UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Revista Científica

http://revistas.udistrital.edu.co/ojs/index.php/revcie/index

Artículo de revisión

Analysis of Code Smells Detection Using Machine Learning and Deep Learning Approaches

Análisis de la detección de *code smells* mediante enfoques de aprendizaje automático y aprendizaje profundo

Análise de detecção de *code smells* usando abordagens de aprendizado de máquina e aprendizado profundo

Enrique Alejandro Chim Mex[1]
Antonio Armando Aguileta Güemez[2]
Raúl Antonio Aguilar Vera[3]

Recibido: 19 de diciembre de 2025 **Aceptado:** 23 de febrero de 2026

Para citar este artículo: Chim, E. A., Aguileta, A. A. y Aguilar, R. A. (2026). Analysis of Code Smells Detection Using Machine Learning and Deep Learning Approaches. *Revista Científica*, *53*(1), 1-19. https://doi.org/10.14483/23448350.24617

Abstract

Detecting code smells (CS) is important for preventing future problems in software development. It also helps to improve software quality and save time on maintenance. This study contributes with a systematic experiment that integrates data leakage control, rigorous preprocessing, and the comparison of machine learning (ML) and deep learning (DL) models, contributing with a replicable methodology for CS detection. To this effect, an experiment was designed which focuses on CS analysis using artificial intelligence approaches. ML and DL models were applied to the dataset based on method-level software metrics. The methodological process included comprehensive processing, which addressed variable cleaning and normalization, transformations, and feature reduction. In addition, the issue of data leakage was controlled to ensure the validity of the results. Multiple ML models (random forest, support vector machines, decision trees, k-Nearest Neighbors, Naive Bayes, and logistic regression) and an DL model based on a multilayer perceptron (MLP) were trained and evaluated. The results showed remarkable performance in most models, achieving accuracy between 94 and 98% after cross-validation with ten folds. However, the MLP stood out with an accuracy close to 99%, positioning it as the best-performing classifier for CS detection.

Keywords: code smells, machine learning, deep learning, software metrics, software engineering

1. Universidad Autónoma De Yucatán, Mérida, México. Contacto: a20216391@alumnos.uady.mx
2. Universidad Autónoma De Yucatán, Mérida, México. Contacto: aaguilet@correo.uady.mx
3. Universidad Autónoma De Yucatán, Mérida, México. Contacto: avera@correo.uady.mx

Resumen

Detectar *code smells* (CS) es fundamental para prevenir problemas futuros en el desarrollo de *software*. Además, contribuye a mejorar la calidad del *software* y a ahorrar tiempo en el mantenimiento. Este estudio aporta un experimento sistemático que integra el control de fugas de datos, un preprocesamiento riguroso y la comparación de modelos de aprendizaje automático (AA) y aprendizaje profundo (AP), lo que proporciona una metodología replicable para la detección de OC. Para ello, se diseñó un experimento centrado en el análisis de OC mediante enfoques de inteligencia artificial. Se aplicaron modelos de AA y AP al conjunto de datos en función de métricas de *software* a nivel de método. El proceso metodológico incluyó un procesamiento exhaustivo que abarcó la limpieza y normalización de variables, transformaciones y reducción de características. Asimismo, se controló el problema de las fugas de datos para garantizar la validez de los resultados. Se entrenaron y evaluaron múltiples modelos de AA (bosque aleatorio, máquinas de vectores de soporte, árboles de decisión, k-vecinos más cercanos, Naive Bayes y regresión logística) y un modelo AP basado en un perceptrón multicapa (MLP). Los resultados mostraron un rendimiento notable en la mayoría de los modelos, alcanzando una precisión de entre el 94 y el 98 % tras la validación cruzada con diez pliegues. Sin embargo, el MLP destacó con una precisión cercana al 99 %, posicionándose como el clasificador con mejor rendimiento para la detección de OC.

Palabras clave: olores de código, aprendizaje automático, aprendizaje profundo, métricas de software, ingeniería de *software*

Resumo

A deteção de *code smells* (CS) é importante para prevenir problemas futuros no desenvolvimento de *software*. Ajuda também a melhorar a qualidade do *software* e a poupar tempo na manutenção. Este estudo contribui com uma experiência sistemática que integra o controlo de fuga de dados, o pré-processamento rigoroso e a comparação de modelos de aprendizagem automática (ML) e aprendizagem profunda (DL), contribuindo com uma metodologia replicável para a deteção de CS. Para tal, foi desenhada uma experiência com foco na análise de CS utilizando abordagens de inteligência artificial. Os modelos de ML e DL foram aplicados ao conjunto de dados com base em métricas de *software* ao nível do método. O processo metodológico incluiu o processamento abrangente, que abordou a limpeza e normalização de variáveis, transformações e redução de recursos. Além disso, a questão da fuga de dados foi controlada para garantir a validade dos resultados. Múltiplos modelos de ML (floresta aleatória, máquinas de vetores de suporte, árvores de decisão, k-vizinhos mais próximos, Naive Bayes e regressão logística) e um modelo de DL baseado em um perceptron multicamada (MLP) foram treinados e avaliados. Os resultados mostraram um desempenho notável na maioria dos modelos, atingindo uma precisão entre 94 e 98% após validação cruzada com dez folds. No entanto, a MLP destacou-se com uma precisão próxima dos 99%, posicionando-se como o classificador com melhor desempenho para a deteção de code smells.

Palavras-chaves: code smells, machine learning, deep learning, métricas de software, engenharia de *software*

INTRODUCTION

Code smells (CS) are code structures linked to poor decisions made during design and implementation (Fowler, 2018). Detecting these bad decisions has been a challenge in the field of software engineering. Although there are different studies (Di Nucci et al., 2018; Arcelli et al., 2016; Guggulothu & Moiz, 2012) and tools (Fontana et al., 2012) in the literature for detection, there are still some challenges in evaluating them. For example, different detection techniques yield different results because developers subjectively interpret the coded CS, and there is little agreement among CS detection tools (Fontana et al. 2012; Paiva et al., 2017; Kiyak *et al*., 2017).

