Low-Cost Machine Learning for Effective and Efficient Bad Smells Detection

J.S.L Figuerêdo, V. T. Sarinho, R. T. Calumby

Universidade Estadual de Feira de Santana, Brasil
jslfigueredo@ecomp.uefs.br, vsarinho@uefs.br, rtcalumby@uefs.br

Abstract. Bad smells are characteristics of software that indicate a code or design problem which can make information system hard to understand, evolve, and maintain. To address this problem, different approaches, manual and automated, have been proposed over the years, including more recently machine learning alternatives. However, despite the advances achieved, some machine learning techniques have not yet been effectively explored, such as the use of feature selection techniques. Moreover, it is not clear to what extent the use of numerous source-code features are necessary for reasonable bad smell detection success. Therefore, in this work we propose an approach using low-cost machine learning for effective and efficient detection of bad smells, through explicit feature selection. Our results showed that the selection allowed to statistically improve the effectiveness of the models. For some cases, the models achieved statistical equivalence, but relying on a highly reduced set of features. Indeed, by using explicit feature selection, simpler models such as Naive Bayes became statistically equivalent to robust models such as Random Forest. Therefore, the selection of features allowed keeping competitive or even superior effectiveness while also improving the efficiency of the models, demanding less computational resources for source-code preprocessing, model training and bad smell detection.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**.

Keywords: bad smell, machine learning, feature selection, data mining

1. INTRODUÇÃO

Sistemas de software estão se tornando cada vez mais complexos [Di Nucci et al. 2018]. Essa complexidade e a forte concorrência de mercado impulsionam a necessidade das organizações realizarem atualizações contínuas nos seus códigos-fonte [Lehman 1980]. Essas atualizações costumam ocorrer com alta frequência, com os desenvolvedores realizando inúmeras alterações em um curto período de tempo. Estas altas demandas sobre os desenvolvedores acabam reduzindo sua capacidade em aderir às boas práticas e princípios de programação para entregar produtos funcionais no menor tempo possível, porém ainda imaturos, por exemplo, em termos estruturais e documentais, de manutenibilidade e de reúso [Kruchten et al. 2012]. Este processo pode acarretar nas chamadas dívidas técnicas do código-fonte [Cunningham 1992], as quais podem ocasionar problemas mais sérios no processo de evolução e manutenção dos sistemas. Uma das características que indicam a presença de dívidas técnicas em sistemas de software é presença dos chamados *bad smells*.

O conceito de *bad smells* foi definido inicialmente por [Martin Fowler 1999] como indicadores de possíveis problemas no código ou no projeto de um sistema. Ou seja, refere-se a partes do código fonte que violam princípios básicos de projeto de um sistema, tais como abstração, hierarquia, encapsulamento, modularidade, entre outros [Booch et al. 2007]. Vários fatores podem contribuir para o surgimento de *bad smells*. Estes problemas podem ser frutos da pressão exercida sob os desenvolvedores para entregar o produto em um curto prazo, mas também podem ser desencadeados por outros motivos, tais como a inexperiência dos desenvolvedores, falhas no processo de análise do domínio ou

integração inadequada entre módulos do sistema [Danphitsanuphan and Suwantada 2012].

Independentemente da motivação, sabe-se que a presença de *bad smells* pode impactar negativamente na qualidade geral do software, a exemplo da dificuldade de manutenção e compreensão, influenciando diretamente na sua evolução [Arcelli Fontana et al. 2016; Sjøberg et al. 2013; Yamashita and Moonen 2012]. Adicionalmente, considerando a complexidade do problema, suas sérias consequências e a imensa quantidade de código a ser verificada, muitos métodos têm sido propostos para a identificação automática de *bad smells*. Dentre os principais benefícios objetivados, destaca-se que após a detecção do problema, por exemplo, o processo de refatoração pode ser aplicado para solucioná-los. A Refatoração é uma técnica que melhora a estrutura interna do código (qualidade de projeto) sem alterar o comportamento externo do software [Martin Fowler 1999]. Esse processo busca alcançar alta qualidade, alto desempenho, baixo custo, fácil reutilização e o fácil desenvolvimento de software [Roy et al. 2016]. Para contribuir com esta tarefa, detectores de *bad smells* têm sido propostos [Moha et al. 2010; Liu et al. 2013; Palomba et al. 2015].

