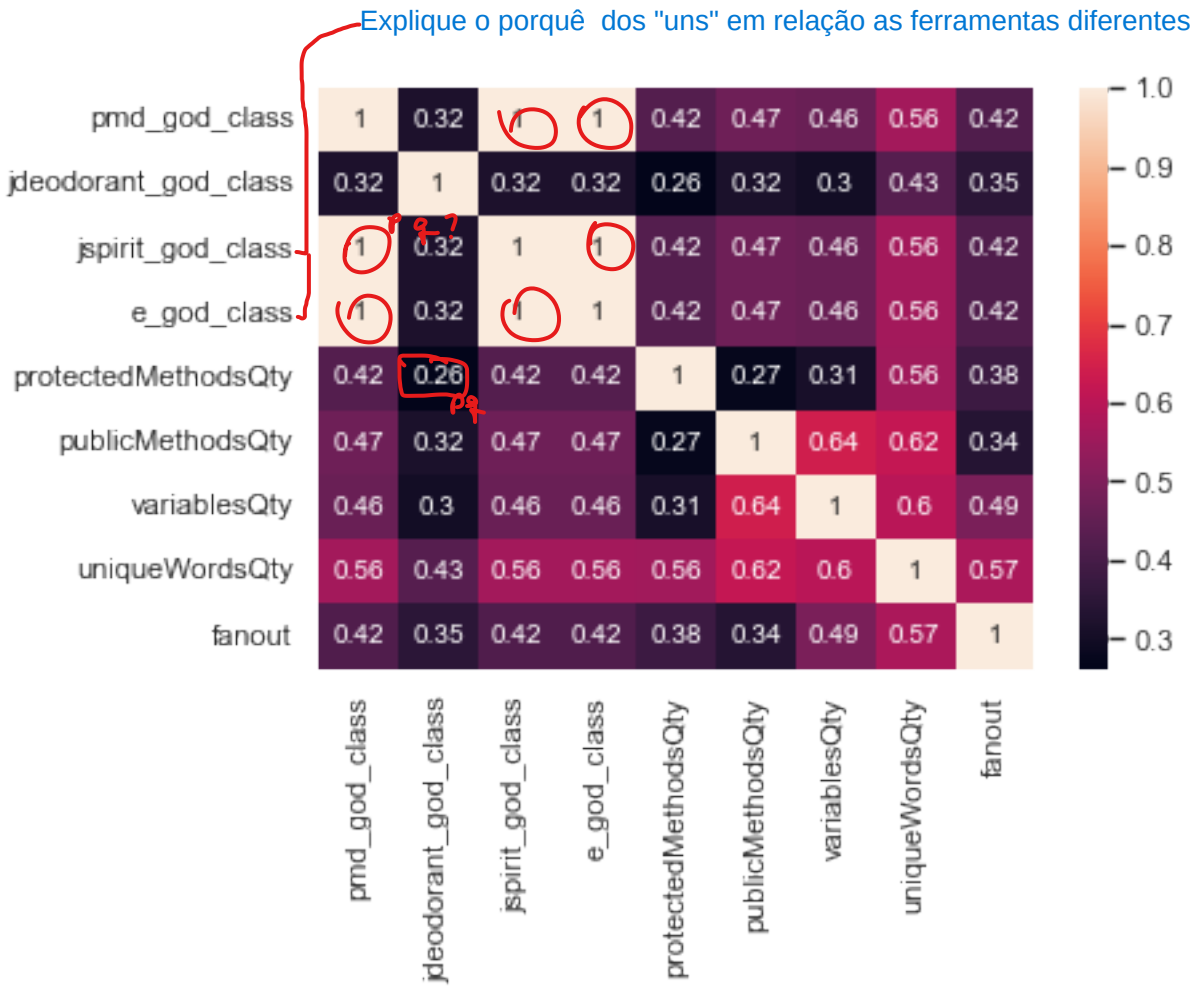

Figura 2. Gráfico de correlação das principais métricas

que a acurácia chegou em 1.0 demonstra que o modelo realizou suposições inadequadas e o algoritmo não representa a classificação corretamente para o problema proposto, e pelo fato da base de dados ter uma distribuição heterogênea, pois a variância das métricas dispersam de 1 à 9 da média, por isso a implementação *ComplementNB* é a que melhor representa o cenário estudado, pois nos outros cenários o algoritmo realiza suposições, o que afeta a confiabilidade dos resultados quando os dados não são lineares. O que responde a primeira questão de pesquisa, o nível de correlação das métricas afeta na acurácia do algoritmo?, a correlação entre as métricas afeta a acurácia do algoritmo, pois dentro do cenário escolhido quanto maior a correlação entre as métricas maior é a acurácia do algoritmo.