Today, one factor that influences an organization's competitive capacity is the quality of its software. However, the existence of CS remains an obstacle to system evolution and maintenance. Among the most prominent CS—*i.e.*, those to which this study devotes the most attention—are Feature Envy and Method.

Machine learning (ML) (Forero-Corba & Negre, 2024) is a branch of artificial intelligence (AI) that has seen exponential growth in recent years. Currently, manual CS detection can be a time-consuming task. Therefore, several studies have applied ML techniques to identify these problems, with the aim of making the identification task more flexible.

On the other hand, deep learning (DL) (García, 2021) refers to ML based on models composed of several artificial neural capabilities. The simplest model, called *Perceptron*, was the first artificial neural network (ANN), created by Rosenblatt in 1958. Its structure consists of an input layer composed of several linear neurons that transmit information and process incoming signals, allowing data to be classified into two groups based on the output results or patterns they generate.

This study addresses CS detection through the application of ML and DL techniques, using the dataset reported by Pereira (Dos Reis et al., 2022). This dataset was constructed from software projects belonging to the Qualitas Corpus repository (Tempero, 2011), which brings together a large collection of real projects from different domains. To this effect, an approach based on six classifiers for ML and a multi-layer perceptron (MLP) for DL is used, complemented by data preprocessing techniques, with the aim of evaluating their effectiveness in classification. This work also focuses on two CS at the method level: Feature Envy and Long Method. The first refers to a method that uses more data from other classes than from its own class, and the second refers to a method that has too many lines and requires too many parameters. Therefore, it tends to be complex and difficult to maintain. The selection of these CS was based on the fact that they are the most frequently occurring in software projects, which makes them representative and important cases for analysis.

The main contributions of this study are: (i) the application of classification techniques to detect CS at the method level; (ii) the application of rigorous preprocessing techniques, including feature selection and normalization, in order to obtain more realistic results; (iii) the provision of a replicable study framework that can serve as a basis for future studies on CS detection using machine learning.

THEORETICAL FRAMEWORK AND RELATED STUDIES

The algorithms were selected based on their simplicity and widespread use reported in the literature (Nguyen et al., 2022; Cruz & Figueiredo, 2020) for CS detection.

Machine learning

ML techniques can be divided into three groups: supervised, unsupervised, and semi-supervised. All of them use features as input that can be categorical, continuous, or binary depending on their nature (Kotsiantis, 2007; Caram *et al.*, 2019). When instances with known labels are provided, this is known as *supervised learning*; otherwise, when these instances do not have labels, it is known as *unsupervised learning*.

The classifiers used in this study are presented below:

- *Random forest (RF)*. This is a tree-based ensemble technique that uses a set of tree models constructed with a subset of features, taking the average of said trees to make predictions (Caram *et al.*, 2019).
- *K-nearest neighbors (kNN)*. kNN is a learning algorithm that classifies elements based on their position and distance on a hyperplane (Caram *et al.*, 2019).
- *Naive Bayes (NB)*. This approach uses Bayesian statistics to determine the probability between an unobserved node and a chain of observed child nodes. It assumes an independent relationship between the child nodes and their parent node (Caram *et al.*, 2019).
- Logistic regression (LR). This method fits a sigmoid function in a linear regression model for binary classification. The function determines the classification probability of each instance and, based on a probability threshold, classifies it as positive or negative (Caram *et al.*, 2019).
- Decision tree (DT). This technique classifies instances by ordering them according to feature values and dividing them into branches. Each branch represents the value thresholds that the contained nodes can assume, and each node represents a feature (Caram *et al.*, 2019).
- Support vector machine (SVM). This method learns the decision surface of two different classes from the input points. As a single-class classifier, the description given by the support vector data can form a decision boundary around the learning data domain with little or no knowledge of the data outside said boundary (Betancourt, 2005).

Deep learning

Artificial neurons are inspired by the functioning of a biological neuron, *i.e.*, a nerve cell with a body or soma that is surrounded by a series of branches or dendrites. There is a filament known as an *axon* that extends and branches into axonal terminals. Each neuron receives electrical signals through the dendrites. The signals received are processed or integrated in the body.

When the resulting electrical signal exceeds a threshold potential or voltage, an action potential is triggered through the axon. The action potential reaches the axonal terminals and, through a chemical mechanism involving substances called *neurotransmitters*, connections or synapses are established with the dendrites of another neuron, thus generating an excitatory or inhibitory postsynaptic potential whose magnitude depends on the intensity or synaptic weight with which the two neurons are connected. This process of signal weighting, summation, and activation is repeated in the other neurons, and it is recovered to model the functioning of an artificial neuron (García, 2021).

MULTILAYER PERCEPTRON

Although network configurations have many different aspects, they all share one thing in common: the arrangement of neurons in layers or levels. There are three types of layers:

- *Input:* this is the layer that directly receives information from external sources on the network.
- *Hidden:* these are internal to the network and have no direct contact with the outside world. A network may not have hidden layers. They can be interconnected in different ways, which, together with their number, determines the topology of the neural network.
- *Output:* this layer transfers information from the network to the outside.

A group of cascading layers forms the basis of multilayer networks. The number of hidden and output layers in a network determines the total number of layers. In this sense, a network composed of an input layer and an output layer is called a *single-layer network*. This is because the neurons in the input layer transfer information without processing it. In other words, they are linear, and the activation function corresponds to the following identity:

$$a_j = f\left(v_j\right) = v_i$$

In feedforward networks, each neuron in a layer descends from all neurons in the next layer, and no neuron receives signals from another one in its own layer or the sublayers behind it. Multilayer networks have been shown to have more features and attributes than single-layer networks.

RELATED WORKS

In recent years, several studies have been proposed which focus on detecting CS via ML, achieving remarkable performance metrics in classification tasks. Most studies have focused on analyzing CS at the method and class level.

Fontana et al. (2012) conducted an experiment that applied ML algorithms to known CS. They experimented with 16 different algorithms on four CSs. As a result, they achieved a high performance on the cross-validation dataset, with the best results obtained by J48 and RF. In addition, the application of ML to CS detection provided high accuracy (greater than 96%).