Contudo, apesar das abordagens existentes apresentarem desempenho promissor, estudos mais recentes identificaram um conjunto de limitações que podem ameaçar a sua utilização prática. Dentre essas limitações destacam-se a subjetividade do método de análise, a disparidade de resultados gerados por diferentes detectores, e a necessidade de ajustes de parâmetros [Di Nucci et al. 2018; Mori et al. 2017]. Para superar essas limitações, técnicas de aprendizado de máquina tem sido adotadas [Arcelli Fontana et al. 2016; Di Nucci et al. 2018; Azeem et al. 2019; Cruz et al. 2020] com resultados promissores. A utilização destas técnicas traz mais flexibilidade, especialmente devido à sua capacidade de adaptação e evolução a partir de novos dados. Entretanto, ainda demandam avanços para viabilizar sua aplicação prática, principalmente considerando a eficácia de detecção dos *bad smells*. Neste contexto, algumas técnicas de aprimoramento do aprendizado de máquina podem contribuir para a melhoria da capacidade de detecção de *bad smells*, a exemplo da seleção de *features* [Al-Shaaby et al. 2020].

Em um conjunto de dados com múltiplas *features*, é possível que nem todas contribuam significativamente para a detecção de *bad smells*. Muitas delas podem ser, inclusive, irrelevantes ou redundantes entre si, trazendo custo desnecessário para a análise de repositórios de código em larga escala. A seleção de *features* pode ser empregada para reduzir o tamanho do conjunto de dados, possibilitando menor custo de treinamento, além de possível aprimoramento da eficácia [Han et al. 2011]. Deste modo, considerando os desafios apresentados, neste trabalho propomos uma abordagem de detecção de *bad smells* otimizada por meio da seleção automática de *features*. Assim, busca-se aprimorar o processo de identificação de sistemas de software com potenciais *bad smells*, utilizando algoritmos de aprendizado de máquina eficazes e eficientes, além de permitir redução do custo do processo de análise e extração de *features* a partir dos repositórios de código.

2. TRABALHOS RELACIONADOS

Um dos primeiros trabalhos utilizando técnicas de aprendizado de máquina foi desenvolvido por [Kreimer 2005]. O autor propôs o uso de árvores de decisão para detecção de *bad smells* do tipo *Large Class* e *Long Method*. A ferramenta desenvolvida foi especializada para programas escritos na linguagem Java. As análises foram realizadas em dois sistemas de software: sistema IYC e pacote Weka. Os resultados encontrados indicam que o modelo alcançou altos valores de acurácia, sugerindo que o aprendizado de máquina pode ser uma alternativa a outras técnicas de detecção de *bad smell*.

Em [Arcelli Fontana et al. 2016], os autores utilizaram algoritmos de aprendizado de máquina para detectar quatro categorias de *bad smells*, sendo elas: *Data Class*, *Large Class*, *Feature Envy* e *Long Method*. Para isso, os autores fizeram uso de 32 algoritmos de classificação e 74 sistemas, com 1986 amostras de *bad smells* validadas manualmente. Considerando a *Precision* e a medida F_1, os autores evidenciaram que a maioria dos classificadores obtiveram eficácia superior a 95%, sendo o algoritmo

J48 e *Random Forest*, os mais eficazes.

Contudo, em estudo posterior realizado por [Di Nucci et al. 2018], os autores identificaram que o conjunto de dados preparados por [Arcelli Fontana et al. 2016] não representavam um cenário do mundo real. Existia uma diferença expressiva entre as métricas dos sistemas com *bad smell* e àqueles sem a presença desse problema, de modo que qualquer técnica de aprendizado de máquina poderia facilmente distinguir as duas classes. Além disso, [Arcelli Fontana et al. 2016] havia construído quatro conjuntos de dados, um para cada dívida técnica, com cada conjunto de dados possuindo elementos de código (instâncias) afetados por apenas aquele tipo específico de *bad smell* ou componentes sem esse problema. Portanto, a base de dados precisava ser ajustada para um cenário mais realístico. Assim, [Di Nucci et al. 2018] replicaram o estudo de [Arcelli Fontana et al. 2016] com uma configuração de conjunto de dados diferente, que incluía, por exemplo, instâncias de mais de um tipo de *bad smell*. Os resultados revelaram que ao utilizar essa nova configuração, as técnicas de aprendizado de máquina apresentaram limitações críticas. Os modelos utilizados alcançaram uma acurácia média de somente 76%, bem aquém do encontrando no estudo de [Arcelli Fontana et al. 2016], indicando a necessidade de novos métodos e experimentos.

