Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

The effects of data balancing approaches: A case study

Paul Mooijman [a], Cagatay Catal [b,*], Bedir Tekinerdogan [a], Arjen Lommen [c], Marco Blokland [c]

[a] Information Technology Group, Wageningen University & Research, Wageningen, The Netherlands
[b] Department of Computer Science and Engineering, Qatar University, Doha, Qatar
[c] Wageningen Food Safety Research Institute, Wageningen University & Research, Wageningen, The Netherlands

ARTICLE INFO

Article history:
Received 6 January 2022
Received in revised form 31 August 2022
Accepted 15 November 2022
Available online 1 December 2022

Keywords:
Imbalanced dataset
Resampling
Supervised machine learning
Classification
Feature selection
Missing data
LC–MS
Hormone abuse detection
Cattle

ABSTRACT

Imbalanced datasets affect the performance of machine learning algorithms adversely. To cope with this problem, several resampling methods have been developed recently. In this article, we present a case study approach for investigating the effects of data balancing approaches. The case study concerns the discrimination between growth hormone treated and non-treated animals using Liquid Chromatography-High Resolution Mass Spectrometry (LC-HRMS) data. Our LC-HRMS dataset contains 1241 bovine urine samples, of which only 65 specimens were from animal studies and guaranteed to contain growth-stimulating hormones while the rest has been reported to be untreated, making it a ∼5% imbalanced dataset. In this research, classification algorithms, combined with resampling strategies and dimensionality reduction methods, were investigated to find a prediction model to correctly identify the samples of treated animals. Furthermore, to cope with a large number of missing data points in the given dataset, a replacement with random low values strategy was applied. Our results showed that the replacement method was effective, and LogisticRegression combined with the oversampling algorithms SMOTE or ADASYN, GaussianProcessClassifier with the oversampling algorithm SMOTE, and LinearDiscriminantAnalysis were the best performing models after log transformation of the dataset was followed by Recursive Feature Elimination.

1. Introduction

Imbalanced datasets affect the performance of machine learning algorithms adversely. Imbalanced or skewed datasets are datasets where the samples-of-interest form a minority within that dataset. Several examples of such datasets mentioned in the literature include fraud case detection cases among credit card users [1], software fault prediction [2,3], hacking activity identification within Internet data streams [4], and detection of patients within a healthy population [5].

To cope with the class imbalance, several resampling methods have been developed recently. Datasets containing many features often benefit from dimensionality reduction by reducing the number of features. However, most of the dimensionality reduction methods do not deal well with imbalanced datasets. Furthermore, most classifiers do not cope well with missing data points in the dataset. Several methods exist to deal with missing data points, but the assumptions on which these methods work, based on either discarding features containing missing data points, or imputation of the missing data points with values from neighboring samples, do not work well in all situations.

In this article, we present a case study approach for investigating the effects of data balancing approaches. The case study concerns the discrimination between growth hormone treated and non-treated animals using Liquid Chromatography-High Resolution Mass Spectrometry (LC-HRMS) data. The European Union forbids the use of growth-stimulating hormones for cattle fattening in the livestock industry.[1] To determine if cattle have illegally been treated with growth hormones, reliable detection methods are needed. Liquid Chromatography–Mass Spectrometry (LC–MS) analysis of cattle urine and blood serum samples is a commonly used method for detecting these growth hormones and their derivatives [6,7]. Although the technique is sensitive enough to detect minute amounts of compounds, distinguishing hormone abuse cases from naturally occurring hormones is difficult. It is further complicated by the large amounts of data that the LC-(HR)MS instrument produces. Furthermore, the expected small number of hormone-abuse cases within the Dutch cattle breeding industry makes it a typical case of an imbalanced dataset.

The goal of this research is to develop a classification model that can correctly identify cases of illegal hormone usage in the Dutch cattle industry. This was done by investigating a dataset

* Corresponding author.
E-mail address: ccatal@qu.edu.qa (C. Catal).

1 https://ec.europa.eu/food/food/chemical-safety/hormones-meat_en

https://doi.org/10.1016/j.asoc.2022.109853

produced by LC-HRMS analysis of a large number of bovine urine samples, of which a small sample number was treated with hormones. The imbalance in the dataset, the large number of features, and a large number of missing data points was a rather unique combination of dataset traits that, to our knowledge, has never been investigated before using machine learning methods. To achieve our goals, the following research questions were formulated:

- RQ-1: What is the state of the art in data balancing algorithms to deal with imbalanced datasets?
  - a RQ-1a: What is the state of the art in dealing with missing data points within a dataset?
  - b RQ-1b: Which state-of-the-art data balancing algorithms are being used for imbalanced datasets derived from LC–MS data?
- RQ-2: Which data balancing methods work best for training machine learning algorithms to detect illegal hormone usage in LC–MS analysis results of bovine urine samples?
  - a RQ-2a: Which data imputation strategy can be used for our LC–MS-data derived imbalanced dataset?
  - b RQ-2b: Which combination of feature selection, data balancing methods, and classification algorithm work best for detecting illegal hormone usage in LC–MS analysis results of bovine urine samples?

The dataset was produced by performing LC-HRMS analysis on a total of 1241 bovine urine samples, of which only 65 specimens were from animal studies and guaranteed to contain growth-stimulating hormones while the rest has been reported to be untreated, making it a ~5% imbalanced dataset. The trimmed LC-HRMS dataset contained 271 features and a multitude of missing data points that resulted from signals falling below the detection threshold of the LC–MS instrument. To the best of our knowledge, the combination of an imbalanced, high-dimensional dataset with an alternative requirement for dealing with missing data points has never been reported before. In the data availability statement section, we provide the source code of this project and the data samples used in this research.

The remainder of the paper is organized as follows. Section 2 provides the related work, followed by Section 3, which presents the adopted research method. Section 4 presents the results, Section 5 explains the discussion, and finally, Section 6 concludes the paper.

2. Related work

2.1. Machine learning applications in LC–MS research

We retrieved machine learning applications in LC–MS research by searching electronic databases, however, we observed that the cases mainly focused on metabolomics studies.

Liebal et al. [8] performed a review where they investigated commonly used machine learning methods such as Random Forest, Support Vector Machines, Artificial Neural Networks, and Genetic Algorithms. They mention the notorious complexity of spectrometry data and the problems that missing data, noisy data, and a high number of features can have on a classifier's learning skill. They stated that the data should be normalized. Their research did not include imbalanced datasets.

Bouwmeester et al. [9] used Deep Learning, which is a subset of machine learning, in LC–MS research to aid in the identification of proteins, however, no imbalanced datasets were mentioned either.

A few papers were found describing strategies for detecting illegal hormones using LC–MS analysis. However, they focused on the LC–MS detection and quantification of hormones in the samples. Machine learning strategies to identify abuse cases based on the found hormone patterns were neither described in these papers nor was there mentioning of imbalanced datasets.

Rocha et al. [10] proposed a new strategy to detect boldenone undecyclenate misuse in cattle using LC–MS. They collected serum samples from 4 treated and 8 control crossbred animals. They developed a statistical model to predict the boldenone treatment based on the selected biomarkers. While their research provides interesting results, they did not use machine learning algorithms and their sample size was rather limited compared to our dataset.

Benedetto et al. [11] aimed to identify molecular markers to detect the illegal use of sex steroids and B_2-agonists on veal calves. Their focus was to develop a Real-Time PCR array analysis for profiling multiple biomarkers. They used Partial Least Squares-Discriminatory Analysis (PLS-DA) to identify candidate biomarkers. This PLS-DA is based on the PLS regression technique, which is a statistical technique.

Benedetto et al. [12] stated that advanced statistics can overcome some limitations of current omics workflows and multi-omics data fusion can help for animal science research. They reviewed some biomarkers-based approaches related to domestic cattle.

Draisci et al. [13] focused on the LC–MS detection and quantification of nortestosterone, testosterone, progesterone, and their derivatives in bovine blood serum and urine samples and claimed a 99.99% confidence in their identification.

Rijk et al. [14] investigated the effect of the prohormone dehydroepiandrosterone (DHEA) on the accumulation of certain metabolites by ultraperformance liquid chromatography in combination with time-of-flight accurate mass spectrometry (UPLC-TOFMS) analysis of bovine urine samples. They refer to the paper of Angeletti et al. [15], who claimed that the testosterone vs. epitestosterone ratio, which is used as a marker for hormone abuse in humans, cannot be applied to detect hormone abuse cases in the cattle industry, due to a far higher 17α-hydroxysteroid oxidoreductase activity in calves than in men.

Verheyden et al. [16] described how ultra-high performance liquid chromatography coupled to a triple quadrupole mass spectrometry (U-HPLC-QqQ-MS-MS) can be used to detect minute amounts of illegal hormones in the wood of crates used for housing veal calves.

2.2. Machine learning research for imbalanced datasets

Richardson and Lidbury [17] investigated the resampling methods of simple downsizing, multiple downsizing, and SMOTE, and used a combination of Random Forest and SVM for classification. The Random Forest algorithm was used to limit the number of features to the top five variables. They claimed that rescaling and log transformation did not contribute significantly to the performance of their classifier, but that data rescaling is performed to some degree by the SVM classifier.

Low et al. [18] investigated random oversampling, random undersampling, and an extension of SMOTE named SMOTE-Nominal Continuous in combination with Gradient Boosting to develop an activity prediction model for vehicles. They make use of feature selection to decrease the size of their dataset, thereby the training time is reduced.

Karatas et al. [19] used six machine learning methods (K-Nearest Neighbors, Random Forest, Gradient Boosting, AdaBoost, Decision Tree, and Linear Discriminant Analysis) in their research on the detection of hacking attempts but limited the resampling strategies to SMOTE.

Kaya et al. [20] used seven classification methods (Random Forest, SVM, RusBoosted Trees, LDA, subspace discriminant algorithm, AdaBoost, and KNN) for software vulnerability prediction models and made a comparison between an unbalanced dataset and the resampling methods ADASYN, Borderline-SMOTE, cluster-SMOTE, and SMOTE.

Lin and Chen [21] used three classification algorithms (i.e., diagonal LDA (DLDA), random forests, and SVMs) to investigate the effects of high-dimensional datasets. They claimed that DLDA and Random Forests do not perform well when the dataset contains highly correlated features. Feature selection was found to significantly improve the performance of DLDA, however, rebalancing the dataset was not found to guarantee high performance for their classifiers.

Liu et al. [5] emphasized the importance of recursive feature addition for a network anomaly detection method and claimed to have found high-performance results for the Random Forest classifier when used in combination with a data balancing method based on KNN outlier detection.

Deep learning algorithms were found to become an increasingly more common practice, and several cases were found where this type of machine learning method was combined with the resampling of imbalanced datasets. Versions of SMOTE and Borderline-SMOTE were commonly used to achieve a balanced dataset, although RUS and ROS were found to be effective as well [22].

Jiang and Li [23] used a one-dimensional Convolutional Neural Network (CNN) for wind turbine malfunction detection, in which they suggested a modified version of SMOTE for rebalancing their dataset.

2.3. Dealing with missing data

Search for information involving strategies for dealing with missing data has led us to Jason Brownlee's website (https://machinelearningmastery.com/handle-missing-data-Python/). The suggested strategies for dealing with missing data involved the removal of columns containing missing data, the imputation of missing data values based on the available data of similar samples/features, and the use of algorithms that support the use of datasets containing missing data. Patrician [24] described the methods as listwise deletion, pairwise deletion, weighting techniques, and single and multiple imputations as variations on the removal and imputation strategies. Gorard [25] described a method that comes close to our approach, where missing data values are replaced with the mean of all known values. However, in our case, listwise deletion would result in the deletion of (almost) the whole dataset, while all the retrieved imputation methods assume that the missing data values can be replaced with values close to the known values. This assumption is not correct in our situation. None of the retrieved strategies are, therefore, applicable to our dataset. Finally, Jason Brownlee's suggested strategy for investigating both machine learning algorithms and coping with imbalanced datasets was the basis of this research [26].

2.4. Summary of the related work and contributions

In summary, our work builds on previous research in literature and aims to build a machine learning-based prediction model to correctly identify the samples of guaranteed treated animals. While earlier work used machine learning in LC–MS research, our focus is particularly on the detection of illegal hormone abuse in the Dutch cattle breeding industry. Furthermore, we had to deal with many challenges regarding the dataset such as imbalanced data distribution and most of the relevant work in this domain did not address imbalanced learning. Since we were able to work with real-world data retrieved from the cattle industry, different data pre-processing techniques had to be applied in this research.

We had to investigate several rescaling and dimensionality reduction strategies of the dataset and their effects on the performance of a set of classifiers. Recursive Feature Elimination (with LogisticRegression for feature importance determination) after log transformation of the dataset was found to significantly improve the performance of the classifiers. We investigated several classifiers in combination with oversampling and undersampling methods on the log-transformed + Recursive Feature Eliminated (RFE) dataset. LogisticRegression, GaussianProcessClassifier, and LinearDiscriminantAnalysis, combined with an oversampler (SMOTE, Borderline-SMOTE, or ADASYN), in some cases combined with an undersampler (Tomek Links or Neighbor Cleaning Rule) were identified as well-performing classifiers on our dataset, with limited numbers of false positive and false negative predictions on the test dataset.

3. Methodology

Once the literature review had supplied sufficient insight into the state-of-the-art approach for optimizing a classifier on an imbalanced dataset, in combination with resampling strategies and feature selection, a promising approach was investigated for our dataset. Details of the approach are discussed in Section 4, however, a summary of the methodology is described in this section.