In 2020, Guggulothu and Moiz proposed a multi-label classification (MLC) method to detect whether a given code element was affected by multiple CS. They found that there is a positive correlation between two given CS. In terms of classification, the two MLC methods considered the correlation between CS and improved performance, achieving a precision of 95% across ten cross-validation iterations.

Di Nucii et al. (2018) focused on the problem regarding the distribution of metrics for 'smelly' instances in the dataset, which is significantly different from the non-smelly instances in Fontana et al. (2012) These authors set out to investigate this issue. The findings showed that the high performance achieved in the latter was, in fact, due to the specific dataset used rather than the actual capabilities of ML techniques for CS detection.

Table 1 presents a comparison of the differences between this study and the studies reported in previous research.

Table 1. *Comparison of studies related to the detection of code smells*

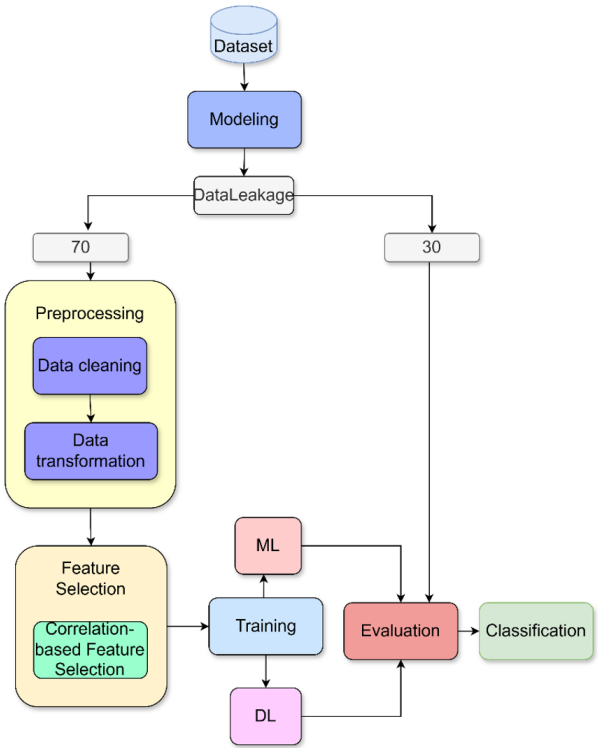
Studies	Method level	Data- leakage	Transformation	Feature selection
Detecting code smells using machine learning techniques: Are we there yet? (Di Nucci *et al.*, 2018)	Yes	No	No	No
Comparing and experimenting Machine learning techniques for code smells detection (Fontana *et al.*, 2012)	Yes	No	No	No
Code smell detection using multi-label classification approach (Guggulothu & Moiz, 2020)	Yes	No	No	No
Our study	Yes	Yes	Yes	Yes

METHODOLOGY

In this study, we designed a methodology to analyze CS detection from the perspective of a binary classification problem, where the target variable takes two mutually exclusive values: the presence of issues related to Feature Envy and Long Method.

The proposed methodology is an original approach that was developed with the aim of preventing data leakage and thereby ensuring the accuracy of the results provided by algorithms.

Figure 1. *Stages of the methodology*



DATASET

The dataset (Dos Reis et al., 2022) used for this study features 82 software metrics at different levels of abstraction, with the method level being one of the most important. The dataset consists of 420 instances labeled according to the type of CS within which they are framed.

Tables 2 and 3 show the software metrics at the method and class levels, respectively.

Table 2. *Class-level software metrics*

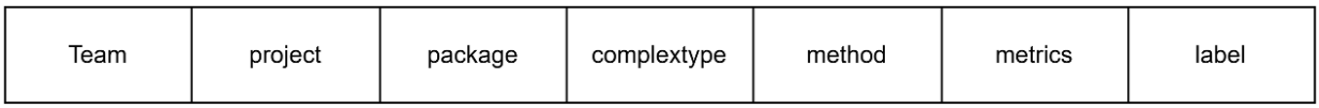
Metrics	Metrics	Metrics
NOII_TYPE	NMO_TYPE	NOMNAMM_TYPE
NOAM_TYPE	ATFD_TYPE	NOA_TYPE
NOCS_TYPE	FANOUT_TYPE	NIM_TYPE
NOM_TYPE	LOC_TYPE	DIT_TYPE
LOCNAMM_TYPE	CFNAMM_TYPE	TCC_TYPE
NOPA_TYPE	CBO_TYPE	RFC_TYPE
NOC_TYPE	WMC_TYPE	LCOM5_TYPE
WOC_TYPE	WMCNAMM_TYPE	AMW_TYPE
AMWNAMM_TYPE	ISSTATIC_TYPE	NUMBER_PRIVATE VISIBILITY ATTRIBUTES
NUMBER_PROTECTED VISIBILITY ATTRIBUTES	NUM_FINAL ATTRIBUTES	NUM_STATIC ATTRIBUTES

Table 3. *Method-level software metrics*

Metrics	Metrics	Metrics
NOP_METHOD	MAXNESTING_METHOD	LAA_METHOD
CC_METHOD	LOC_METHOD	FANOUT_METHOD
ATFD_METHOD	CYCLO_METHOD	CFNAMM_METHOD
FDP_METHOD	NMCS_METHOD	ATLD_METHOD
MEMCL_METHOD	NOLV_METHOD	CLNAMM_METHOD
MAMCL_METHOD	NOAV_METHOD	CINT_METHOD
CDISP_METHOD	ISSTATIC_METHOD	CM_METHOD

Furthermore, one of the components of the database is that it was analyzed by human teams. This made it possible to avoid individual biases and maintain consistency in CS classification. Figure 2 shows the structures of the databases.

Figure 2. *Database structure*



MODELING

The datasets were arranged at the Feature Envy and Long Method method levels, both with the same software metrics and values, but with different CS. One of the advantages of this approach is that it allows identifying the symptoms of two CS, which may affect software maintenance in the future.

The approach was formulated as a classification of two categories by level of abstraction.