Observando as limitações apontadas por [Di Nucci et al. 2018], [Guggulothu and Moiz 2020] desenvolveram um estudo que buscava investigar a baixa eficácia das técnicas de aprendizado de máquina após as modificações efetuadas no *dataset*. Os autores identificaram que o conjunto de dados modificados por [Di Nucci et al. 2018] tinham algumas instâncias que são idênticas, mas com rótulos de classe diferentes (*bad smell* e não *bad smell*), denominado de disparidade. Por exemplo, na fusão de nível de método realizada por [Di Nucci et al. 2018], se o conjunto de dados do *long method* possui uma instância que é *bad smell*, e se a mesma instância está no conjunto de dados do *feature envy*, essa instância é replicada no conjunto do *long method* como não *bad smell*. Essas instâncias de disparidade podem confundir os algoritmos de aprendizado de máquina, induzindo à perda de eficácia das técnicas de classificação, o que pode levar a falsas interpretações. Para enfrentar esta limitação, [Guggulothu and Moiz 2020] removeram essas disparidades e realizaram os experimentos utilizando o conjunto de dados correspondentes aos *bad smells long method* e *feature envy*. Assim, semelhante ao encontrado no trabalho de [Arcelli Fontana et al. 2016], o trabalho desenvolvido por [Guggulothu and Moiz 2020] alcançou uma acurácia média de 95% após a remoção dessas instâncias díspares, valores bem superiores ao encontrado por [Di Nucci et al. 2018].

O nosso trabalho possui objetivo semelhante aos trabalhos mencionados, mas aborda o problema de *bad smell* sob a perspectiva da demanda por grandes quantidades de *features*, o impacto disso na eficácia preditiva de *bad smell* e o potencial de aprimoramento de eficácia e eficiência. Diferentemente dos trabalhos anteriores, neste estudo, desenvolvemos uma abordagem que considera a seleção explícita de *features* para redução de custo computacional e melhoria de eficácia dos métodos de predição. Além disso, aplicamos um processo de avaliação rigoroso para verificar a efetividade e generalização da abordagem.

3. ABORDAGEM PROPOSTA E CONFIGURAÇÃO EXPERIMENTAL

Para o tipo de dado utilizado neste trabalho, as *features* ou variáveis independentes são métricas do código-fonte, enquanto a variável dependente corresponde ao rótulo que indica a presença ou não de *bad smells*. Considerando o grande número de *features* possíveis e complementariedade de diferentes modelos de classificação, a abordagem proposta baseia-se em um processo de seleção de *features*. O processo de seleção pode ser feito de forma manual por um especialista de domínio. No entanto, isso pode ser uma tarefa difícil e demorada, especialmente quando o comportamento dos dados não é bem conhecido. Por este motivo, frequentemente é feito de forma automática a partir de um conjunto vasto de técnicas de seleção de *features* ou redução de dimensionalidade. As principais técnicas utilizadas são: *Univariate Selection*, *Recursive Feature Elimination*,*Principle Component Analysis (PCA)*, *Feature importance*, Forward selection, entre outros [Brownlee 2016; Han et al. 2011].

Considerando sua simplicidade, neste trabalho utilizamos a técnica de *Forward selection*. Esta técnica consiste basicamente em três passos: *i*) O procedimento começa com um conjunto vazio de *features* selecionadas. *ii*) A melhor das *features* originais é determinada e adicionada ao conjunto de *features* selecionadas. Em cada iteração subsequente, a melhor dentre as *features* originais restantes é adicionada ao conjunto; *iii*) Este processo é finalizado quando uma regra de parada pré-especificada é alcançada ou até que todas as *features* em consideração sejam selecionadas. Em relação a esta técnica, a escolha da próxima *feature* a ser adicionada no conjunto reduzido utilizou a medida F_1 como critério. Com relação à condição de parada, nenhum critério foi estabelecido, isto é, a seleção é finalizada somente quando todas as *features* em consideração são incluídas. A seleção final das *features* é feita aplicando-se um filtro que determina qual subconjunto de *features* mais maximizaram a medida F_1. Após a etapa de seleção de *features*, realiza-se a etapa de classificação, a qual pode utilizar diferentes algoritmos de classificação. Em especial, neste trabalho utilizamos 8 algoritmos, separados em duas categorias: algoritmos clássicos (*k-nearest Neighbors (KNN)*, *Naíve Bayes (NB)*, *Logistic Regression (LR)*, *Support Vector Machines (SVM)* e Árvore de Decisão (AD)); E algoritmos baseados em ensemble (*Gradient Boosting Trees (GBT)*, *Random Forest (RF)* e *Tree Ensemble Learner (TLE)*).