3.1. Creating the dataset

The hormone-related data was extracted as follows:

- Extraction of hormone data from the LC-Orbitrap-MS data acquired in negative mode.
- Isotope patterns fitting the elemental compositions of the compounds given in the attachments as well as the elemental compositions of their phase II conjugates (sulphates and glucuronides) were filtered out of all individual data files.
- Estrogenic (generally C18), androgenic (generally C19), and gestagenic (generally C21) hormones are mostly present as sulphate and/or glucuronide conjugates.
- Prostaglandins (generally C20), lipoxins (C20), HODE/HETE/ETE (C18/C20), and bile acid (variable) type compounds are expected to be present as glucuronides or in the free form.
- Per the data file intensities of compounds with the same elemental composition were added together; this is done to eliminate the retention time dimension so as to simplify the matrix.

This led to the following structure: data file, elemental composition, and the sum of intensities. The dataset consisted of the LC–MS analysis results of 1241 bovine urine samples, of which 65 samples were derived from cows that were treated with growth-stimulating hormones. The remaining 1176 samples were assumed to be non-hormone-treated, although of only 21 samples this claim could be guaranteed.

If a hormone was not detected in a certain sample, a random 'low' threshold value was inserted in the cell in the range of 10.000 ± 5.000 since the detection threshold during LC–MS analysis was set at 10.000. A final column containing the class label for the hormone treatment was added as follows: *class_label=0* for non-hormone-treated samples, and *class_label=1* for hormone-treated samples. Samples for which the hormone treatment was unknown were assumed to be non-treated and were assigned to class 0.

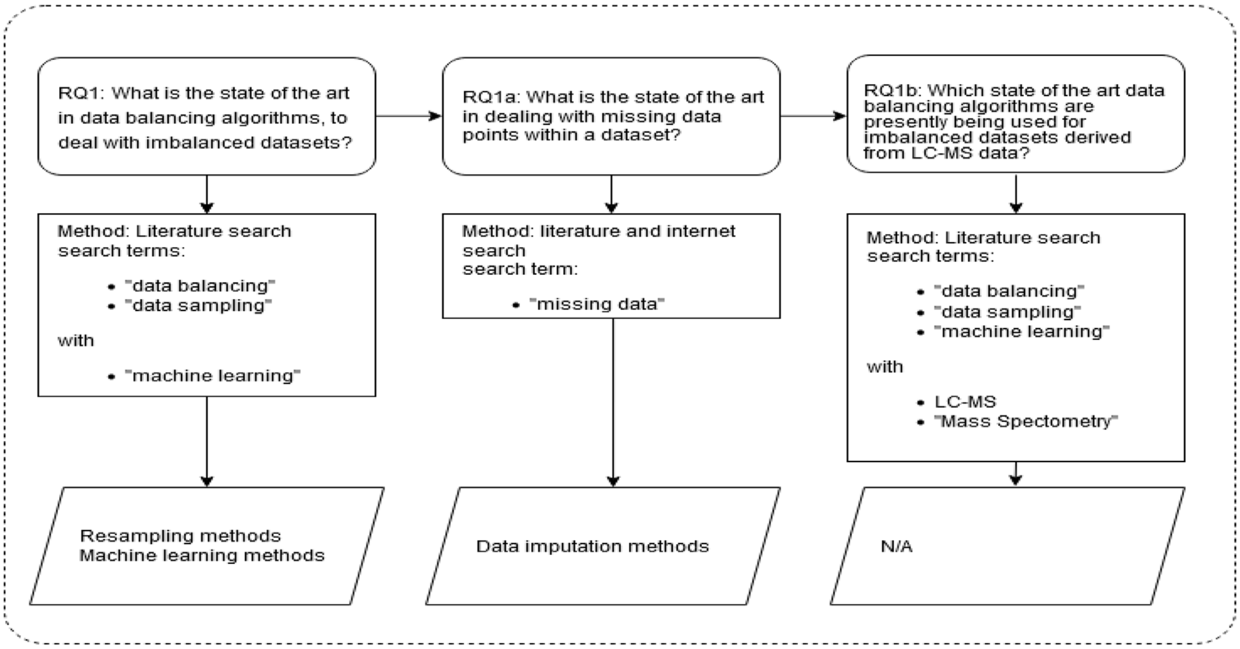

RQ1: What is the state of the art in data balancing algorithms, to deal with imbalanced datasets?
RQ1a: What is the state of the art in dealing with missing data points within a dataset?
RQ1b: Which state of the art data balancing algorithms are presently being used for imbalanced datasets derived from LC-MS data?
Method: Literature search search terms: "data balancing" "data sampling" with "machine learning"
Method: literature and internet search search term: "missing data"
Method: Literature search search terms: "data balancing" "data sampling" "machine learning" with LC-MS "Mass Spectometry"
Resampling methods Machine learning methods
Data imputation methods
N/A

Fig. 1. Overview of the strategy for answering the Research Question-1 (RQ-1).

Table 1
Basic statistical information of the dataset.

	C18H22O2	C18H22O3	C18H22O5S1	...	C33H50O17	C33H52O15
count	1241	1241	1241	...	1241	1241
mean	69713.16	92272.66	6689762	...	25507.1008	13772.5797
std	391191.6	355551.1	28719780	...	73740.5542	24864.3716
min	5016.415	5016.629	5006.237	...	5007.25441	5001.78862
25.00%	7767.79	8168.393	8675.246	...	7737.57733	7803.98978
50.00%	10456.37	11381.37	12435.46	...	10470.1746	10300.0278
75.00%	13175.11	14413.19	76087.16	...	13281.4815	12792.3322
max	6544897	4652422	354611500	...	971381.25	410041.25

The dataset was clearly not only imbalanced but also contained a large number of features, where a majority of the feature values were compensated for 'missing data'. These traits are known to complicate the learning process of machine learning algorithms.

3.2. Main steps

The LC–MS data files prepared for this research were combined into one table. Missing data points that were the result of signals in certain samples falling below the detection threshold of the LC-HRMS instrument were replaced by random values around the threshold in the range of 10.000 +/- 5.000.

The dataset was split into a training and test dataset, after which a baseline performance for several classifiers was determined. LogisticRegression was found to be the best performing classifier at this stage. Several normalizations and rescaling options were investigated, using LogisticRegression to determine the effects on a classifier's performance, in which log transformation of the dataset was found to give the best results. Several dimensionality reduction methods were investigated, with Recursive Feature Elimination (using the cost-sensitive version of LogisticRegression for feature importance determination) eventually providing the best results.

The produced log-transformed + Feature Eliminated dataset was used to re-test the best performing classifiers, combined with several undersampling and oversampling methods. Several cost-sensitive learning classifiers were also investigated. The hyperparameter tuning of the classifiers LinearDiscriminantAnalysis, GaussianProcessClassifier, and AdaBoost was investigated to improve their performance. Eventually, the best-performing combinations of resamplers and classifiers were tested on the test dataset. The strategy used to address RQ-1 is summarized in Fig. 1. The results for RQ-1 were summarized in the Related Work section. Fig. 2 shows the strategy that we used to address RQ-2. We elaborate on this in the following section.

4. Results

In this section, we provide answers to the following research questions: RQ-2, RQ-2a, RQ-2b.

4.1. Data analysis

4.1.1. Exploration of the dataset

For the data analysis, we developed a script that starts by reading the data file, producing a summary of the class distribution, then providing a statistical overview of the dataset as shown in Table 1.

A visual inspection of the feature information showed that none of the features followed a normal distribution. This made sense since each feature (i.e., hormone) is only present in a relatively small number of samples; each collection of feature values is, therefore, a combination of true detected intensity values and artificially generated intensity values for the 'below the threshold' cases as described above.

Both the Shapiro–Wilk test and the d'Agostino–Pearson test confirmed the finding that the features did not follow a Gaussian

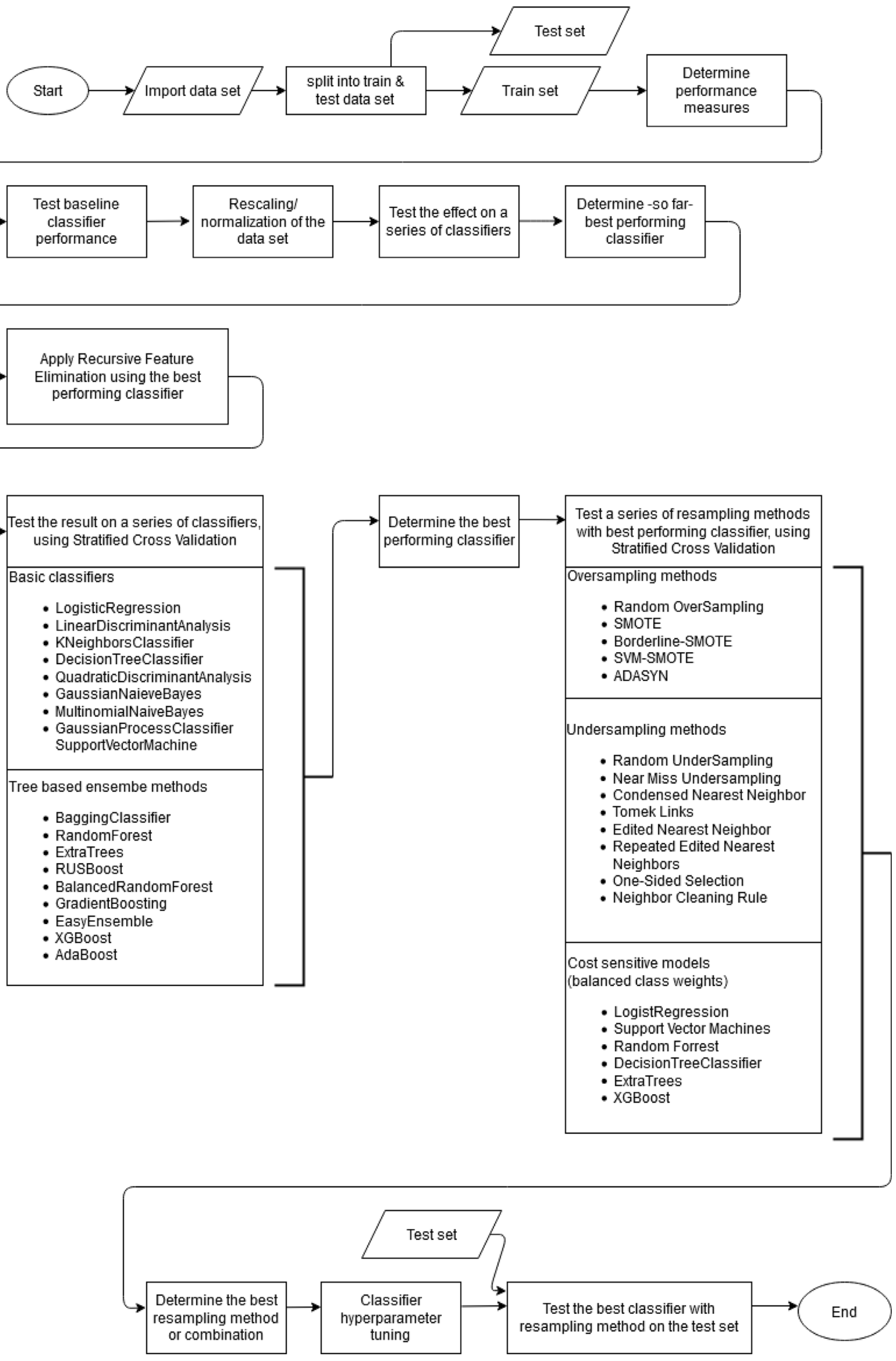

Start
Import data set
split into train & test data set
Test set
Train set
Determine performance measures
Test baseline classifier performance
Rescaling/ normalization of the data set
Test the effect on a series of classifiers
Determine -so far- best performing classifier
Apply Recursive Feature Elimination using the best performing classifier
Test the result on a series of classifiers, using Stratified Cross Validation
Basic classifiers
LogisticRegression
LinearDiscriminantAnalysis
KNeighborsClassifier
DecisionTreeClassifier
QuadraticDiscriminantAnalysis
GaussianNaieveBayes
MultinomialNaiveBayes
GaussianProcessClassifier
SupportVectorMachine
Tree based ensembe methods
BaggingClassifier
RandomForest
ExtraTrees
RUSBoost
BalancedRandomForest
GradientBoosting
EasyEnsemble
XGBoost
AdaBoost
Determine the best performing classifier
Test a series of resampling methods with best performing classifier, using Stratified Cross Validation
Oversampling methods
Random OverSampling
SMOTE
Borderline-SMOTE
SVM-SMOTE
ADASYN
Undersampling methods
Random UnderSampling
Near Miss Undersampling
Condensed Nearest Neighbor
Tomek Links
Edited Nearest Neighbor
Repeated Edited Nearest Neighbors
One-Sided Selection
Neighbor Cleaning Rule
Cost sensitive models (balanced class weights)
LogistRegression
Support Vector Machines
Random Forrest
DecisionTreeClassifier
ExtraTrees
XGBoost
Test set
Determine the best resampling method or combination
Classifier hyperparameter tuning
Test the best classifier with resampling method on the test set
End

Fig. 2. Overview of the strategy used for the Research Question-2 to find an optimal resampling and classifier combination.