Figure 3. *Dataset unification process*

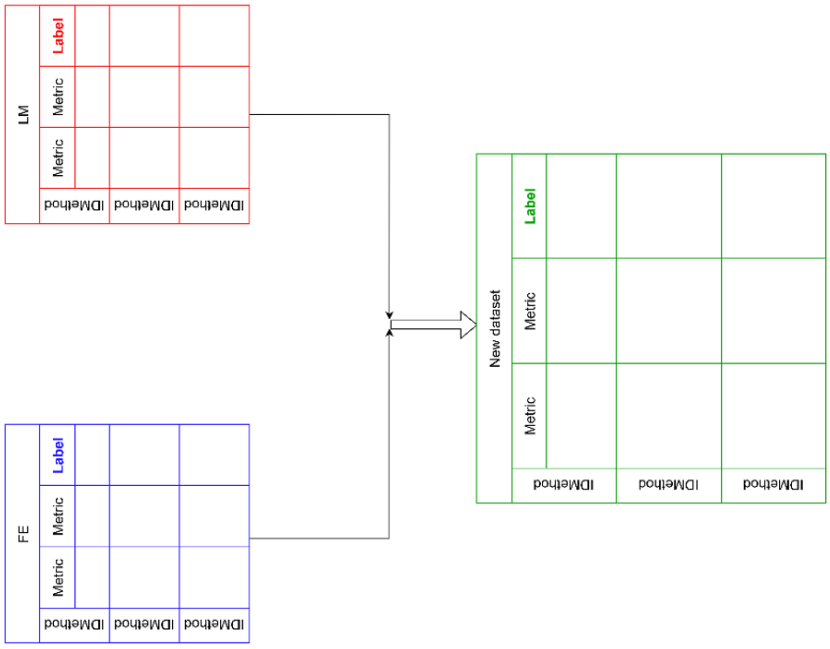


Figure 3 shows the process of integrating the datasets used in this study. It shows the merging of software metrics at the method level to form a unified database. Both databases are merged using a common identifier—in this case, the IDMethod. During this process, each instance retains both its software metrics and the corresponding label. The result is a new database with two labels (Feature ENVY and Long Method), which integrates the characteristics of both levels, thus allowing the algorithms to learn different patterns for each CS.

Data leakage

Data leakage during ML processing is a problem where unintended external information distorts the training process, causingan artificially high metric performance (Bouke et al., 2024). To avoid this problem, the dataset was divided at the team level, meaning that all instances belonging to the same project were kept together, either in the training set or in the test set, but never in both. This ensures that the algorithms learn only from independent information and are not influenced by internal correlations within the same team. In total, 65 teams were considered for the training set (*df_train*) and 28 teams for the test set (*df_test*).

PREPROCESSING

Data preparation is one of the key phases carried out during the data analysis process. For this phase, raw data is converted into a format that computers and machine learning algorithms can understand and evaluate. For this stage, various techniques were implemented, with the aim of improving data quality and optimizing algorithm performance. The main techniques applied are described below:

Data cleaning

In this stage, the dataset was analyzed, and inconsistent records or those with missing values were identified and eliminated. First, the database was checked for null values and inconsistent symbols (?). To this effect, the following checks were applied to the original DataFrame (df_train and df_test), as shown in Code 1.

Code 1. *Verification of null values and question marks in the dataset*

```
nulos = df.isnull().sum()
nulos = nulos[nulos > 0]


interrogaciones = (df == '?').sum()
interrogaciones = interrogaciones[interrogaciones > 0]
```

Subsequently, the results were evaluated to verify that there were no inconsistencies in the columns. The analysis yielded no null values or question marks in the dataset.

Outliers. According to the study by Falahi et al. (2023), which was based on previous studies, detecting outliers or anomalies in the data consists of identifying observations that deviate significantly from the expected general behavior. Any type of outlier detection depends on two basic assumptions. Figure 4 illustrates how outliers behave. The reader should recall:

- Outliers are rare.
- Outliers are very different from normal data.

Figure 4. *Boxplot with outliers*

To identify variables with outliers, the interquartile range (IQR) criterion was applied. In this procedure, for each variable X, the first and third quartiles Q_1 and Q_3 were calculated, obtaining the interquartile range as follows:

$$IQR = Q_3 - Q_1$$

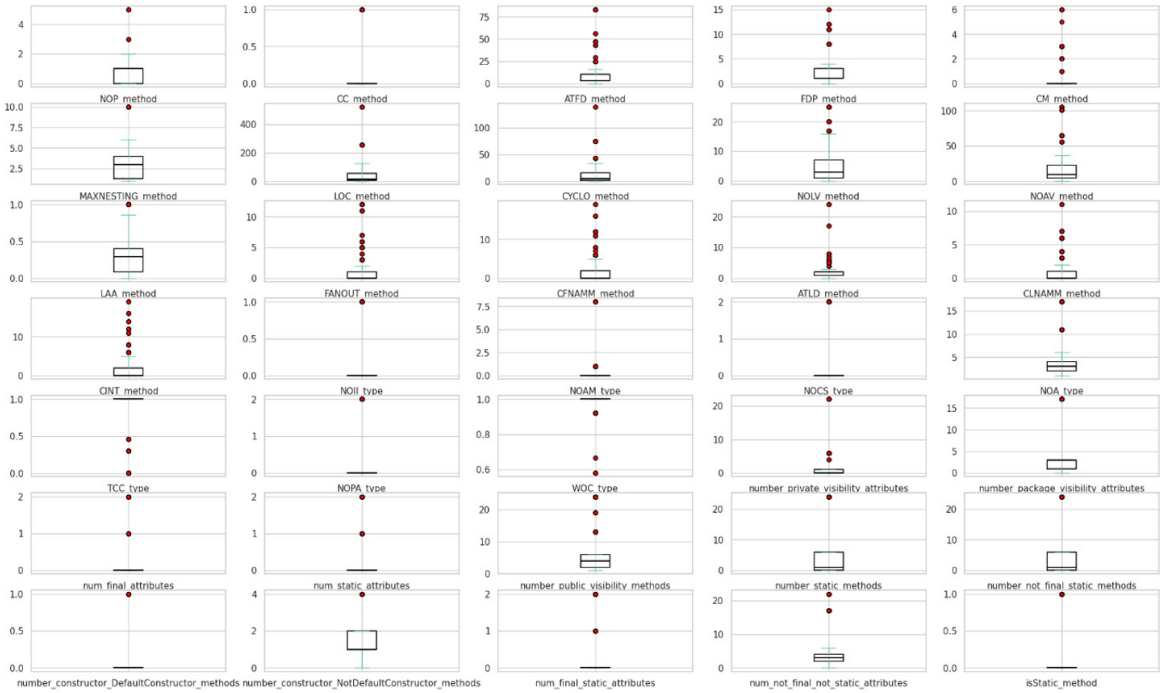
The lower and upper limits were defined according to Tukey's criterion (Hernández Vargas, 2015):

$$L_{inf} = Q_1 - 1.5 * IQR, \qquad L_{sup} = Q_3 + 1.5 * IQR$$

Any observation outside these limits was considered an outlier.