Os experimentos foram realizados usando o conjunto de dados, em nível de método, do dataset *bad smell Long method*, desenvolvido inicialmente em [Arcelli Fontana et al. 2016] e refinado em [Di Nucci et al. 2018; Guggulothu and Moiz 2020]. Em [Arcelli Fontana et al. 2016], os autores analisaram sistemas oriundos do *Qualitas Corpus* [Tempero et al. 2010], na sua versão *20120401r*. A coleção é composta por 111 sistemas escritos em Java, caracterizados por diferentes tamanhos e pertencentes a diferentes domínios de aplicação. Entre esses 111 sistemas, 74 foram utilizados para criar o *dataset*, os outros 37 restantes foram descartados, pois não era possível compila-los, de modo que não era possível calcular os valores das métricas corretamente. Para cada um desses sistemas, os autores calcularam um conjunto métricas em nível de classe (61) e em nível de método (82). Considerando o problema de *bad smell Long method* as métricas em nível de métodos são consideradas mais adequadas e portanto foram as utilizadas neste trabalho.

A versão dos dados que foi utilizada nos experimentos corresponde àquela utilizada em [Guggulothu and Moiz 2020], a qual é uma adaptação do *dataset* utilizado em [Di Nucci et al. 2018], realizada no intuito de adequar o conjunto de dados para cenários de casos de uso reais. Em sua versão refinada, o *dataset*[1] possui 708 instâncias, sendo 140 rotuladas como incluindo *long method* e as outras 568 restantes rotuladas como sem *long method*. As 82 métricas em nível de método calculadas cobrem diferentes aspectos do código, sendo elas complexidade, coesão, tamanho e acoplamento. Além disso, métricas personalizas foram criadas por [Arcelli Fontana et al. 2016], por exemplo, número de métodos abstratos, número de métodos construtores, número de métodos finais, entre outras.

A Figura 1 apresenta o processo experimental realizado. Particionou-se o conjunto de dados utilizando a proporção 70/30, sendo 70% para o treinamento e os 30% restantes para o conjunto de teste. Uma etapa de pré-processamento foi realizada para normalização dos dados via método *Min-Max* considerando dados de treino. O conjunto de treino foi utilizado para gerar os modelos e também para selecionar as *features*. A seleção de *features* é feita para cada algoritmo separadamente, por meio da validação cruzada baseada em *k-folds*, com $k = 10$. O conjunto de teste, por sua vez, é utilizado para avaliar a eficácia geral dos modelos. Para efeito de comparação e avaliação do impacto do processo de seleção de *features*, a eficácia dos classificadores individuais também foi avaliada considerando todos os atributos disponíveis sobre os dados, ou seja, sem seleção de *features*. O processo experimental descrito na Figura 1 foi executado 10 vezes para cada algoritmo. Deste modo, o resultado reportado neste trabalho corresponde à média dessas execuções, além do desvio padrão.

Os experimentos foram executados usando a plataforma KNIME[2]. Os algoritmos utilizados nos

[1]https://github.com/thiru578/Datasets-LM-FE/blob/master/long-method.arff - Acessado em Agosto, 2021
[2]https://www.knime.com/ - Acessado em Julho, 2021.

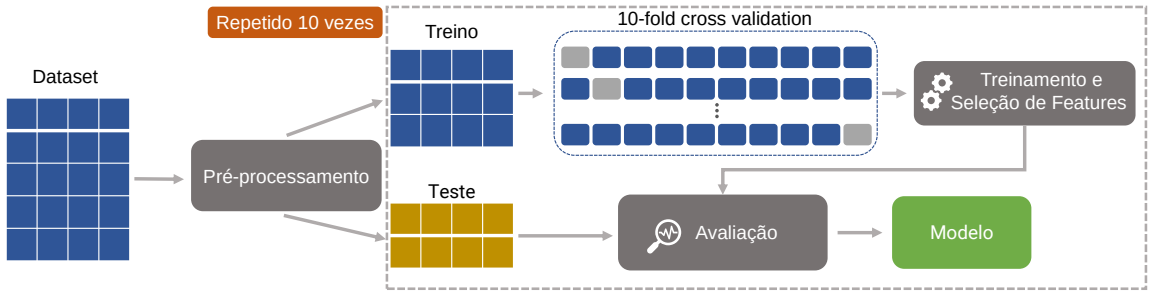

Fig. 1. Workflow experimental conduzido neste trabalho.

experimentos utilizaram a configuração padrão de parâmetros do KNIME. Os testes estatísticos foram realizados no software SPSS[3].