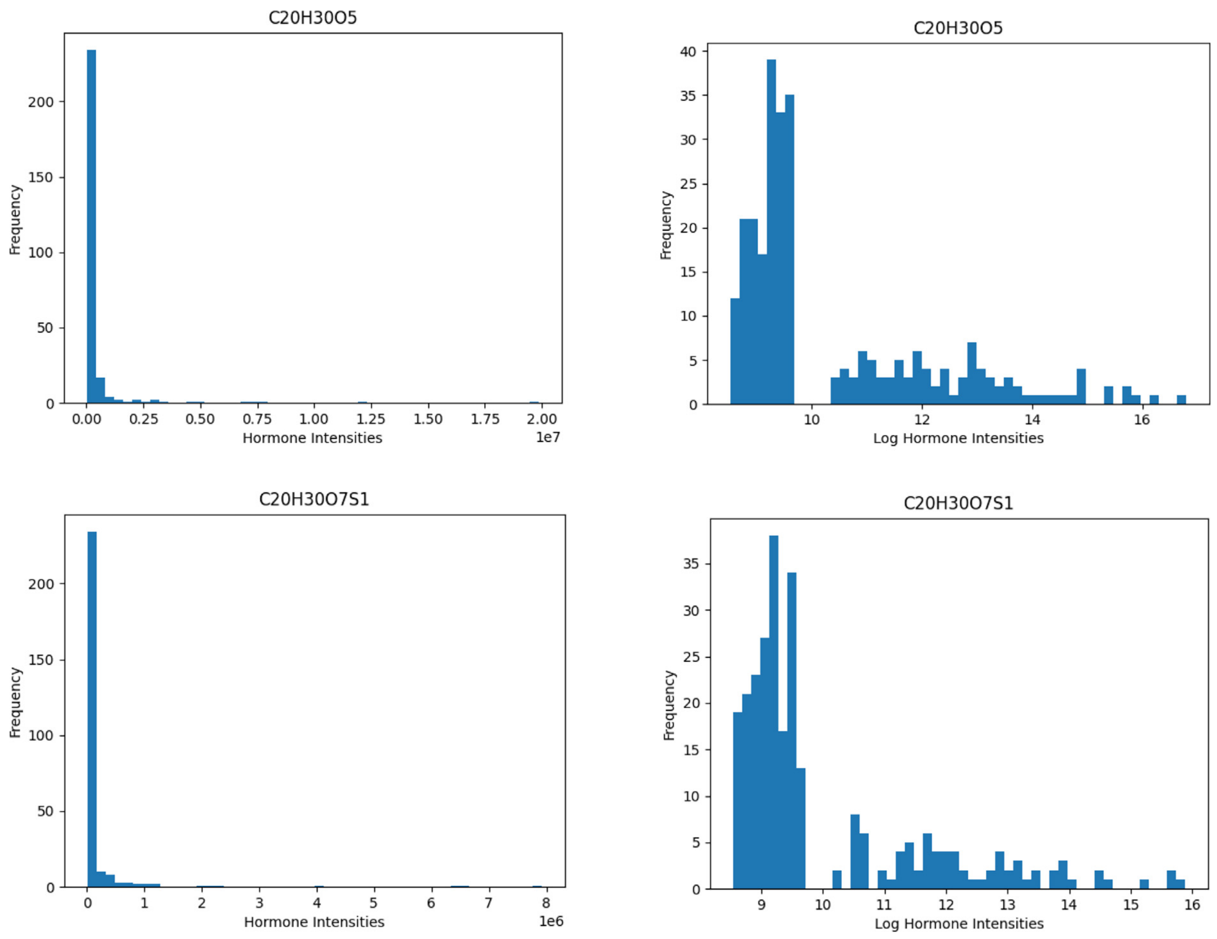


Fig. 3. Visual inspection of the distribution for two features (hormones $C_{20}H_{30}0_5$ and $C_{20}H_{30}O_7S_1$).

distribution. Visual inspection also suggested that a log transformation of the feature values might improve the distribution although no normal distribution could be achieved. This means that for statistical tests no tests assuming a normal distribution could be used. Fig. 3 shows the visual inspection of the distribution for two features (i.e., hormones C20H3005 and C20H30O7S1) before and after log transformation.

4.1.2. Splitting the dataset into a training and test set

A 70%–30% split between training and test set is not uncommon. However, due to the small number of minority samples in our dataset and the expected difficulties that this may imply for the learning skills of our classifiers, a training set that is as large as possible is preferred. It was, therefore, decided to go for a 90%–10% training/test set ratio. Class distribution of the training and test datasets are shown as follows:

Information of the training set:
Total number of samples: 1116
Total samples of class 0: 1058 (94.80%)
Total samples of class 1: 58 (5.20%)
Total number of features: 271
Information of the test set:
Total number of samples: 125
Total samples of class 0: 118 (94.40%)
Total samples of class 1: 7 (5.60%)
Total number of features: 271

4.1.3. Setting the performance baseline for the classifiers

The following nine basic classifiers were used for tests: LogisticRegression (LR), LinearDiscriminantAnalysis (LDA), KNeighborsClassifier (KNN), DecisionTreeClassifier (DecisionTree), QuadraticDiscriminantAnalysis (QDA), GaussianNaiveBayes (GaussianNB), MultinomialNaiveBayes (MNB), GaussianProcessClassifier (GaussianProc), and SupportVectorMachine (SVM).

The following nine decision tree-based ensemble algorithms were used for tests on the raw dataset: BaggingClassifier (Bagging), RandomForestClassifier (RandomForrest), ExtraTrees Classifier (ExtraTrees), RUSBoostClassifier (RusBoost), BalancedRandomForestClassifier (BalancedBagging), GradientBoostingClassifier (GradientBoosting), EasyEnsembleClassifier (EasyEnsemble), XGBClassifier (XGBoost), and AdaBoostClassifier (AdaBoost).

For comparison of the classifier performance, the following ten measures were selected: F1-score, Precision, Recall, Precision–Recall curve - Area Under the Curve (PR_AUC), Receiver Operating Characteristics curve - Area Under the Curve (ROC_AUC), Brier Skill Score, True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN). For evaluating the classifiers, a repeated stratified k-fold cross-validation strategy was used (i.e., the number of folds $k = 10$, and the number of repeats $n = 10$).

The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 4.

For each tested classifier, the mean measure scores with their standard deviations were summarized in Table 2.

Based on the F1 scores, LR AdaBoost, XGBoost, GradientBoosting, EasyEnsemble, and RusBoost were the only classifiers having an above no-skill performance, with LR and AdaBoost performing best. All other classifiers showed an F1 score below the no-skill threshold of 0.5.

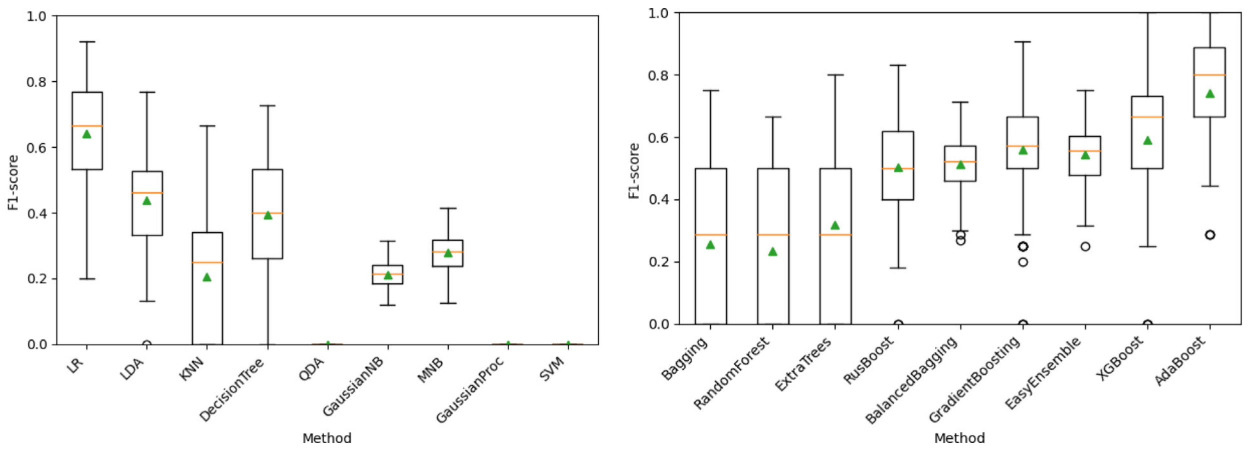


Fig. 4. Performance comparison of nine basic classifiers and nine decision tree-based ensemble classifiers on the raw dataset using RepeatedStratifiedKFold cross validation (k=10, n=10) and F1-score for performance measure.

Table 2
Performance comparison of nine basic classifiers and nine decision tree based ensemble classifiers on the raw dataset using RepeatedStratifiedKFold cross validation (k=10, n_repeats=10) and ten performance measures.

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.642	0.163	0.630	0.174	0.689	0.213	0.466	0.195	0.832	0.106	0.208	0.349	4.0	1.8	2.5	103.3
LDA	0.439	0.148	0.357	0.132	0.600	0.211	0.254	0.127	0.769	0.107	-0.633	0.538	3.5	2.3	6.6	99.2
KNN	0.205	0.183	0.361	0.354	0.154	0.139	0.137	0.102	0.570	0.069	-0.145	0.249	0.9	4.9	1.4	104.4
DecisionTree	0.393	0.173	0.391	0.177	0.421	0.203	0.220	0.124	0.692	0.101	-0.330	0.409	2.5	3.4	4.0	101.8
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.211	0.037	0.120	0.022	0.897	0.133	0.115	0.026	0.766	0.068	-6.185	1.105	5.2	0.6	38.7	67.1
MNB	0.279	0.060	0.170	0.039	0.791	0.168	0.150	0.044	0.788	0.082	-3.381	0.824	4.6	1.2	22.8	83.0
GaussianProc	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
SVM	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.261	0.233	0.610	0.477	0.174	0.168	0.212	0.159	0.587	0.084	0.116	0.185	1.0	4.8	0.1	105.7
RandomForest	0.242	0.216	0.610	0.488	0.156	0.147	0.200	0.139	0.578	0.073	0.109	0.155	0.9	4.9	0.0	105.8
ExtraTrees	0.325	0.238	0.730	0.444	0.219	0.178	0.260	0.169	0.610	0.089	0.177	0.188	1.3	4.5	0.0	105.8
RusBoost	0.517	0.198	0.658	0.249	0.458	0.208	0.353	0.182	0.722	0.104	0.152	0.314	2.7	3.1	1.5	104.3
BalancedBagging	0.509	0.088	0.366	0.079	0.860	0.134	0.326	0.088	0.887	0.066	-0.799	0.509	5.0	0.8	9.1	96.7
GradientBoosting	0.552	0.208	0.874	0.233	0.429	0.206	0.424	0.196	0.713	0.103	0.337	0.244	2.5	3.3	0.3	105.5
EasyEnsemble	0.544	0.094	0.401	0.086	0.873	0.138	0.361	0.099	0.899	0.069	-0.585	0.485	5.1	0.7	8.0	97.8
XGBoost	0.589	0.203	0.936	0.191	0.455	0.203	0.468	0.192	0.727	0.102	0.401	0.222	2.6	3.2	0.1	105.7
AdaBoost	0.741	0.166	0.919	0.130	0.649	0.204	0.619	0.198	0.823	0.102	0.561	0.235	3.8	2.0	0.4	105.4

4.1.4. Testing the effects of data rescaling techniques

Since baseline performance for the classifiers was determined, the effects of several normalizations and transformation techniques are needed to be investigated. Since AdaBoost is a complex and time-consuming classifier, it was decided to use LR for investigating the effect of these rescaling methods. The dataset was rescaled using Normalization, MinMaxScaler, StandardScaler, PowerTransformation, Log transformation, and several combinations of these re-scalers, after which its effect on the LR classifier performance was compared to the basic LR performance. In all cases, LR was run with the following hyperparameter settings (solver='liblinear' and max_iter=500). The classifier's training and validation was performed using cross-validation with the RepeatedStratifiedKFold method (n_splits = 10, n_repeats = 10, random_state = 1). The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 5.

The effects of the dataset modifications on the LR were summarized in Table 3.

Log transformation of the dataset alone and the combination of normalization + log transformation were found to provide the best performance on the LR classifier, followed by log transformation + standard scaling and normalization + PowerTransformation. It was further noticed that the order in which the modifications were applied to the dataset can have a large effect on the classifier's performance. In the following step, the log-transformed dataset and the normalized + log-transformed dataset were tested on the set of nine basic classifiers and nine decision tree-based ensemble classifiers. The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 6.

The effects of the log transformation or normalization + log transformation of the dataset on the performance of different classifiers were summarized in Tables 4 and 5. Table 4 shows the performance results of basic classifier algorithms and Table 5 represents the performance of decision tree-based classifiers.

On the log-transformed dataset, GaussianProc appeared to perform the best, followed by LDA and LR. LR, however, seemed to perform slightly better on the normalized + log-transformed dataset. To determine whether the difference was significant, a statistical test was required. First, a Shapiro–Wilk test was run to determine whether the F1-score distribution was Gaussian.

Since the Shapiro–Wilk test showed that most of the distributions were not Gaussian, Wilcoxon's signed-rank test was used to determine whether the classifiers performed significantly differently on the log-transformed and the normalized+log transformed datasets.