In order to identify the distribution of the variables in the training set (*df_train*), a box-and-whisker plot was constructed for each of the features, which allowed visualizing the behavior and detect outliers. This behavior is shown in Figure 5.

Figure 5. *Distribution of attributes in the dataset with the presence of outliers*



The results of the graph reveal that some variables have markedly asymmetrical distributions skewed to the left, with a high concentration of low values. These characteristics suggest a possible influence of these values. The graph shows that many of the variables have markedly skewed and non-normal distributions, with pronounced peaks and extended tails. Metrics with strong positive asymmetry can be observed, as well as metrics with cumulative values around zero, which may affect the performance of some models. Table 4 shows the variables with outliers in the dataset.

Table 4. *Variables with outliers according to the IQR criterion*

Metrics	**Metrics**	**Metrics**
NOP_method	NOCS_type	NOLV_method
ATFD_method	num. final attributes	LAA_method
MAXNESTING method	num. final static attributes	CFNAMM_method
CYCLO_method	num. public visibility methods	CLNAMM_method
NOAV_method	num. package visibility attributes	TCC_type
FANOUT_method	num. not final static methods	NOAM_type

ATLD_method	num. constructor NotDefaultConstructor methods	WOC_type
CINT_method	CC_method	NOA_type
CINT_method	CM_method	Num. static attributes
NOPA_ype	LOC_method	num. not final not static attributes
num. private visibility attributes	num. static methods	num. constructor DefaultConstructor methods

Data transformation for outliers

Subsequently, the Yeo-Johnson (YJ) transformation was applied to the variables, in order to improve their distribution and mitigate the influence of outliers. As described in the previous section on outliers, the dataset contains a considerable number of extreme observations and a skewed distribution. In this context, a power transformation was chosen to reduce skewness and stabilize variance without removing relevant information.

According to Raymaekers and Rousseeuw (2024), to transform a positive variable and provide it with a more normal distribution, a power transformation is typically used. The most widely used function is the Box-Cox (BC) power transformation (see 1).

$$g_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{si } \lambda \neq 0 \\ \log(x) & \text{si } \lambda = 0 \end{cases} \tag{1}$$

where x represents the observed characteristic, which is transformed into g(x) using a parameter. One limitation of the BC family of transformations is that they can only be applied to positive data. To address this issue, Yeo and Johnson (2000) proposed an alternative family of transformations that can handle positive, negative, and zero data. This transformation is given by 2.

$$g_\lambda(x) = \begin{cases} \frac{(1+x)^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \text{ and } x \geq 0 \\ \log(1+x), & \text{if } \lambda = 0 \text{ and } x \geq 0 \\ -\frac{(1-x)^{2-\lambda} - 1}{2-\lambda}, & \text{if } \lambda \neq 2 \text{ and } x < 0 \\ -\log(1-x), & \text{if } \lambda = 2 \text{ and } x < 0 \end{cases} \tag{2}$$

Parameter estimation for the YJ and BC transformations is typically performed using maximum likelihood, under the assumption that the transformed variable follows a normal distribution. Considering that the dataset is influenced by outliers, we decided to apply the YJ transformation to reduce asymmetry, stabilize variance, and mitigate these outliers. The transformation was implemented using the PowerTransformer class via *method='yeo-johnson'*. This model was adjusted only to the training set (*df_train*), using *fit_transform* to avoid information leakage, and then applied to the test set (*df_test*) with *transform*, so that the test set was not mixed and modified with respect to the training. This resulted in two new variables: *df_train_yeo* and *df_test_yeo*).

Standardization

As a result of a standardization process (Z-score), the features are rescaled to ensure that the mean and standard deviation are 0 and 1. This technique is useful for optimization algorithms. It is also used for algorithms that employ distance measures, such as kNN. On the other hand, tree-based algorithms are insensitive to scale; the tree divides a node into a feature that increases the homogeneity of the node. This division into a feature is not influenced by other features. Therefore, there is virtually no effect of the remaining features on the division. This is what makes them invariant to feature scale (Vinay Sachin, 2021).

Since several of the models evaluated are sensitive to feature magnitude, standardization was applied to ensure numerical stability and balanced contribution of all variables during training.

The implementation was carried out using the StandardScaler class, adjusting it only to the training set (*df_train_yeo*) and using *fit_transform* to prevent information leakage. It was then applied to the test set *df_test_yeo* using *transform*. This allowed all software metrics to contribute in a balanced way to the training of the models.

FEATURE SELECTION

Feature selection consists of identifying and discarding irrelevant or redundant variables from the dataset, using evaluation criteria to determine their usefulness for the model. There are three types of feature selection: filtering, wrapping, and embedding. Filter methods use the general properties of the training data to perform selection as a step-by-step process that is independent of an induction algorithm (Noroozi & Erfannia, 2023). In this study, the filter approach was used, specifically correlation-based feature selection (CFS) method. This approach was chosen because it does not require a high computational effort compared to other approaches where features are combined with the algorithm. Instead, this approach allows selecting the most relevant variables based solely on the statistical properties of the dataset.