A eficácia dos algoritmos foi avaliada com base em medidas clássicas, como *Precision*, *Recall*, F_1 e Acurácia. A *Precision* quantifica a porção de amostras preditas como pertencentes à classe de interesse (bad smell) e que são de fato *bad smell*. Por outro lado, o *Recall* quantifica a porção de amostras da classe de interesse que foram corretamente identificadas como pertencentes à essa classe. A medida F_1, por sua vez, é computada como a média harmônica ponderada da *Precision* e do *Recall*. Por fim, a Acurácia corresponde à fração total de predições que o modelo acertou. Para cada medida, reportamos a média para as classes. Para a comparação estrita da nossa abordagem, os resultados obtidos foram comparados com os algoritmos sem a seleção de *features*, e também com o melhor resultado obtido por [Guggulothu and Moiz 2020]. Em [Guggulothu and Moiz 2020], os autores utilizaram métodos baseados em ensemble disponíveis no pacote de software Weka[4], sendo o RF o algoritmo que obteve o melhor resultado. Adicionalmente, aplicamos o teste t de *Student* para amostras independentes, com confiança de 95%.

4. RESULTADOS E DISCUSSÃO

Inicialmente, apresentamos os resultados obtidos utilizando os algoritmos clássicos, seguido pelos resultados obtidos com algoritmos baseados em ensemble. Para estas duas categorias são apresentados os resultados obtidos na validação cruzada e também no conjunto final de testes. Por fim, são apresentados alguns resultados adicionais referentes ao impacto da seleção de *features* em algoritmos como o NB, comparando-o com o RF, que é tipicamente considerado mais eficaz, mas computacionalmente mais custoso. Como principais *baselines* são utilizados as versões dos algoritmos sem a seleção de *features*, bem como o melhor resultado obtido em [Guggulothu and Moiz 2020]. Os resultados em [Guggulothu and Moiz 2020] são reportados em percentual, com o melhor modelo alcançando $Acurácia = 95{,}9\%$ e $F_1 = 96,0\%$

4.1 Algoritmos clássicos

Após o processo de seleção, houve uma grande redução no número de *features*, resultando em média: $KNN = 11$, $NB = 15$, $LR = 26$, $SVM = 28$ e $AD = 18$. Mesmo considerando o algoritmo que teve a menor redução no número de *features*, ou seja, o SVM, observa-se que a redução ainda assim foi cerca de 65%. No caso de maior redução, para o KNN, a redução foi de cerca de 86%.

Considerando o resultado no conjunto de validação (Tabela I), verificou-se que os modelos que utilizaram a seleção de *features* obtiveram os melhores resultados. Em comparação ao trabalho desenvolvido por [Guggulothu and Moiz 2020], cujo processo de avaliação foi similar ao realizado aqui,

[3]https://www.ibm.com/br-pt/analytics/spss-statistics-software - Acessado em Julho, 2021.
[4]Weka é software que possui uma coleção de algoritmos de aprendizado de máquina para tarefas de mineração de dados.

alguns dos modelos que utilizaram a seleção de *features* apresentaram um desempenho similar em termos de F_1 e ainda valores superiores de acurácia. Por exemplo, o NB alcançou valores médios de $F_1 = 95,3\%$ e Acurácia $96,8\%$ utilizando em média apenas 15 *features* ao invés das 82 utilizadas em [Guggulothu and Moiz 2020].

Tabela I. Eficácia dos modelos clássicos. Os melhores resultados são destacados em negrito. A significância estatística também é relatada: a superioridade é destacada com "*"; Todo o resto são equivalentes.