Results showed that the log-transformed dataset provided a significantly different performance on all classifiers as the normalized + log-transformed dataset, except for LR and LDA, which show no significant difference. Nevertheless, it was decided to

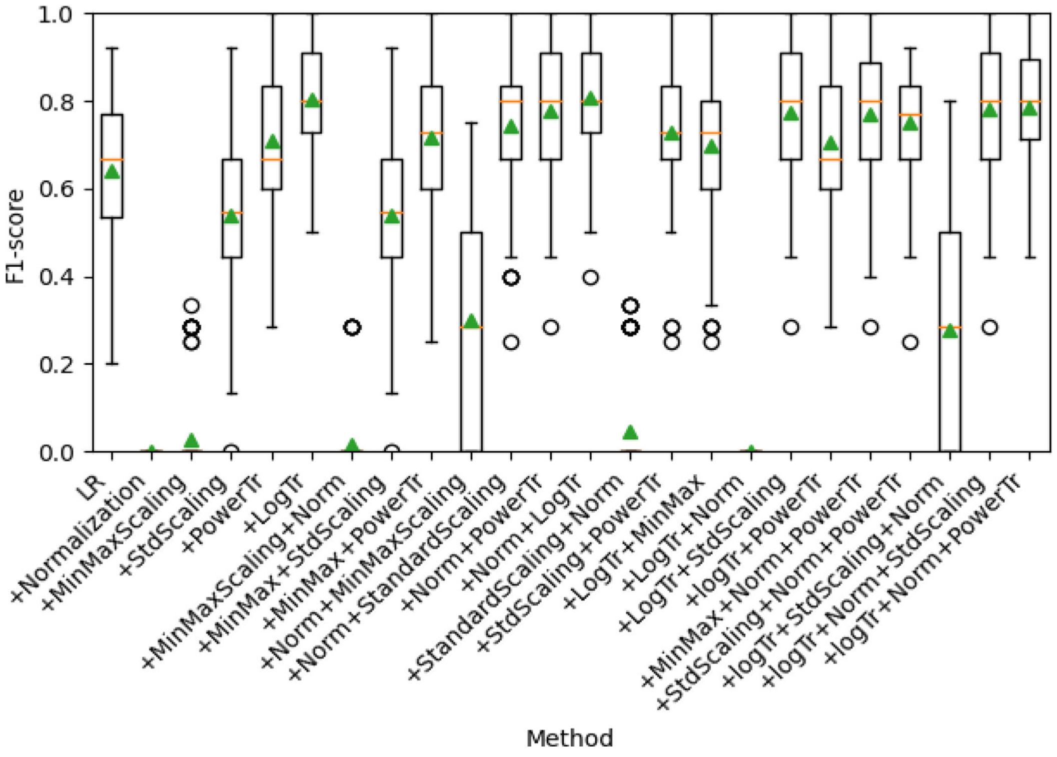


Fig. 5. Performance comparison of the LR classifier with 23 rescaling combinations of the dataset.

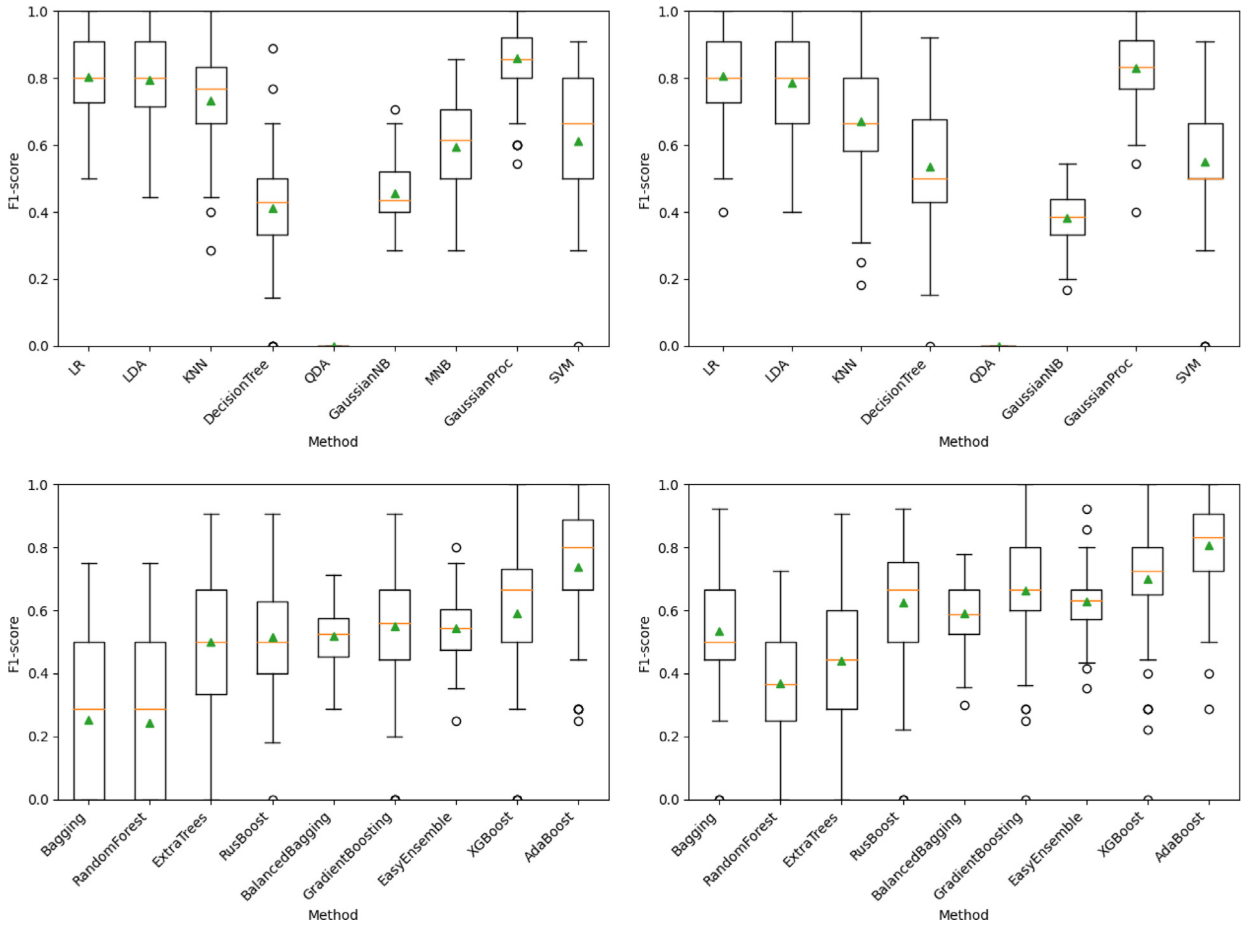


Fig. 6. Performance comparison of nine basic classifiers (top) and nine decision tree based ensemble classifiers (bottom) on the log transformed dataset (left) and the normalized + log transformed dataset (right).

Table 3
Performance comparison of the LogisticRegression classifier with 23 rescaling combinations.

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.642	0.163	0.630	0.174	0.689	0.213	0.466	0.195	0.832	0.106	0.208	0.349	4.0	1.8	2.5	103.3
+Normalization	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
+MinMaxScaling	0.028	0.085	0.090	0.277	0.017	0.051	0.066	0.045	0.508	0.025	-0.042	0.053	0.1	5.7	0.0	105.8
+StdScaling	0.539	0.172	0.545	0.195	0.561	0.203	0.351	0.174	0.767	0.102	0.003	0.391	3.2	2.5	2.9	102.9
+PowerTr	0.709	0.162	0.870	0.147	0.630	0.206	0.571	0.201	0.812	0.103	0.493	0.248	3.7	2.1	0.6	105.2
+LogTr	0.803	0.118	0.890	0.118	0.758	0.172	0.686	0.168	0.876	0.085	0.627	0.210	4.4	1.4	0.6	105.2
+MinMaxScaling+Norm	0.014	0.062	0.050	0.218	0.008	0.036	0.060	0.035	0.504	0.018	-0.046	0.038	0.1	5.8	0.0	105.8
+MinMax+StdScaling	0.539	0.172	0.545	0.195	0.561	0.203	0.351	0.174	0.767	0.102	0.003	0.391	3.2	2.5	2.9	102.9
+MinMax+PowerTr	0.717	0.154	0.788	0.161	0.695	0.199	0.567	0.193	0.841	0.099	0.450	0.267	4.0	1.8	1.2	104.5
+Norm+MinMaxScaling	0.302	0.221	0.725	0.444	0.199	0.161	0.239	0.153	0.599	0.080	0.151	0.174	1.1	4.7	0.0	105.8
+Norm+StandardScaling	0.743	0.147	0.781	0.162	0.734	0.184	0.598	0.193	0.861	0.092	0.479	0.283	4.3	1.5	1.3	104.5
+Norm+PowerTr	0.777	0.133	0.860	0.131	0.741	0.186	0.650	0.177	0.866	0.092	0.577	0.220	4.3	1.5	0.8	105.0
+Norm+LogTr	0.807	0.121	0.896	0.125	0.756	0.163	0.692	0.172	0.875	0.082	0.634	0.220	4.4	1.4	0.6	105.2
+StandardScaling+Norm	0.045	0.107	0.150	0.357	0.026	0.063	0.077	0.059	0.513	0.031	-0.027	0.067	0.1	5.7	0.0	105.8
+StdScaling+PowerTr	0.730	0.149	0.819	0.154	0.703	0.198	0.588	0.182	0.846	0.098	0.484	0.244	4.1	1.7	1.1	104.7
+LogTr+MinMax	0.700	0.162	0.936	0.116	0.587	0.193	0.571	0.185	0.792	0.096	0.512	0.215	3.4	2.4	0.3	105.5
+LogTr+Norm	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
+LogTr+StdScaling	0.775	0.146	0.892	0.126	0.714	0.192	0.653	0.193	0.854	0.096	0.592	0.232	4.2	1.6	0.6	105.2
+logTr+PowerTr	0.705	0.162	0.862	0.152	0.632	0.204	0.566	0.199	0.812	0.102	0.480	0.262	3.7	2.1	0.7	105.1
+MinMax+Norm+PowerTr	0.772	0.143	0.868	0.147	0.729	0.186	0.647	0.187	0.861	0.093	0.568	0.251	4.2	1.6	0.8	105.0
+StdScaling+Norm+PowerTr	0.750	0.129	0.843	0.140	0.703	0.175	0.610	0.164	0.848	0.087	0.527	0.213	4.1	1.7	0.9	104.9
+logTr+StdScaling+Norm	0.278	0.210	0.700	0.458	0.179	0.147	0.222	0.139	0.590	0.073	0.134	0.155	1.0	4.8	0.0	105.8
+logTr+Norm+StdScaling	0.782	0.135	0.880	0.132	0.731	0.178	0.659	0.182	0.862	0.089	0.593	0.227	4.2	1.6	0.7	105.1
+logTr+Norm+PowerTr	0.785	0.132	0.878	0.128	0.737	0.181	0.662	0.181	0.865	0.090	0.597	0.224	4.3	1.5	0.7	105.1

Table 4
Performance comparison of nine basic classifier algorithms on the log transformed dataset (top), and the normalized + log transformed dataset (bottom).

Log transformation

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.803	0.118	0.890	0.118	0.758	0.172	0.686	0.168	0.876	0.085	0.627	0.210	4.4	1.4	0.6	105.2
LDA	0.794	0.149	0.861	0.141	0.764	0.195	0.677	0.208	0.878	0.097	0.605	0.264	4.4	1.4	0.8	105.0
KNN	0.734	0.135	0.722	0.150	0.770	0.166	0.577	0.180	0.876	0.083	0.415	0.299	4.5	1.3	1.9	103.9
DecisionTree	0.411	0.158	0.409	0.170	0.440	0.184	0.229	0.123	0.701	0.092	-0.300	0.383	2.6	3.2	3.9	101.9
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.456	0.085	0.320	0.075	0.831	0.139	0.277	0.082	0.864	0.065	-1.162	0.633	4.8	1.0	10.9	94.9
MNB	0.596	0.133	0.518	0.136	0.730	0.184	0.404	0.150	0.845	0.092	-0.049	0.384	4.2	1.6	4.2	101.6
GaussianProc	0.859	0.106	0.849	0.121	0.886	0.139	0.762	0.167	0.938	0.070	0.697	0.219	5.2	0.7	1.0	104.8
SVM	0.612	0.189	0.990	0.099	0.468	0.197	0.495	0.187	0.734	0.099	0.439	0.208	2.7	3.1	0.0	105.8

Normalization + log transformation

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.807	0.121	0.896	0.125	0.756	0.163	0.692	0.172	0.875	0.082	0.634	0.220	4.4	1.4	0.6	105.2
LDA	0.787	0.141	0.857	0.146	0.756	0.188	0.665	0.195	0.874	0.094	0.587	0.261	4.4	1.4	0.8	105.0
KNN	0.670	0.150	0.653	0.170	0.712	0.176	0.494	0.185	0.845	0.089	0.258	0.355	4.2	1.6	2.4	103.4
DecisionTree	0.535	0.198	0.603	0.215	0.516	0.231	0.362	0.205	0.748	0.116	0.099	0.379	3.0	2.8	2.1	103.7
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.383	0.077	0.257	0.053	0.775	0.181	0.217	0.067	0.826	0.087	-1.637	0.524	4.5	1.3	13.1	92.7
GaussianProc	0.830	0.114	0.805	0.131	0.874	0.144	0.716	0.175	0.931	0.072	0.624	0.245	5.1	0.7	1.3	104.5
SVM	0.550	0.195	0.970	0.171	0.404	0.186	0.435	0.176	0.702	0.093	0.371	0.196	2.3	3.5	0.0	105.8

test both the log-transformed and normalization + log transformation dataset with LR and LDA as options for the Recursive Feature Elimination.