The CFS method, initially proposed by Hall (2000), is classified as a subset selection approach since it evaluates groups of features rather than analyzing them individually. This method uses a heuristic strategy to perform a partial search of the possible combination space. As a filtering technique, CFS does not depend on the performance of a specific classifier; it is based on Ghiselli's testing theory, according to which an optimal subset must include features that are highly correlated with the target variable but weakly correlated with each other. CFS is expressed as follows (Palma et al., 2018):

$$|M_S| = \frac{K * \bar{r}_{cf}}{\sqrt{K + K(K-1) * \bar{r}_{ff}}} \tag{3}$$

where:

- M_S : merit of the subset of features S
- K : number of features in the subset
- $\bar{r}_{cf}$: average correlation between each feature and the class
- $\bar{r}_{ff}$: average correlation between the features themselves

After applying CFS, a set of variables was obtained that have a low correlation and provide information for algorithms. The selected variables are presented in Table 5.

Table 5. *Metrics selected after CFS filtering*

Metrics	**Metrics**	**Metrics**
NOP_method	CBO_type	CLNAMM_method
ATFD_method	AMW_type	NOII_type
MAXNESTING_method	NONMAMM_package	NOM_type
NOLV_method	num. public visibility methods	FANOUT_type
FANOUT_method	num. constructor DefaultConstructor methods	LOC_type
ATLD_method	num. constructor NotDefaultConstructor methods	NOPA_type
CDISP_method	CC_method	LCOM5_type
NOAM_type	FDP_method	NOCS_package
ATFD_type	LOC_method	num. package visibility attributes
NOA_type	LAA_method	num. private visibility methods
TCC_type	CFNAMM_method	

TRAINING

For both approaches mentioned in our theoretical framework, a supervised training process was carried out, using RF, DT, KNN, SVM, NB, and RL for ML and MLP for DL. To ensure the robustness of the results, cross-validation was applied, which allowed evaluating the performance of the models on different partitions and reduce the probability of overfitting.

CLASSIFICATION

The models were used to classify new instances of the dataset by assigning them to the appropriate category. Each instance was classified into one of two categories: Long Method or Feature Envy. Classification performance was evaluated using metrics that are widely used in the literature (Dos Reis *et al.*, 2022): accuracy, precision, recall, and the F1-score, both globally and for each class.

EXPERIMENTATION

Model configuration

To evaluate the performance of ML and DL models for CS detection, a set of experiments was conducted using the already balanced and processed dataset. Table 6 describes the main configuration parameters used by the models.

Table 6. *Configuration of the algorithms used in the experiment*

Algorithm	Implementation	Key parameters
DT	`DecisionTreeClassifier`	`random_state=42`
RF	`RandomForestClassifier`	`random_state=42`
SVM	`SVC`	`kernel='rbf', gamma='scale'`
LR	`LogisticRegression`	`max_iter=1000`
KNN	`KNeighborsClassifier`	`n_neighbors=5`
NB	`GaussianNB`	(por defecto)

Parameter selection aims to ensure the reproducibility of the results. For decision tree-based algorithms (RF and DT), the parameter *random_state=42* was set to maintain the consistency of randomness in the data partitioning process and for tree generation. For the SVM, *kernel='rbf'* was used to capture information on nonlinear relationships within the feature space. In addition, *gamma='scale'* was set so that the parameter value would adjust according to the variance. For LR, the number of iterations was increased to *max_iter=1000* to ensure convergence, since the dataset has multiple variables. For KNN, *n_neighbors=5* was selected, which is a standard value that offers a balance between bias, avoiding both overfitting and underfitting. Finally, NB was set with the default parameters, given that its probabilistic assumptions are suitable for comparison with the other models.

Below is the configuration of the MLP model used in the experiments:

- Input
  - The number of predictor variables (columns in the dataset after selection transformation)
- Hidden layers
  - First hidden layer
    - » Size = 4 * number of predictor variables limited to a maximum of 256
  - Second hidden layer
    - » Size = 4 * number of predictor variables limited to a maximum of 128
- Activation function
  - ReLU activation function
- Output layer
  - (1 outputs 1 neuron + sigmoid per output

CROSS-VALIDATION

This procedure consisted of dividing the training dataset into ten folds, using one for testing and the remaining nine for training. It was applied to the training set (*X_train_scaled, y_train*), which had previously been normalized using the StandardScaler technique. The main performance metrics were calculated using the weighted average (*average='weighted'*) to account for possible class imbalance. The predictions generated for each iteration were compared against the actual labels (*y_train*). Finally, the results of the metrics were documented, allowing the best algorithm to be identified and compared. The predictions generated for each iteration were also contrasted with the actual labels. As a result, the final metrics were documented, allowing the performance of the algorithms to be compared.

RESULTS

Table 7. *Model classification report*

Model	Class	Precision	Recall	F1-Score	Accuracy
RF	1	1.000	1.000	1.000	1.000
	2	1.000	1.000	1.000	
KNN	1	0.943	1.000	0.971	0.972
	2	1.000	0.946	0.972	
SVM	1	1.000	0.980	0.990	0.991
	2	0.982	1.000	0.991	
LR	1	1.000	0.980	0.990	0.991
	2	0.982	1.000	0.991	
NB	1	0.961	0.980	0.970	0.972
	2	0.982	0.964	0.973	
DT	1	1.000	1.000	1.000	1.000
	2	1.000	1.000	1.000	

The results in Table 7, obtained after applying preprocessing, YJ transformation, and feature selection, show remarkable performance at the method level, with metrics above 90 in most of the models evaluated. The RF and DT models achieved perfect performance, with recall, precision, and F1-Score metrics of 1.000 in both classes. On the other hand, SVM and RL performed excellently, with an accuracy of 0.991 and balanced metrics in both classes. This suggests that the dataset may exhibit a high degree of separability in the feature space, as the linear classifiers achieved a performance close to 1.000.

The KNN and NB models showed a slightly lower performance, with an accuracy of 0.972. In the case of KNN, the precision in class 1 (0.943) indicates that there are false positives. Since the classifier is based on similarity in the feature space, methods with structural metric values close to the decision boundary may be misclassified. NB, on the other hand, showed limitations associated with its assumption of unconditional independence, which explains the slight reduction in metrics. Table 8 shows the performance of the classification models using 10-fold cross-validation.