		Validação Cruzada				Teste			
Modelo	Seleção	F_1	*Precision*	*Recall*	Acurácia	F_1	*Precision*	*Recall*	Acurácia
KNN	Não	0,830 (±0,07)	0,885 (±0,07)	0,796 (±0,07)	0,904 (±0,03)	0,781 (±0,04)	0,863 (±0,04)	0,781 (±0,04)	0,897 (±0,02)
KNN	Sim	*0,951 (±0,04)	*0,941 (±0,04)	*0,962 (±0,04)	*0,968 (±0,02)	*0,896 (±0,03)	0,885 (±0,02)	*0,910 (±0,04)	*0,935 (±0,02)
NB	Não	0,872 (±0,06)	0,859 (±0,06)	0,889 (±0,06)	0,915 (±0,04)	0,886 (±0,02)	0,850 (±0,03)	0,866 (±0,03)	0,910 (±0,02)
NB	Sim	***0,953** (±0,03)	*0,936 (±0,04)	***0,974** (±0,02)	***0,968** (±0,02)	*0,924 (±0,02)	*0,907 (±0,02)	***0,945** (±0,02)	*0,951 (±0,02)
LR	Não	0,899 (±0,05)	0,904 (±0,06)	0,894 (±0,06)	0,934 (±0,03)	0,899 (±0,01)	0,901 (±0,02)	0,898 (±0,01)	0,938 (±0,01)
LR	Sim	*0,937 (±0,04)	0,932 (±0,05)	*0,942 (±0,04)	*0,958 (±0,03)	0,898 (±0,03)	0,897 (±0,03)	0,899 (±0,02)	0,937 (±0,01)
SVM	Não	0,915 (±0,05)	0,905 (±0,05)	0,926 (±0,05)	0,943 (±0,03)	0,917 (±0,02)	0,905 (±0,02)	0,931 (±0,02)	0,948 (±0,01)
SVM	Sim	*0,944 (±0,04)	*0,933 (±0,04)	*0,956 (±0,04)	*0,962 (±0,03)	**0,926** (±0,03)	**0,916** (±0,03)	0,939 (±0,02)	**0,955** (±0,02)
AD	Não	0,899 (±0,05)	0,901 (±0,05)	0,896 (±0,06)	0,935 (±0,03)	0,896 (±0,01)	0,898 (±0,02)	0,895 (±0,02)	0,935 (±0,01)
AD	Sim	*0,949 (±0,04)	***0,943** (±0,04)	*0,956 (±0,04)	*0,967 (±0,03)	0,897 (±0,01)	0,898 (±0,02)	0,897 (±0,02)	0,935 (±0,01)

Destaca-se que NB obteve resultados superiores ao trabalho de [Guggulothu and Moiz 2020] (cujo melhor resultado foi obtido pela RF disponível no Weka). Esse resultado reflete a importância da seleção explícita de *features*, ao possibilitar o aprimoramento da eficácia, mesmo de um algoritmo considerando mais simples, como o NB. Além disso, ressalta-se que este resultado foi obtido utilizando um conjunto reduzido de *features*, o que, em termos práticos, é mais desejado para aplicações reais. Dado que se conhece quais *features* são as mais importantes, a sua extração se torna mais simples e menos custosa para utilização rotineira.

A partir dos resultados do conjunto de teste (Tabela I), observa-se que os modelos tiveram um comportamento similar ao reportado no conjunto de validação. Considerando-se as múltiplas medidas de avaliação, observou-se que a seleção de *features* permitiu aprimoramento da eficácia de detecção dos *bad smells*. Especificamente para o KNN e o NB, houve uma diferença estatisticamente significativa ao utilizar a seleção de *features*. Por sua vez, para a LR, SVM e AD, os resultados demonstraram equivalência estatística. No entanto, destaca-se que os algoritmos que fizeram uso da seleção obtiveram desempenho equivalente, mesmo utilizando um conjunto consideravelmente reduzido de *features*. Ou seja, além de manterem eficácia equivalente, estes algoritmos seriam executados com demandas inferiores de recursos computacionais para construção de modelos preditivos e para execução prática da predição sobre os códigos-fonte de interesse.

Vale destacar que a seleção de *features* permitiu um melhoria estatisticamente significativa em termos de *Recall* para o KNN e NB. Isso torna-se ainda mais valioso, em razão do *dataset* utilizado ser altamente desbalanceado. Isso evidencia, que mesmo sendo expostos a poucos exemplos da classe positiva, os modelos conseguiram aprender a identificar esse tipo de problema. Demonstra-se, deste modo, que as *features* selecionadas permitiram caracterizar efetivamente os casos positivos, ou seja, a ocorrência de *bad smells*.

4.2 Algoritmos baseados em ensemble

Para estes algoritmos também houve uma grande redução no número de *features*, resultando em média: *GBT = 23*, *RF = 10* e *TLE = 12*. Mesmo para o algoritmo que selecionou o maior número de *features*, o percentual de redução foi de aproximadamente 71%.

Considerando os resultados da validação cruzada, Tabela II, observa-se que os algoritmos que utilizaram a seleção de *features* foram estatisticamente superiores ao modelos sem a seleção. Além de alcançarem significância estatística no conjunto de validação, apesar de resultados marginalmente inferiores, os modelos com seleção obtiveram equivalência estatística no conjunto de teste, mesmo

utilizando um conjunto altamente reduzido de *features*. Destaca-se ainda que estes algoritmos incorporam processos implícitos de seleção de features, o que pode ter limitado o poder de aprimoramento de eficácia do processo de seleção explícita aqui proposto. Contudo, a seleção aqui proposta tem impacto direto na redução de demanda computacional, especialmente para estes algoritmos baseados em ensemble que são naturalmente mais custosos.