4.1.5. Dimensionality reduction by feature removal

Our raw dataset contains a total of 271 features. Most likely not all of these features are informative. Features may have no relationship to the class 1 specific traits and can, therefore, inhibit a classifier's learning process. There are several types of uninformative features. Low variance features are features with little variation among the samples, and, therefore, aid little in a classifier's learning process. They are better removed from the dataset. Highly correlated features are features that show similar patterns among their samples. Although they can be informative, the presence of both such features is known to complicate several classifiers' learning processes. If two features show a similar pattern, it is best to remove one of them.

Highly collinear features are features that may not share a similar pattern but do have a cause–effect relationship. The presence of such features can complicate a classifier's learning skills. To remove these features and simplify a dataset's complexity, there are several functions available in the scikit-learn software library (https://scikit-learn.org/stable/modules/feature_selection.html). Unsupervised feature selection methods are based on the analysis of the variance within a feature. They include but are not limited to, low variance, highly correlated, and multicollinear feature removal. Supervised feature selection methods try to select features based on their relevance to the class labels within the dataset. SelectKBest and Recursive Feature Elimination are two such methods.

4.1.5.1. Low variance feature removal. This function makes use of the sklearn function VarianceThreshold to remove all features whose variance among its samples is lower than the set threshold. A threshold of 0.90 or 0.95 is suggested to remove all low variance threshold features. Low Variance feature removal was attempted with these default thresholds, along with several other thresholds in the range of 0.5 to 1.0, but none resulted in the removal of any features.

Table 5
Performance comparison of nine decision tree based classifiers on the LogTransformed dataset (top), and the normalized + log transformed dataset (bottom).

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.252	0.232	0.595	0.483	0.167	0.166	0.207	0.157	0.583	0.083	0.111	0.182	1.0	4.8	0.1	105.7
RandomForest	0.243	0.223	0.600	0.490	0.158	0.155	0.202	0.147	0.579	0.078	0.112	0.164	0.9	4.9	0.0	105.8
ExtraTrees	0.500	0.222	0.920	0.271	0.361	0.193	0.394	0.183	0.681	0.097	0.326	0.204	2.1	3.7	0.0	105.8
RusBoost	0.516	0.185	0.671	0.233	0.450	0.199	0.349	0.186	0.718	0.100	0.155	0.304	2.6	3.2	1.5	104.3
BalancedBagging	0.518	0.090	0.375	0.082	0.861	0.129	0.334	0.092	0.889	0.064	-0.748	0.539	5.0	0.8	8.8	97.0
GradientBoosting	0.551	0.208	0.890	0.227	0.425	0.208	0.427	0.196	0.711	0.104	0.345	0.239	2.5	3.3	0.3	105.5
EasyEnsemble	0.545	0.094	0.402	0.087	0.875	0.136	0.362	0.100	0.900	0.068	-0.580	0.483	5.1	0.7	7.9	97.9
XGBoost	0.591	0.200	0.939	0.188	0.455	0.199	0.469	0.190	0.727	0.100	0.403	0.219	2.6	3.2	0.1	105.7
AdaBoost	0.737	0.164	0.920	0.131	0.644	0.203	0.615	0.195	0.820	0.101	0.556	0.233	3.8	2.0	0.4	105.4

Normalization+LogTransformation

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.476	0.226	0.864	0.288	0.350	0.202	0.366	0.190	0.674	0.101	0.284	0.222	2.0	3.8	0.2	105.6
RandomForest	0.309	0.228	0.730	0.444	0.205	0.168	0.247	0.160	0.603	0.084	0.162	0.177	1.2	4.6	0.0	105.8
ExtraTrees	0.354	0.233	0.780	0.414	0.240	0.178	0.279	0.169	0.620	0.089	0.198	0.188	1.4	4.4	0.0	105.8
RusBoost	0.619	0.177	0.749	0.209	0.566	0.207	0.457	0.187	0.777	0.103	0.306	0.281	3.3	2.5	1.3	104.5
BalancedBagging	0.534	0.085	0.386	0.082	0.899	0.123	0.354	0.088	0.908	0.060	-0.706	0.508	5.2	0.6	8.7	97.1
GradientBoosting	0.631	0.205	0.906	0.191	0.513	0.223	0.505	0.219	0.755	0.112	0.435	0.258	3.0	2.8	0.3	105.5
EasyEnsemble	0.580	0.105	0.431	0.100	0.918	0.125	0.405	0.114	0.924	0.064	-0.456	0.520	5.3	0.5	7.5	98.3
XGBoost	0.701	0.186	0.971	0.084	0.579	0.211	0.586	0.205	0.789	0.105	0.535	0.233	3.4	2.4	0.1	105.7
AdaBoost	0.773	0.164	0.969	0.093	0.671	0.205	0.669	0.200	0.835	0.103	0.625	0.232	3.9	1.9	0.1	105.7

4.1.5.2. Multicollinear feature removal. Features may not only be directly correlated; they may also have a strong cause–effect relationship (multicollinearity) that is not detected by performing a highly correlated feature removal. The presence of multicollinear features may complicate a classifier's learning skills. Therefore, it is recommended to remove all but one of such multicollinear features. Based on the script on the following link https://www.kaggle.com/ffisegydd/sklearn-multicollinearity-class, multicollinear feature removal was performed with its default threshold of 5.0. This script detects and discards the feature with the highest variance inflation factor (vif) and repeats this process until the set threshold is met. Features with a vif below 1 are considered non-correlated, a vif between 1–5 indicates that features are moderately correlated, while a vif above 5 indicates that features are highly correlated. Prior to the Multicollinear Feature Removal, the dataset was normalized and log-transformed. This Multicollinear Feature Removal discarded 266 of the 271 features. Testing this 5 feature counting dataset with the basic classifiers and ensemble classifiers resulted in a no-skill performance for all classifiers (PR_AUC and ROC_AUC scores of ∼0.5 and lower, and negative BrierSkill scores).

4.1.5.3. Feature removal with SelectKBest. Besides several unsupervised feature removal methods, there are also several supervised methods. The first supervised feature selection method that was tested made use of the sklearn function SelectKbest. This function makes use of several scoring functions to determine the feature's importance and removes all but the k-best features. For this experiment, the scoring functions chi2, f_class, and mutual_info_regression were tested. Although SelectKBest is a supervised method, it is not built to deal with imbalanced datasets. For this reason, our dataset was first run through the oversampling method Borderline-SMOTE, prior to running SelectKbest. The test was performed both on the normalized + log-transformed dataset and the log-transformed dataset. As k-value (i.e., the number of features to keep) values, the numbers 10, 20, 50, 100, 200, 250, and 260 were tested. The Borderline-SMOTE modified dataset was compared with several Borderline-SMOTE + SelectKBest-filtered datasets by using LogisticRegression and seeing the effect on its classification performance.

Used dataset:

- Log transformed dataset
- Normalized + log-transformed dataset

Tested modifications:

- Borderline-SMOTE oversampling
- Borderline-SMOTE + SelectKbest with chi2, k=10
- Borderline-SMOTE + SelectKbest with f_class, k=10
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=10
- Borderline-SMOTE + SelectKbest with chi2, k=20
- Borderline-SMOTE + SelectKbest with f_class, k=20
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=20
- Borderline-SMOTE + SelectKbest with chi2, k=50
- Borderline-SMOTE + SelectKbest with f_class, k=50
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=50
- Borderline-SMOTE + SelectKbest with chi2, k=100
- Borderline-SMOTE + SelectKbest with f_class, k=100
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=100
- Borderline-SMOTE + SelectKbest with chi2, k=200
- Borderline-SMOTE + SelectKbest with f_class, k=200
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=200
- Borderline-SMOTE + SelectKbest with chi2, k=250
- Borderline-SMOTE + SelectKbest with f_class, k=250
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=250
- Borderline-SMOTE + SelectKbest with chi2, k=260
- Borderline-SMOTE + SelectKbest with f_class, k=260
- Borderline-SMOTE + SelectKbest with mutual_info_regression, k=260

Tested basic classifiers:

- LogisticRegression (LR) (with max_iter=500)

The classifier's training and validation was performed using cross-validation with the RepeatedStratifiedKFold method (n_splits=10, n_repeats=10, random_state=1).

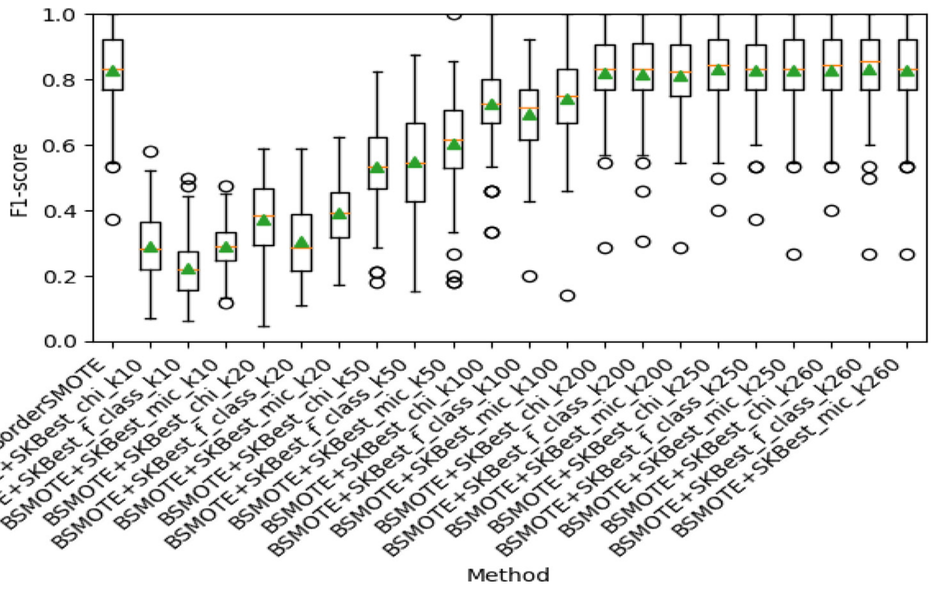


Fig. 7. Comparison of the effect of combinations of Borderline-SMOTE oversampling and SelectKBest Feature Selection on the log transformed dataset using LR classifier for validation.

Table 6
The effect of several combinations of Borderline-SMOTE + SelectKBest Feature Selection on the log transformed dataset on the classification performance of the LR classifier.

On log transformed dataset																
Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
BorderSMOTE	0.832	0.114	0.835	0.138	0.845	0.135	0.719	0.170	0.917	0.069	0.635	0.276	5.5	1.0	1.2	116.4
BSMOTE+SKBest_chi_k10	0.288	0.117	0.186	0.085	0.685	0.213	0.157	0.081	0.751	0.118	-2.873	1.400	4.5	2.0	21.6	96.0
BSMOTE+SKBest_f_class_k10	0.234	0.092	0.145	0.064	0.651	0.207	0.122	0.056	0.712	0.109	-3.737	1.323	4.2	2.3	26.8	90.8
BSMOTE+SKBest_mic_k10	0.287	0.073	0.183	0.049	0.688	0.194	0.149	0.054	0.758	0.093	-2.633	0.801	4.5	2.0	20.3	97.3
BSMOTE+SKBest_chi_k20	0.369	0.114	0.249	0.088	0.750	0.196	0.211	0.087	0.807	0.105	-1.869	1.003	4.9	1.6	15.9	101.7
BSMOTE+SKBest_f_class_k20	0.304	0.128	0.203	0.105	0.686	0.193	0.166	0.093	0.754	0.106	-2.728	1.474	4.5	2.0	20.9	96.7
BSMOTE+SKBest_mic_k20	0.389	0.108	0.278	0.094	0.693	0.200	0.217	0.090	0.794	0.097	-1.356	0.702	4.5	2.0	12.4	105.2
BSMOTE+SKBest_chi_k50	0.532	0.149	0.441	0.146	0.710	0.206	0.343	0.142	0.828	0.103	-0.335	0.500	4.6	1.9	6.3	111.3
BSMOTE+SKBest_f_class_k50	0.562	0.157	0.474	0.159	0.728	0.202	0.376	0.167	0.839	0.102	-0.236	0.551	4.7	1.8	5.8	111.8
BSMOTE+SKBest_mic_k50	0.574	0.130	0.530	0.165	0.670	0.177	0.378	0.144	0.816	0.087	-0.059	0.382	4.3	2.1	4.3	113.3
BSMOTE+SKBest_chi_k100	0.713	0.132	0.681	0.165	0.780	0.174	0.550	0.166	0.878	0.087	0.333	0.322	5.1	1.4	2.7	114.9
BSMOTE+SKBest_f_class_k100	0.701	0.130	0.675	0.144	0.760	0.178	0.532	0.165	0.869	0.088	0.322	0.303	4.9	1.6	2.6	115.0
BSMOTE+SKBest_mic_k100	0.735	0.125	0.717	0.152	0.778	0.156	0.576	0.172	0.879	0.078	0.401	0.306	5.1	1.4	2.2	115.4
BSMOTE+SKBest_chi_k200	0.827	0.111	0.828	0.139	0.842	0.134	0.711	0.161	0.916	0.068	0.625	0.252	5.5	1.0	1.3	116.3
BSMOTE+SKBest_f_class_k200	0.811	0.120	0.811	0.150	0.833	0.143	0.689	0.175	0.910	0.072	0.586	0.278	5.4	1.1	1.5	116.1
BSMOTE+SKBest_mic_k200	0.806	0.113	0.798	0.131	0.828	0.138	0.676	0.167	0.908	0.070	0.577	0.259	5.4	1.1	1.5	116.1
BSMOTE+SKBest_chi_k250	0.833	0.112	0.836	0.135	0.849	0.141	0.722	0.168	0.919	0.072	0.640	0.264	5.5	1.0	1.2	116.4
BSMOTE+SKBest_f_class_k250	0.839	0.110	0.843	0.132	0.850	0.136	0.730	0.166	0.920	0.069	0.654	0.255	5.5	1.0	1.1	116.5
BSMOTE+SKBest_mic_k250	0.823	0.113	0.821	0.133	0.841	0.137	0.705	0.169	0.915	0.070	0.616	0.267	5.5	1.0	1.3	116.3
BSMOTE+SKBest_chi_k260	0.833	0.111	0.832	0.137	0.849	0.129	0.720	0.167	0.919	0.065	0.633	0.274	5.5	1.0	1.3	116.3
BSMOTE+SKBest_f_class_k260	0.835	0.108	0.843	0.135	0.845	0.135	0.724	0.162	0.918	0.068	0.644	0.260	5.5	1.0	1.2	116.4
BSMOTE+SKBest_mic_k260	0.827	0.113	0.828	0.136	0.843	0.135	0.712	0.168	0.916	0.069	0.624	0.273	5.5	1.0	1.3	116.3

The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 7.