Table 8. *Classification report of models with 10-fold cross-validation*

Model	Precision	Recall	F1-Score	Accuracy
RF	0.982	0.982	0.982	0.982
KNN	0.952	0.951	0.951	0.951
SVM	0.989	0.980	0.989	0.989
LR	0.986	0.986	0.986	0.986
NB	0.944	0.944	0.944	0.944
DT	0.979	0.979	0.979	0.979

In general terms, all algorithms achieved high metrics, with precision, recall, F1-score, and accuracy values above 0.94, which demonstrates the quality of the set of attributes for classification. The best-performing model was SVM, with the following values: precision = 0.989, recall = 0.980, F1-score = 0.989, and accuracy = 0.989. The models that performed poorly were KNN and NB, with accuracies of 0.951 and 0.944, respectively.

Table 9. *MLP results*

Class	Precision	Recall	F1-score	Support
1	1.00	0.98	0.99	50
2	0.98	1.00	0.99	56
Accuracy			0.99	106
Macro Avg	0.99	0.99	0.99	106
Weighted Avg	0.99	0.99	0.99	106

The results obtained when applying for the MLP show remarkable performance in the classification of both classes, as can be seen in Table 9. In terms of accuracy, class 1 achieved a value of 1.00, while class 2 achieved a value of 0.98, indicating that the predictions for class 1 were correct. In terms of recall, a value of 0.98 was observed. In class 2, a value of 1.00 was reached. The F1-score remained at 0.99 for both classes, confirming an almost perfect balance between precision and completeness. This performance is reflected in the overall metrics, with an accuracy of 0.99, suggesting that the model maintains a consistent behavior in the classification of the two classes.

DISCUSSION

The results of this research provide an overview of how CS detection is carried out using two different AI approaches. These findings can benefit software engineers. The databases used for this study are used in various detection studies with different approaches and results. Due to their structure based on method-level software metrics, the AI models performed excellently in our experiments.

For ML models, the databases yielded remarkable results. In general terms, most of the selected models achieved values above 98%, and some models even achieved outstanding metrics. However, this behavior can be attributed to a possible overfitting of the data. To mitigate this behavior, cross-validation was performed using ten folds, where values of 94-98% were achieved.

In the case of the DL model, the results showed competitive performance, albeit with characteristics different from those of traditional ML models. In particular, the MLP-type neural network achieved accuracy values close to 99%, demonstrating the potential of this approach with the data.

Overfitting is a potential danger, especially considering that certain tree-based classifiers have achieved perfect scores. To reduce this risk, after a preliminary evaluation using training and test splits, 10-fold cross-validation was implemented to ascertain whether the results were stable across different divisions of the data.

Compared to other studies focused on CS detection (Di Nucci et al., 2018; Guggulothu & Moiz, 2020; Fontana et al., 2012), our work paid special attention to the issue of data leakage, an aspect that research has overlooked. The aim was to ensure that the methodology more accurately reflected the conditions of application in real contexts. In addition, our preprocessing procedures were employed and demonstrated, an aspect that is often omitted in the literature. In this study, we considered a fundamental stage related to data quality, since it directly influences model performance. Thanks to this stage, the models performed excellently. In addition to its positive results, this distribution stands out among previous studies, where the absence of such a stage limits the validity of the results. For example, Guggulothu et al. (2020) reported favorable results, achieving 95-98% accuracy, while Di Nucci et al. (2018) achieved results ranging from 73-75%.

The results of the ML and DL models are deeply related the methodology used, which included the application of rigorous preprocessing techniques, a detailed analysis of data behavior, the handling of missing values, and the implementation of feature selection strategies to reduce data redundancy. Among the models used, MLP stood out for achieving the highest accuracy, close to 99%, which demonstrates its potential in the automatic detection of CS. Nevertheless, the ML models showed remarkable performance in cross-validation.

When conducting the experiments, method-level software metrics from specific corpora were used. This could limit the applicability of the results to other programming languages, domains, or real-world environments in which software is developed. Therefore, although the suggested approach demonstrated outstanding performance under the analyzed conditions, further validation in industrial and cross-project environments is necessary to corroborate its strength and ability to generalize.

CONCLUSIONS

This study proposed a methodology for detecting CS using ML and DL approaches. Various ML models were evaluated (RF, SVM, DT, NB, KNN, and RL), while a MLP-based methodology was proposed for DL.

Processing techniques were used to mitigate data defects, including the analysis and treatment of missing values, the reduction of redundancies through feature selection, and normalization. These techniques made it possible to obtain more accurate results and ensure the validity of the findings. Overall, the ML models achieved metrics between 94 and 98%, while MLP stood out by obtaining a value of 99%, thus demonstrating the potential of both approaches. These results offer practical insights for software engineers, showing that automatic CS detection can reduce the effort of manual reviews, thereby improving code quality and maintenance.

Future work should include validations on larger and more diverse datasets. Similarly, incorporating natural language processing perspectives can broaden the scope of detection beyond structural metrics and optimize its applicability in practical terms.

AUTHORSHIP CONTRIBUTIONS

Enrique Alejandro Chim Mex: Data Curation, Formal Analysis, Research, Methodology, Drafting, Visualization.
Antonio Armando Aguileta Güemez: Conceptualization, Supervision, Project Management, Methodology, Validation, Writing, Revision, and Editing.
Raul-Antonio Aguilar-Vera: Formal Analysis, Fundraising, Methodology, Validation, Revision, and Editing, Corresponding Author.

DECLARATION ON THE USE OF ARTIFICIAL INTELLIGENCE.

All content of the manuscript was reviewed by the authors. No artificial intelligence tools were used for the original manuscript content, data analysis, results, or conclusions.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of Universidad Autónoma de Yucatán for the institutional support provided for the development of this research.