Similar àquilo descrito para os algoritmos clássicos, os modelos com a seleção alcançaram valores equivalentes ou superiores ao trabalho de [Guggulothu and Moiz 2020] em termos de F_1 e Acurácia no processo de validação cruzada. O comportamento geral observado é altamente desejado, pois a nossa abordagem conseguiu ser efetiva ao mesmo tempo que utilizou um número bastante reduzido de *features*.

Tabela II. Eficácia dos modelos baseados ensemble. Os melhores resultados são destacados em negrito. A significância estatística também é relatada: a superioridade é destacada com "*"; Todo o resto são equivalentes.

		Validação Cruzada				Teste			
Modelo	Seleção	F_1	*Precision*	*Recall*	Acurácia	F_1	*Precision*	*Recall*	Acurácia
GBT	Não	0,924 (±0,04)	0,915 (±0,05)	0,934 (±0,05)	0,949 (±0,03)	0,921 (±0,02)	0,910 (±0,02)	0,935 (±0,02)	0,949 (±0,01)
GBT	Sim	***0,958** (±0,04)	***0,946** (±0,04)	*0,973 (±0,03)	***0,972** (±0,02)	0,911 (±0,03)	0,904 (±0,02)	0,921 (±0,03)	0,944 (±0,02)
RF	Não	0,936 (±0,04)	0,923 (±0,05)	0,952 (±0,04)	0,957 (±0,03)	**0,934** (±0,02)	**0,919** (±0,02)	**0,955** (±0,02)	**0,958** (±0,01)
RF	Sim	*0,950 (±0,03)	*0,934 (±0,04)	*0,968 (±0,03)	*0,966 (±0,02)	0,931 (±0,02)	0,916 (±0,02)	0,950 (±0,02)	0,955 (±0,01)
TLE	Não	0,936 (±0,05)	0,922 (±0,05)	0,953 (±0,05)	0,957 (±0,03)	0,933 (±0,02)	0,917 (±0,02)	0,953 (±0,01)	0,957 (±0,01)
TLE	Sim	*0,956 (±0,03)	*0,937 (±0,04)	***0,979** (±0,02)	*0,971 (±0,02)	0,928 (±0,02)	0,915 (±0,02)	0,944 (±0,02)	0,954 (±0,01)

4.3 Análise Adicional

Dentre os algoritmos clássicos, levando em consideração a sua simplicidade, o algoritmo que obteve o melhor desempenho com seleção de *features* foi o NB. Por outro lado, para os baseados em ensemble, o melhor modelo foi a RF sem a seleção explícita de *features*. Vale ressaltar que o algoritmo RF foi o modelo que obteve o melhor resultado em [Guggulothu and Moiz 2020]. Neste sentido, dado o baixo custo computacional do NB, característica atrativa para uma aplicação prática, realizamos um comparativo específico entre eles.

Para verificar a existência de diferença estatisticamente significativa entre os algoritmos também utilizamos o teste t de *Student* para amostras independentes com confiança de 95%. A medida F_1 foi utilizada como variável dependente alvo da análise. Destaca-se que a medida F_1 foi usada por representar a eficácia global do modelo, uma vez que considera a *Precision* e o *Recall*.

Observou-se que a diferença entre estes dois algoritmos é estaticamente desprezível ($p = 0,2480$), ou seja, os algoritmos são estatisticamente equivalentes ($p > 0,05$). Este resultado confirma a efetividade da seleção de *features*, tendo vista que um modelo simples, como o NB, conseguiu alcançar a eficácia da RF. Este resultado ratifica a proposta geral deste trabalho por dois motivos principais: primeiro porque o treinamento do NB é considerado de baixo custo; segundo porque este desempenho foi alcançado utilizando um conjunto reduzido de *features*, o que o torna ainda mais eficiente. Além disso, considerando um conjunto reduzido de *features*, tem-se menor custo computacional tanto para o treinamento de novos modelos, quanto para a execução prática do método sobre repositórios de código em larga escala.

5. CONCLUSÃO

Neste trabalho foi proposta uma abordagem de detecção de *bad smell* considerando a seleção explícita de *features*. Os experimentos realizados evidenciaram que a seleção de *features* aumentou estatisticamente a eficácia dos modelos, especialmente considerando seu poder de generalização avaliado com validação cruzada. Além disso, para alguns algoritmos, os modelos com seleção apresentaram equivalência estatística no conjunto de teste, mesmo utilizando um conjunto reduzido de *features*. Este

comportamento é altamente positivo, tendo em vista que os modelos conseguiram ser eficazes, mesmo utilizando menos *features*, o que contribui para a redução das demandas computacionais e aumento da eficiência do modelo. Destaca-se que modelos considerados simples, como o NB, após a seleção, tornaram-se estatisticamente equivalente a modelos mais complexos, como a RF.