The effects of several combinations of Borderline-SMOTE + SelectKBest Feature Selection of the dataset on the LR classifier's performance were summarized in Table 6. The test was also performed on the normalized + log-transformed dataset, however, this did not only provide poorer results in all cases but also caused "nan" errors for the test performed with chi2 as a scoring function. As such, those results are not shown in the table.

Although it was expected that the classifier's performance would improve as the dataset is narrowed down more towards the best features, the classifier's performance decreased drastically as more features were removed. The best results were achieved when only 21 of the 271 features were discarded, using f_class as the scoring function with only a moderate improvement compared to the use of Borderline-SMOTE oversampling alone. For this reason, this method was not investigated any further.

4.1.5.4. Recursive feature elimination. Recursive Feature Elimination is another supervised feature selection algorithm. It needs a relatively well-performing classifier to determine the feature's importance, which is used to determine which features need to be removed. Since we were dealing with an imbalanced dataset, it was important not to discard features that were linked to class 1 (i.e., minority class). Therefore, the choice of the classifier-to-use for this feature importance determination was based on the results of our baseline classifier performance experiments. These experiments had shown that the choice of a suitable classifier was limited to GaussianProc, LDA, and LR. For Recursive Feature Elimination, a classifier is required to produce either a coef or a feature_importances_ attribute. Since GaussianProc does not produce such attributes, the choice was further limited to either LR or LDA.

Recursive Feature Elimination was performed on both the log-transformed dataset and the normalized + log-transformed dataset using LR or LDA for feature importance determination.

Table 7
Number of selected and discarded features after performing Recursive Feature Elimination on the log transformed or normalized + log transformed dataset, with either LR or LDA for feature importance determination.

	Recursive Feature Elimination			
	On log transformed dataset		On normalized + log transformed dataset	
Feature Importance classifier	LR	LDA	LR	LDA
Selected features	114	234	130	246
Discarded features	157	37	141	25
Original nr of features	271	271	271	271

Table 8
Performance comparison of nine basic classifiers on the log transformed dataset with Recursive Feature Elimination performed with either LR (top) or LDA (bottom) for feature importance determination.

Log transformed dataset

Feature selection with LogisticRegression

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.940	0.075	0.986	0.047	0.907	0.118	0.900	0.122	0.953	0.059	0.887	0.139	5.3	0.5	0.1	105.7
LDA	0.872	0.097	0.919	0.110	0.847	0.140	0.787	0.153	0.921	0.070	0.745	0.193	4.9	0.9	0.5	105.3
KNN	0.786	0.124	0.800	0.138	0.798	0.158	0.652	0.178	0.893	0.079	0.550	0.251	4.6	1.2	1.3	104.5
DecisionTree	0.393	0.175	0.409	0.180	0.404	0.209	0.222	0.130	0.686	0.103	-0.251	0.356	2.3	3.5	3.4	102.4
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.560	0.098	0.423	0.093	0.854	0.141	0.374	0.108	0.894	0.070	-0.451	0.438	5.0	0.8	7.1	98.7
GaussianProc	0.839	0.114	0.801	0.150	0.903	0.123	0.732	0.178	0.944	0.062	0.622	0.286	5.2	0.6	1.5	104.3
SVM	0.732	0.162	1.000	0.000	0.602	0.195	0.622	0.185	0.801	0.098	0.580	0.206	3.5	2.3	0.0	105.8

Feature selection with LinearDiscriminantAnalysis

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.833	0.108	0.916	0.111	0.784	0.154	0.730	0.162	0.890	0.077	0.682	0.201	4.6	1.2	0.5	105.3
LDA	0.827	0.133	0.898	0.126	0.792	0.182	0.725	0.188	0.893	0.091	0.671	0.235	4.6	1.2	0.6	105.2
KNN	0.735	0.136	0.734	0.156	0.761	0.167	0.580	0.184	0.872	0.084	0.425	0.295	4.4	1.4	1.8	104.0
DecisionTree	0.385	0.164	0.383	0.188	0.418	0.189	0.212	0.121	0.689	0.095	-0.387	0.443	2.4	3.4	4.2	101.5
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.457	0.091	0.323	0.078	0.818	0.146	0.277	0.088	0.859	0.071	-1.109	0.612	4.8	1.1	10.5	95.3
GaussianProc	0.861	0.106	0.875	0.129	0.867	0.140	0.767	0.164	0.930	0.070	0.706	0.226	5.0	0.8	0.8	105.0
SVM	0.635	0.186	0.990	0.099	0.492	0.196	0.518	0.186	0.746	0.098	0.464	0.207	2.9	2.9	0.0	105.8

Table 9
Performance comparison of nine basic classifiers on the normalized + log transformed dataset with Recursive Feature Elimination performed with either LR (top) or LDA (bottom) for feature importance determination.

Normalized + log transformed dataset

Feature selection with LogisticRegression

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.904	0.096	0.967	0.068	0.861	0.139	0.841	0.144	0.930	0.070	0.818	0.163	5.0	0.8	0.2	105.6
LDA	0.854	0.110	0.905	0.120	0.827	0.150	0.760	0.167	0.911	0.075	0.710	0.221	4.8	1.0	0.6	105.2
KNN	0.758	0.124	0.755	0.138	0.781	0.161	0.608	0.174	0.883	0.081	0.480	0.258	4.5	1.2	1.6	104.2
DecisionTree	0.592	0.188	0.643	0.202	0.585	0.225	0.418	0.200	0.783	0.112	0.183	0.359	3.4	2.4	2.1	103.7
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.532	0.094	0.396	0.080	0.832	0.157	0.344	0.101	0.880	0.078	-0.559	0.394	4.8	1.0	7.6	98.2
GaussianProc	0.807	0.115	0.777	0.146	0.863	0.142	0.681	0.174	0.924	0.072	0.560	0.271	5.0	0.8	1.6	104.2
SVM	0.702	0.162	1.000	0.000	0.565	0.192	0.588	0.182	0.782	0.096	0.541	0.202	3.3	2.5	0.0	105.8

Feature selection with LinearDiscriminantAnalysis

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR	0.809	0.125	0.919	0.110	0.744	0.168	0.698	0.172	0.870	0.084	0.649	0.204	4.3	1.5	0.4	105.4
LDA	0.817	0.139	0.886	0.133	0.785	0.189	0.710	0.196	0.889	0.094	0.649	0.251	4.6	1.2	0.7	105.1
KNN	0.670	0.150	0.653	0.172	0.716	0.180	0.495	0.185	0.847	0.091	0.256	0.354	4.2	1.6	2.4	103.4
DecisionTree	0.522	0.190	0.584	0.210	0.517	0.237	0.347	0.189	0.748	0.117	0.068	0.340	3.0	2.8	2.3	103.5
QDA	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.003	0.500	0.000	-0.055	0.004	0.0	5.8	0.0	105.8
GaussianNB	0.376	0.079	0.252	0.055	0.770	0.181	0.212	0.069	0.821	0.088	-1.707	0.576	4.5	1.3	13.5	92.3
GaussianProc	0.783	0.119	0.749	0.142	0.840	0.147	0.645	0.177	0.912	0.074	0.507	0.274	4.9	0.9	1.8	104.0
SVM	0.524	0.207	0.960	0.196	0.382	0.192	0.414	0.182	0.691	0.096	0.348	0.202	2.2	3.6	0.0	105.8

The sklearn function Recursive Feature Eliminator with CrossValidation (RFECV) was used with LR (solver= 'liblinear', class_weights ='balanced', max_iter=500) or LDA as model, and RepeatedStratifiedKfold for cross-validation (n_splits=10, n_repeats=10, random_state=1). Although the LR hyperparameter class_weights= 'balanced' was not used during the baseline and feature value scaling using, it was thought necessary to be used here to stimulate the selection of class 1-related features.

Results in Table 7 showed that more features were removed when the log-transformed dataset was used instead of the normalized + log-transformed dataset. Also, more features were removed when LR was used for feature importance determination compared to LDA. The effects of the four produced Recursive Feature Elimination datasets were tested on the classifiers.

Fig. 8 shows the performance comparison of the nine basic classifiers on the log-transformed dataset and normalized + log-transformed dataset with Recursive Feature elimination performed with LR and LDA for feature importance determination. Fig. 9 depicts the performance comparison of nine tree-based ensemble classifiers on the log-transformed and normalized + log-transformed dataset with Recursive Feature Elimination performed with LR and LDA for feature importance determination. Table 8 shows the performance comparison of nine basic classifiers on the log-transformed dataset with Recursive Feature Elimination performed with LR and LDA for feature importance

Table 10
Performance comparison of nine tree based ensemble classifiers on the log transformed dataset with Recursive Feature Elimination performed with either LR (top) or LDA (bottom) for feature importance determination.

Log transformed dataset

Feature selection with LogisticRegression

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.410	0.218	0.803	0.349	0.290	0.177	0.309	0.167	0.644	0.088	0.215	0.204	1.7	4.1	0.2	105.6
RandomForest	0.290	0.223	0.710	0.454	0.190	0.162	0.232	0.154	0.595	0.081	0.146	0.171	1.1	4.7	0.0	105.8
ExtraTrees	0.536	0.214	0.950	0.218	0.395	0.202	0.427	0.191	0.698	0.101	0.362	0.213	2.3	3.5	0.0	105.8
RusBoost	0.620	0.178	0.750	0.210	0.558	0.195	0.457	0.191	0.773	0.098	0.309	0.304	3.2	2.6	1.2	104.6
BalancedBagging	0.581	0.085	0.438	0.083	0.886	0.120	0.397	0.098	0.910	0.059	-0.380	0.416	5.1	0.7	6.9	98.9
GradientBoosting	0.625	0.181	0.905	0.168	0.502	0.190	0.489	0.190	0.749	0.095	0.416	0.227	2.9	2.9	0.3	105.5
EasyEnsemble	0.601	0.094	0.454	0.095	0.917	0.118	0.423	0.107	0.926	0.058	-0.334	0.470	5.3	0.5	6.8	99.0
XGBoost	0.596	0.191	0.933	0.187	0.463	0.197	0.470	0.178	0.731	0.098	0.405	0.199	2.7	3.1	0.2	105.6
AdaBoost	0.818	0.144	0.943	0.112	0.744	0.188	0.720	0.193	0.871	0.094	0.678	0.229	4.3	1.5	0.3	105.5

Feature selection with LinearDiscriminantAnalysis

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.285	0.232	0.652	0.461	0.191	0.169	0.227	0.160	0.595	0.085	0.129	0.190	1.1	4.7	0.1	105.7
RandomForest	0.234	0.211	0.600	0.490	0.149	0.141	0.193	0.134	0.575	0.071	0.102	0.149	0.9	4.9	0.0	105.8
ExtraTrees	0.500	0.220	0.930	0.255	0.362	0.198	0.395	0.188	0.681	0.099	0.327	0.209	2.1	3.7	0.0	105.8
RusBoost	0.534	0.172	0.666	0.208	0.478	0.197	0.359	0.157	0.732	0.097	0.179	0.244	2.8	3.0	1.5	104.3
BalancedBagging	0.515	0.091	0.372	0.080	0.861	0.139	0.332	0.093	0.889	0.069	-0.761	0.523	5.0	0.8	8.8	97.0
GradientBoosting	0.578	0.195	0.906	0.197	0.454	0.203	0.449	0.185	0.725	0.101	0.367	0.231	2.6	3.2	0.3	105.5
EasyEnsemble	0.552	0.095	0.408	0.086	0.878	0.134	0.370	0.102	0.903	0.067	-0.543	0.481	5.1	0.7	7.8	98.0
XGBoost	0.602	0.183	0.949	0.145	0.468	0.196	0.476	0.181	0.733	0.097	0.410	0.207	2.7	3.1	0.2	105.6
AdaBoost	0.754	0.157	0.927	0.118	0.662	0.193	0.635	0.192	0.829	0.097	0.581	0.224	3.9	1.9	0.3	105.5

Table 11
Performance comparison of nine tree based ensemble classifiers on the normalized + log transformed dataset with Recursive Feature Elimination performed with either LR (top) or LDA (bottom) for feature importance determination.