REFERENCES

Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., & Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, *21*(3), 1143-1191. https://doi.org/10.1007/s10664-015-9378-4

Betancourt, G. A. (2005). Las máquinas de soporte vectorial (SVMs). *Scientia et technica*, *1*(27), 67-72. https://www.redalyc.org/pdf/849/84911698014.pdf

Bouke, M. A., Zaid, S. A., & Abdullah, A. (2024). *Implications of data leakage in machine learning preprocessing: A multi-domain investigation* [Preprint]. https:/doi.org/10.21203/rs.3.rs-4579465/v1

Caram, F. L., Rodrigues, B. R. D. O., Campanelli, A. S., & Parreiras, F. S. (2019). Machine learning techniques for code smells detection: A systematic mapping study. *International Journal of Software Engineering and Knowledge Engineering*, 29(02), 285-316. https://doi.org/10.1142/S021819401950013X

Cruz, D., Santana, A., & Figueiredo, E. (2020, June). Detecting bad smells with machine learning algorithms: an empirical study. In ACM (Eds.), *TechDebt '20: Proceedings of the 3rd International Conference on Technical Debt* (pp. 31-40). ACM. https://doi.org/10.1145/3387906.3388618

Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., & De Lucia, A. (2018, March). Detecting code smells using machine learning techniques: Are we there yet? In IEEE (Eds.), *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (saner)* (pp. 612-621). IEEE. https://doi.org/10.1109/SANER.2018.8330266

Fontana, F. A., Braione, P., & Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *The Journal of Object Technology*, *11*(2), 5-1. https://doi.org/10.5381/jot.2012.11.2.a5

Forero-Corba, W., & Bennasar, F. N. (2024). Técnicas y aplicaciones del Machine Learning e Inteligencia Artificial en educación: una revisión sistemática. *RIED-Revista Iberoamericana de Educación a Distancia*, *27*(1), 209-253. https://doi.org/10.5944/ried.27.1.37491

Falahi, T., Nassreddine, G., & Younis, J. (2023). Detecting data outliers with machine learning. *Al-Salam Journal for Engineering and Technology*, *2*(2), 152-164. https://doi.org/10.55145/ajest.2023.02.02.018

Fowler, M. (2018). *Refactoring: Improving the design of existing code*. Addison-Wesley Professional.

García, R. (2021). El perceptrón: una red neuronal artificial para clasificar datos. *Revista de Investigación en Modelos Matematicos Aplicados a la Gestión de la Economía*, *8*(1), 1-14. https://www.economicas.uba.ar/investigacion/wp-content/uploads/Garcia-Roberto-1.pdf

Guggulothu, T., & Moiz, S. A. (2020). Code smell detection using multi-label classification approach. *Software Quality Journal*, *28*(3), 1063-1086. https://doi.org/10.1007/s11219-020-09498-y

Hall, M. A. (2000). Correlation-based feature selection of discrete and numeric class machine learning. https://researchcommons.waikato.ac.nz/server/api/core/bitstreams/95d64129-4e47-485b-a6f4-bca127238988/content.

Hernández Vargas, L. A. (2015). *Selección de la metodología para determinar atipicos en las bases de cálculo de un índice de costos* [Trabajo de Investigación Aplicada para Especialización en Estadística Aplicada, Fundacion Universitaria los Libertadores]. https://repository.libertadores.edu.co/server/api/core/bitstreams/06bd2e2c-db97-4769-8d05-fa5ac769d729/content.

Kiyak, E. O., Birant, D., & Birant, K. U. (2019, October). *Comparison of multi-label classification algorithms for code smell detection* [Conference article]. 2019 3rd international symposium on multidisciplinary studies and innovative technologies (ISMSIT). https://doi.org/10.1109/ISMSIT.2019.8932855

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, *160*(1), 3-24.

Luiz, F. C., de Oliveira Rodrigues, B. R., & Parreiras, F. S. (2019, May). *Machine learning techniques for code smells detection: An empirical experiment on a highly imbalanced setup* [Conference article]. XV Brazilian Symposium on Information Systems. https//doi.org/10.1145/3330204.3330275

Nguyen Thanh, B., Nguyen NH, M., Le Thi My, H., & Nguyen Thanh, B. (2022, December). ml-Codesmell: A code smell prediction dataset for machine learning approaches. In ACM (Eds.), *Proceedings of the 11th International Symposium on Information and Communication Technology* (pp. 368-374). ACM. https://doi.org/10.1145/3568562.3568643

Noroozi, Z., Orooji, A., & Erfannia, L. (2023). Analyzing the impact of feature selection methods on machine learning algorithms for heart disease prediction. *Scientific reports*, *13*(1), 22588. https://doi.org/10.1038/s41598-023-49962-w

Paiva, T., Damasceno, A., Figueiredo, E., & Sant'Anna, C. (2017). On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, *5*(1), 7. https://doi.org/10.1186/s40411-017-0041-1

Palma-Mendoza, R. J., de-Marcos, L., Rodriguez, D., & Alonso-Betanzos, A. (2019). Distributed correlation-based feature selection in spark. *Information Sciences*, *496*, 287-299. https://doi.org/10.1016/j.ins.2018.10.052

Tempero, E. (2011). *Qualitas Corpus* [Dataset]. http://qualitascorpus.com/

Ramírez-Gallego, S., Krawczyk, B., Garca, S., Woniak, M., & Herrera, F. (2017). A survey on data preprocessing for data stream mining. *Neurocomputing*, *239*(C), 39-57. https://doi.org/10.1016/j.neucom.2017.01.078

Raymaekers, J., & Rousseeuw, P. J. (2024). Transforming variables to central normality. *Machine Learning*, *113*(8), 4953-4975. https://doi.org/10.1007/s10994-021-05960-5

dos Reis, J. P., Brito e Abreu, F., & Carneiro, G F. (2022). *Code smells dataset (oracles)* [Dataset]. https://doi.org/10.5281/zenodo.6555241

dos Reis, J. P., Abreu, F. B. E., & Carneiro, G. D. F. (2022). Crowdsmelling: A preliminary study on using collective knowledge in code smells detection. *Empirical Software Engineering*, *27*(3), 69. https://doi.org/10.1007/s10664-021-10110-5

Vinay S. (2021). *Standardization in machine learning*. https://www.researchgate.net/publication/349869617_STANDARDIZATION_IN_MACHINE_LEARNING#fullTextFileContent

Yeo, I. K., & Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, *87*(4), 954-959.