Como trabalhos futuros pode-se realizar novos experimentos utilizando outras técnicas de seleção, considerando outros algoritmos de aprendizado de máquina e processos de otimização de modelos. Estes experimentos podem ser expandidos com outras bases de dados, especialmente considerando múltiplos tipos de *bad smell*, representando cenários ainda mais desafiadores do mundo real.

REFERENCES

Al-Shaaby, A., Aljamaan, H., and Alshayeb, M. Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review. *Arabian Journal for Science and Engineering* 45 (4): 2341–2369, apr, 2020.

Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., and Marino, A. Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering* 21 (3): 1143–1191, jun, 2016.

Azeem, M. I., Palomba, F., Shi, L., and Wang, Q. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Inf. Softw. Technol.* vol. 108, pp. 115–138, 2019.

Booch, G., Maksimchuk, R. A., Engel, M. W., Bobbi J. Young, J. C., and Houston, K. A. *Object-oriented analysis and design with applications*. The Addison-Wesley object technology series. Addison-Wesley, 2007.

Brownlee, J. Machine learning mastery with python. *Machine Learning Mastery Pty Ltd*, 2016.

Cruz, D., Santana, A., and Figueiredo, E. Detecting bad smells with machine learning algorithms: an empirical study. In *TechDebt '20: International Conference on Technical Debt, Seoul, Republic of Korea, June 28-30, 2020*, C. Izurieta, M. Galster, and M. Felderer (Eds.). ACM, pp. 31–40, 2020.

Cunningham, W. The wycash portfolio management system. In *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum)*. OOPSLA '92. Association for Computing Machinery, New York, NY, USA, pp. 29–30, 1992.

Danphitsanuphan, P. and Suwantada, T. Code smell detecting tool and code smell-structure bug relationship. In *2012 Spring Congress on Engineering and Technology*. IEEE, pp. 1–5, 2012.

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

Guggulothu, T. and Moiz, S. A. Code smell detection using multi-label classification approach. *Software Quality Journal* 28 (3): 1063–1086, sep, 2020.

Han, J., Kamber, M., and Pei, J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

Kreimer, J. Adaptive detection of design flaws. *Electronic Notes in Theoretical Computer Science* 141 (4): 117 – 136, 2005. Proceedings of the Fifth Workshop on Language Descriptions, Tools, and Applications (LDTA 2005).

Kruchten, P., Nord, R. L., and Ozkaya, I. Technical debt: From metaphor to theory and practice. *IEEE Softw.* 29 (6): 18–21, Nov., 2012.

Lehman, M. M. Programs, life cycles, and laws of software evolution. *Proc. of the IEEE* 68 (9): 1060–1076, 1980.

Liu, H., Guo, X., and Shao, W. Monitor-based instant software refactoring. *IEEE TSE* 39 (8): 1112–1126, 2013.

Martin Fowler, K. B. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

Moha, N., Gueheneuc, Y.-G., Duchien, L., and Le Meur, A.-F. Decor: A method for the specification and detection of code and design smells. *IEEE Trans. Softw. Eng.* 36 (1): 20–36, Jan., 2010.

Mori, A., Figueiredo, E., and Cirilo, E. Towards the definition of domain-specific thresholds. In *Anais do XIII Simpósio Brasileiro de Sistemas de Informação*. SBC, Porto Alegre, RS, Brasil, pp. 404–411, 2017.

Palomba, F., Bavota, G., Penta, M., Oliveto, R., Poshyvanyk, D., and Lucia, A. D. Mining version histories for detecting code smells. *IEEE TSE* 41 (05): 462–489, may, 2015.

Roy, R., Stark, R., Tracht, K., Takata, S., and Mori, M. Continuous maintenance and the future – foundations and technological challenges. *CIRP Annals* 65 (2): 667 – 688, 2016.

Sjøberg, D. I. K., Yamashita, A., Anda, B. C. D., Mockus, A., and Dybå, T. Quantifying the effect of code smells on maintenance effort. *IEEE TSE* 39 (8): 1144–1156, 2013.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. The qualitas corpus: A curated collection of java code for empirical studies. In *APSEC*. pp. 336–345, 2010.

Yamashita, A. and Moonen, L. Do code smells reflect important maintainability aspects? In *2012 28th IEEE international conference on software maintenance (ICSM)*. IEEE, pp. 306–315, 2012.