Normalized + log transformed dataset

Feature selection with LogisticRegression

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.485	0.248	0.880	0.325	0.354	0.213	0.388	0.202	0.677	0.107	0.319	0.224	2.1	3.7	0.0	105.8
RandomForest	0.328	0.225	0.760	0.427	0.218	0.165	0.259	0.156	0.609	0.082	0.175	0.173	1.3	4.5	0.0	105.8
ExtraTrees	0.419	0.245	0.830	0.376	0.296	0.198	0.332	0.188	0.648	0.099	0.257	0.209	1.7	4.1	0.0	105.8
RusBoost	0.668	0.177	0.802	0.187	0.609	0.214	0.518	0.201	0.800	0.106	0.406	0.271	3.5	2.3	1.0	104.8
BalancedBagging	0.582	0.089	0.435	0.092	0.911	0.111	0.403	0.099	0.921	0.055	-0.427	0.475	5.3	0.5	7.3	98.5
GradientBoosting	0.666	0.211	0.949	0.149	0.543	0.230	0.552	0.232	0.771	0.116	0.494	0.268	3.2	2.6	0.1	105.7
EasyEnsemble	0.598	0.106	0.445	0.104	0.936	0.116	0.425	0.117	0.934	0.061	-0.391	0.529	5.4	0.4	7.2	98.6
XGBoost	0.686	0.194	0.954	0.138	0.566	0.216	0.570	0.208	0.782	0.108	0.514	0.237	3.3	2.5	0.1	105.7
AdaBoost	0.787	0.156	0.972	0.076	0.690	0.202	0.686	0.192	0.844	0.101	0.646	0.217	4.0	1.8	0.1	105.7

Feature selection with LinearDiscriminantAnalysis

Classifier	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
Bagging	0.496	0.223	0.887	0.258	0.369	0.209	0.383	0.197	0.684	0.105	0.303	0.229	2.1	3.6	0.2	105.6
RandomForest	0.302	0.232	0.700	0.458	0.200	0.168	0.242	0.160	0.600	0.084	0.157	0.177	1.2	4.6	0.0	105.8
ExtraTrees	0.347	0.238	0.770	0.421	0.236	0.187	0.276	0.177	0.618	0.093	0.194	0.197	1.4	4.4	0.0	105.8
RusBoost	0.615	0.198	0.740	0.225	0.563	0.229	0.459	0.208	0.775	0.114	0.315	0.301	3.3	2.5	1.2	104.6
BalancedBagging	0.525	0.094	0.378	0.089	0.890	0.131	0.345	0.097	0.903	0.065	-0.765	0.557	5.2	0.6	9.0	96.8
GradientBoosting	0.648	0.205	0.913	0.185	0.530	0.222	0.524	0.219	0.764	0.111	0.457	0.258	3.1	2.7	0.3	105.5
EasyEnsemble	0.592	0.108	0.444	0.102	0.916	0.128	0.417	0.118	0.925	0.066	-0.382	0.508	5.3	0.5	7.1	98.7
XGBoost	0.682	0.187	0.973	0.083	0.557	0.212	0.566	0.202	0.778	0.106	0.512	0.229	3.2	2.6	0.1	105.7
AdaBoost	0.764	0.155	0.950	0.105	0.666	0.200	0.652	0.193	0.832	0.100	0.603	0.225	3.9	1.9	0.2	105.5

determination. Table 9 represents the performance comparison of nine basic classifiers on the normalized + log-transformed dataset with Recursive Feature Elimination performed with LR and LDA for feature importance determination. Table 10 shows the performance comparison of nine tree-based ensemble classifiers on the log-transformed dataset with Recursive Feature Elimination performed with either LR or LDA for feature importance determination. Table 11 represents the performance comparison of nine tree-based ensemble classifiers on the normalized + log-transformed dataset with Recursive Feature Elimination performed with LR and LDA (bottom) for feature importance determination.

These results showed that all basic classifiers performed best on the log-transformed dataset with Recursive Feature Elimination performed with LR for feature importance determination. The same was the case for the above no-skill performing tree-based ensemble classifiers. In general, the classifiers showed better performance on the log-transformed dataset compared to the normalized + log-transformed dataset, and better performance on the LR Recursive Feature Elimination compared to the LDA Recursive Feature Elimination. The best performing classifiers (in order of performance) were LR, LDA, GaussianProc, and AdaBoost.

To confirm that the two different Recursive Feature Elimination methods (LR vs. LDA) had a significant effect on the performance of the classifiers, a statistical test was performed. First, a Shapiro–Wilk test was performed on the F1 scores of all the results to see if the results followed a Gaussian distribution.

Most of the classifier performance measures using the F1 scores did not show a Gaussian distribution. For this reason, a Wilcoxon's signed-rank test was run to compare the performance of the used classifiers, on the Recursive Feature Eliminated dataset that was either produced using LR or LDA for feature selection.

The dataset that was produced using Recursive Feature Elimination with LR for feature importance determination provided significantly better performance for all tested classifiers, except

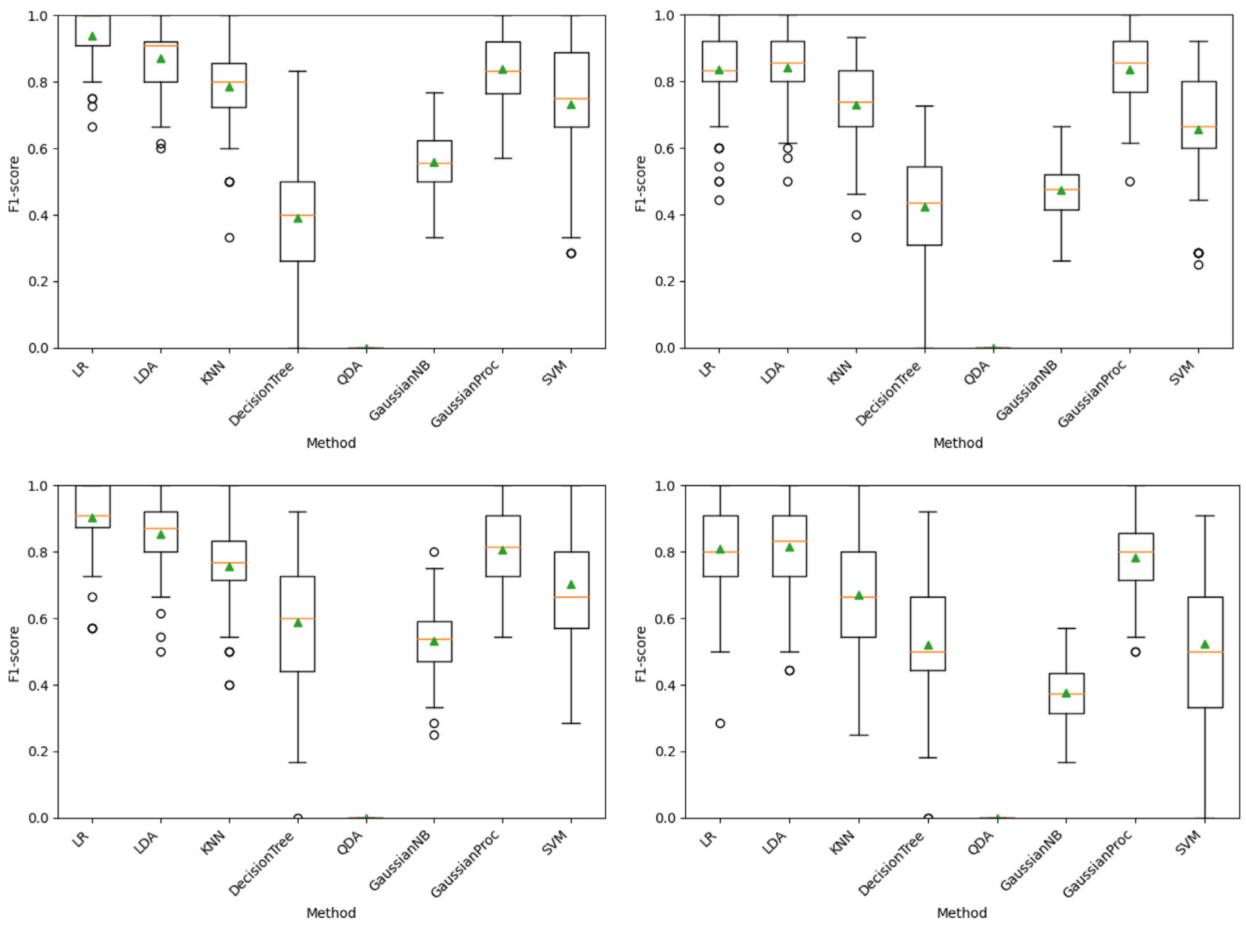


Fig. 8. Performance comparison of the nine basic classifiers on the log transformed dataset (top) or normalized + log transformed dataset (bottom).

when tested with KNN, DecisionTree, and XGBoost, in which case no significant difference was found. In a similar way, the effect of log transformation vs. normalization + log transformation of the dataset was tested. Again, a Wilcoxon's signed-rank test was performed to confirm that the performance difference was significant. For this, the F1-performance scores of the seven basic classifiers were compared after running them on the either log-transformed + RFE dataset or the normalized + log-transformed + RFE dataset.

Results demonstrated that the log-transformed dataset provided significantly better performance than the normalized + log-transformed dataset for all tested classifiers.

4.1.5.5. Highly correlated feature removal. Since the Spearman correlation matrix provided the impression that several features were highly correlated, these features need to be removed. This was done by first producing a correlation matrix of the feature data, followed by the removal of all features that were at least 90% correlated to other features. This was the case for 16 features, leaving 255 selected features.

Removed features are shown as follows:

[‘**C18H24O5S1**’, ‘**C19H26O6S1**’, ‘**C19H28O5S1**’, ‘**C19H28O6S1**’, ‘C19H30O5S1’, ‘**C19H30O6S1**’, ‘**C19H32O5S1**’, ‘C19H32O6S1’, ‘**C20H32O3**’, ‘C24H30O8’, ‘**C25H40O9**’, ‘C26H45N1O5S1’, ‘C27H42O11’, ‘**C27H42O8**’, ‘**C27H44O8**’, ‘C27H44O9’]

It was assumed that the removal of these highly correlated features, combined with the Recursive Feature Elimination, which at this stage had given the so-far best results, would give a further improvement of the classifier's performance. This assumption was tested by comparing the performance of the classifier with the results achieved with only Recursive Feature Elimination. However, this achieved no improvement, as such, this data is not shown here. Furthermore, of the 16 features removed by Highly correlated feature removal, 10 were also removed by Recursive Feature Elimination (marked in bold). For these reasons, this method was not considered an improvement and was, therefore, rejected.

4.2. Investigating the effect of undersampling and oversampling methods

Since LR provided the best performance of all tested classifiers, in this experiment, the effect of using different undersampling and oversampling methods was investigated. The oversampling methods ROS, SMOTE, Borderline-SMOTE, SVM-SMOTE, and ADASYN, and the undersampling methods RUS, Near Miss, CNN, Tomek Links, ENN, OSS, and NCL were tested using the ten performance measures. The functions for these methods were imported from the Python package imbalanced-learn 0.8.0. The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 10.

The effects of all tested oversamplers and undersamplers on the LogisticRegression classifier were summarized in Table 12. Of the oversampling methods SMOTE, Borderline-SMOTE and ADASYN provided the best results on the LR classifier. Of the undersampling methods Edited Nearest Neighbor Rule, One-Sided Selection, and Neighborhood Cleaning Rule provided the best results. These methods all improved the performance of the basic LR classifier, which optimum was so far found at an F1-score of 0.940.

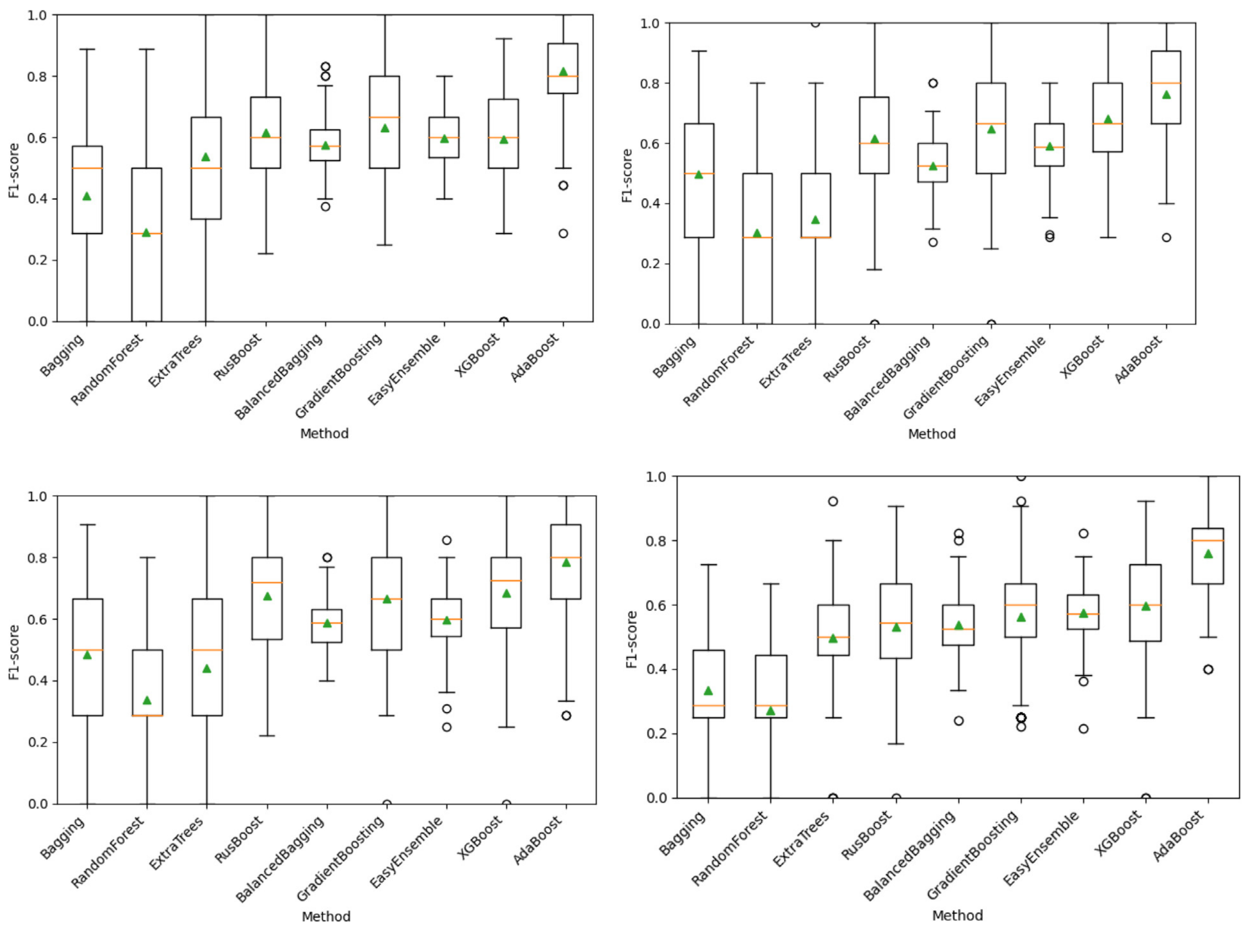


Fig. 9. Performance comparison of nine tree based ensemble classifiers on the log transformed (top) or normalized + log transformed dataset (bottom).