Implementação	Base de dados completa	Base de dados particionada abaixo de 0.2 de correlação	Base de dados particionada acima de 0.2 de correlação	Base de dados particionada acima de 0.3 de correlação
GNB	0.9924	1.0	0.9646	0.9886
MB	0.9734	1.0	0.9810	1.0
CNB	0.9064	0.7661	0.9367	0.9393
NMB	0.9077	0.8811	0.9367	0.9393

Ao avaliar a implementação *ComplementNB* que teve a acurácia de 0.9393, 0.0474% dos casos foram Falsos Negativos e 0.0053%. Com essa análise foi possível responder a segunda questão de pesquisa, a detecção realizada pelo algoritmo de *Naive Bayes* encontra a mesma quantidade de *Large Class* que as ferramentas?, que foi observado que não, as ferramentas encontram 133 a mais do que o algoritmo encontra.

Para responder a terceira questão de pesquisa, os FPs e FNs que foram encontrados pelo algoritmo de *Naive Bayes* foram detectados por alguma das ferramentas?. Foi realizada a análise apenas no cenário dos FPs e FNs.

Dos Falsos Positivos foi observado que 0.1765% dos casos foram encontrados pela ferramenta *JDeodorant* e 0.8235% dos casos não foram detectados por nenhuma ferramenta. É interessante relembrar o fato que as métricas relacionadas à avaliação do *JDeodorant* tiveram um contraste em relação a outras ferramentas. Além disso, os FPs tiveram uma variância das métricas entre 5 a 2144, o que mostra que quanto mais dispersos os dados maior a probabilidade de erro do algoritmo. Focando na métrica que teve a maior variância foi o *lcom*, sendo uma métrica relevante para encontrar o *code smell Large Class*.

Em relação aos FNs foi observado que 0.3134% foram detectados por pelas ferramentas *PMD e Jspirit* e 0.6866% foram detectados pelas três ferramentas. Sendo que a variância das métricas varia entre 4 a 214, o que mostra que quanto mais dispersos os dados maior a probabilidade de erro do algoritmo. Focando na métrica que teve a maior variância foi o *lcom*, sendo uma métrica relevante para encontrar o *code smell Large Class*.

7. Conclusão

Com este estudo foi possível observar que os falsos positivos e falsos negativos são compostos por uma dispersão dos dados, e quanto menos dispersos e maior a diversidade e a correlação maior é a acurácia do algoritmo. Foi avaliado também que as métricas que compõem o cenário que tem maior acurácia são métricas que auxiliam a identificação da coesão e do acoplamento da classe. Como trabalhos futuros pode-se avaliar como ocorre a variação das métricas. Com a finalidade de encontrar o ponto de erro do algoritmo em detectar o *code smell Large Class*.

Referências

Aniche, M. (2015). *Java code metrics calculator (CK)*. Available in https://github.com/mauricioaniche/ck/.

dos Reis, J. P., e Abreu, F. B., and de Figueiredo Carneiro, G. (2020). Crowdsmelling: The use of collective knowledge in code smells detection. *CoRR*, abs/2012.12590.

Fontana, F. A. and Zanoni, M. (2011a). On investigating code smells correlations. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 474–475.

Fontana, F. A. and Zanoni, M. (2011b). On investigating code smells correlations. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 474–475.

Guggulothu, T. and Moiz., S. A. (2019). Detection of shotgun surgery and message chain code smells using machine learning techniques. *IJRSDA*, pages 34–50.

Katti, F., Lorena, A. C., Gama, J., Almeida, T. A. d., and Carvalho, A. C. P. L. F. d. (2021). *Inteligência artificial uma abordagem de aprendizado de máquina*. Rio de Janeiro LTC, 2 edition.

Luiz, F. C., de Oliveira Rodrigues, B. R., and Parreiras, F. S. (2019). Machine learning techniques for code smells detection: An empirical experiment on a highly imbalanced setup. In *Proceedings of the XV Brazilian Symposium on Information Systems*, SBSI'19, New York, NY, USA. Association for Computing Machinery.

Mhawish, M. Y. and Gupta, M. (2020). Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. In *Journal of Computer Science and Technology*, page 1428–1445.

Oliveira, D., Assunção, W. K. G., Souza, L., Oizumi, W., Garcia, A., and Fonseca, B. (2020). Applying machine learning to customized smell detection: A multi-project study. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, SBES '20, page 233–242, New York, NY, USA. Association for Computing Machinery.

Pecorelli, F., Di Nucci, D., De Roover, C., and De Lucia, A. (2019a). On the role of data balancing for machine learning-based code smell detection. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, MaLTeSQuE 2019, page 19–24, New York, NY, USA. Association for Computing Machinery.

Pecorelli, F., Palomba, F., Di Nucci, D., and De Lucia, A. (2019b). Comparing heuristic and machine learning approaches for metric-based code smell detection. In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, page 93–104. IEEE Press.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345.