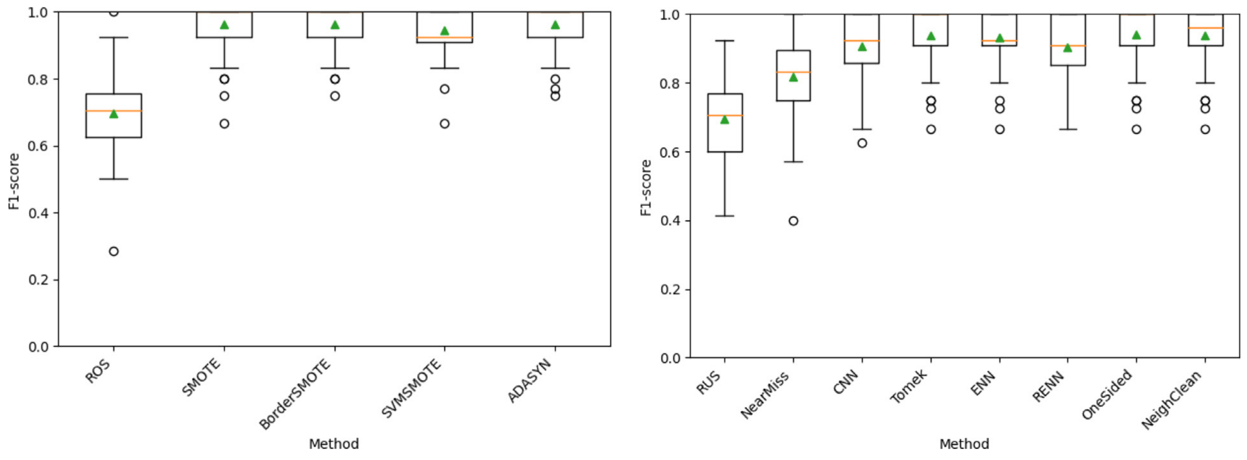


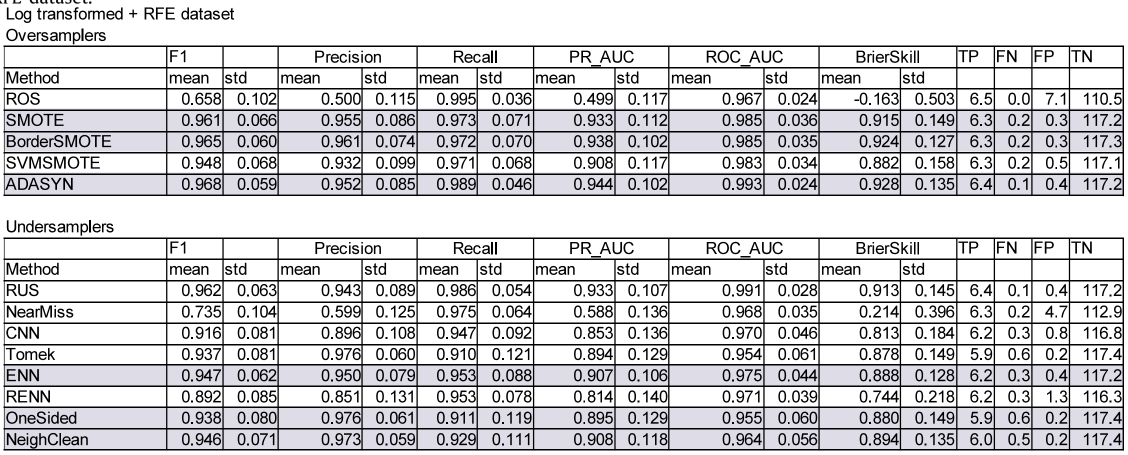
Fig. 10. The effects of several oversampling and undersampling methods on the performance of the LR classifier, on the log transformed + Recursive Feature filtered dataset (with LR for feature importance determination).

4.2.1. LogisticRegression classifier

The best performing oversampling methods SMOTE, Borderline-SMOTE, and ADASYN and the undersampling methods Tomek Links, Edited Nearest Neighbor Rule, One-Sided Selection, and Neighborhood Cleaning Rule were tested in all possible combinations to investigate the effect on the performance of the LR classifier. The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 11.

A Wilcoxon signed-rank test was performed to determine if the differences in performance were significant, this analysis result is provided in Appendix. Use of the oversamplers SMOTE, Borderline-SMOTE, and ADASYN provided significantly improved performance over the basic LR, and all performed equally well. Only adding an undersamplers provided no significantly different performance over the basic LR classifier. The addition of an undersampler to SMOTE provided no significant improvement. The addition of an undersampler to Borderline-SMOTE provided

Table 12
The effects of several oversampling and undersampling methods on the performance of the LR classifier tested on the log transformed + RFE dataset.


Log transformed + RFE dataset

Oversamplers

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
ROS	0.658	0.102	0.500	0.115	0.995	0.036	0.499	0.117	0.967	0.024	-0.163	0.503	6.5	0.0	7.1	110.5
SMOTE	0.961	0.066	0.955	0.086	0.973	0.071	0.933	0.112	0.985	0.036	0.915	0.149	6.3	0.2	0.3	117.2
BorderSMOTE	0.965	0.060	0.961	0.074	0.972	0.070	0.938	0.102	0.985	0.035	0.924	0.127	6.3	0.2	0.3	117.3
SVMSMOTE	0.948	0.068	0.932	0.099	0.971	0.068	0.908	0.117	0.983	0.034	0.882	0.158	6.3	0.2	0.5	117.1
ADASYN	0.968	0.059	0.952	0.085	0.989	0.046	0.944	0.102	0.993	0.024	0.928	0.135	6.4	0.1	0.4	117.2

Undersamplers

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
RUS	0.962	0.063	0.943	0.089	0.986	0.054	0.933	0.107	0.991	0.028	0.913	0.145	6.4	0.1	0.4	117.2
NearMiss	0.735	0.104	0.599	0.125	0.975	0.064	0.588	0.136	0.968	0.035	0.214	0.396	6.3	0.2	4.7	112.9
CNN	0.916	0.081	0.896	0.108	0.947	0.092	0.853	0.136	0.970	0.046	0.813	0.184	6.2	0.3	0.8	116.8
Tomek	0.937	0.081	0.976	0.060	0.910	0.121	0.894	0.129	0.954	0.061	0.878	0.149	5.9	0.6	0.2	117.4
ENN	0.947	0.062	0.950	0.079	0.953	0.088	0.907	0.106	0.975	0.044	0.888	0.128	6.2	0.3	0.4	117.2
RENN	0.892	0.085	0.851	0.131	0.953	0.078	0.814	0.140	0.971	0.039	0.744	0.218	6.2	0.3	1.3	116.3
OneSided	0.938	0.080	0.976	0.061	0.911	0.119	0.895	0.129	0.955	0.060	0.880	0.149	5.9	0.6	0.2	117.4
NeighClean	0.946	0.071	0.973	0.059	0.929	0.111	0.908	0.118	0.964	0.056	0.894	0.135	6.0	0.5	0.2	117.4

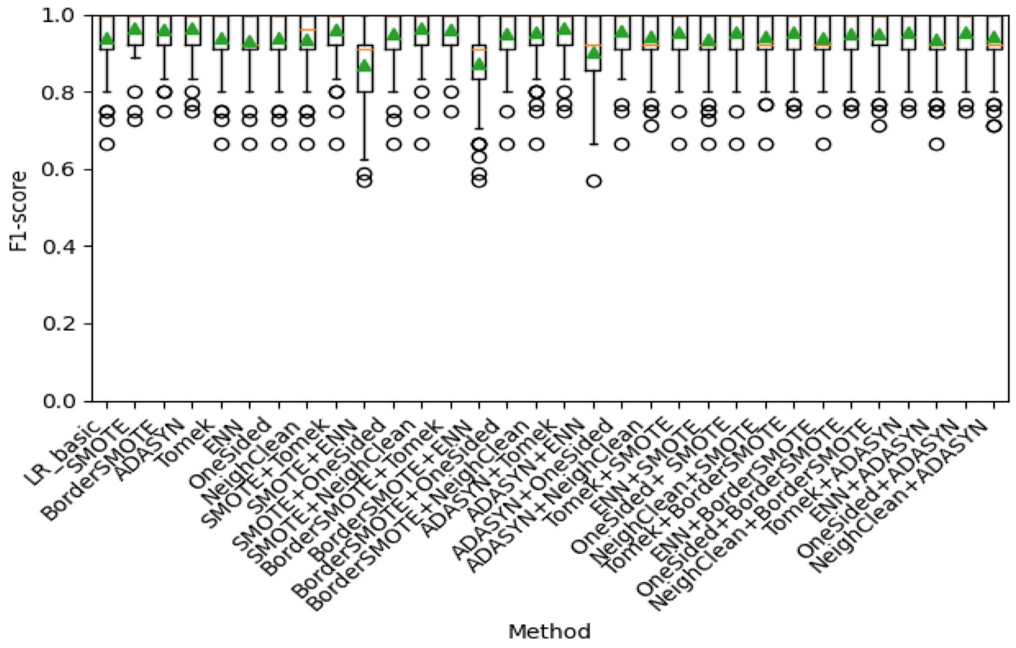


Fig. 11. The effects of several oversampling and undersampling combinations on the LR classifier performance (F1-score) tested on the log transformed + RFE dataset.

no significant improvement. The addition of an undersampler to ADASYN provided no significant improvement. Performing the ENN undersampling method, after an oversampler provided consistently significantly poorer results on the performance. It was further noticed that this was the only undersampler that provided better performance if undersampling took place before oversampling, but the addition of this undersampler provided no improved performance over running an oversampler alone. The effects of several oversampling and undersampling combinations on the performance of the LR classifier were summarized in Table 13.

4.2.2. LinearDiscriminantAnalyser classifier

For comparison reasons, the best performing oversampling methods SMOTE, Borderline-SMOTE, and ADASYN and the undersampling methods Tomek Links, Edited Nearest Neighbor Rule, One-Sided Selection, and Neighborhood Cleaning Rule were also tested in all possible combinations on the second-best classifier LDA. The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 12. In Table 14, the effects of several oversampling and undersampling combinations on the performance of the LDA classifier, tested on the log-transformed + RFE dataset are shown.

This test showed that the use of an oversampler, with or without the use of an undersampler, provided no improved performance over the basic LDA classifier.

4.2.3. GaussianProcessClassifier

For comparison reasons, the second-best performing classifier, GaussianProc was tested with the combinations of the oversampling methods SMOTE, Borderline-SMOTE, and ADASYN and the undersampling methods Tomek Links, Edited Nearest Neighbor Rule, One-Sided Selection, and Neighborhood Cleaning Rule. The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 13. Table 15 shows the effects of several oversampling and undersampling combinations on the performance of the GaussianProc, tested on the log-transformed + RFE dataset. This test showed that the addition of an oversampler, with or without the addition of an undersampler, provided poorer performance than using only the basic GaussianProc.

Table 13
The effects of several oversampling and undersampling combinations on the performance of the LR classifier, tested on the log transformed + RFE dataset.

On log transformed + RFE dataset

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR_basic	0.940	0.075	0.986	0.047	0.907	0.118	0.900	0.122	0.953	0.059	0.887	0.139	5.3	0.5	0.1	105.7
SMOTE	0.965	0.055	0.970	0.061	0.967	0.082	0.939	0.092	0.982	0.041	0.928	0.107	5.6	0.2	0.2	105.6
BorderSMOTE	0.961	0.057	0.963	0.066	0.965	0.079	0.932	0.098	0.981	0.040	0.919	0.115	5.6	0.2	0.2	105.6
ADASYN	0.965	0.055	0.967	0.066	0.968	0.074	0.938	0.095	0.983	0.037	0.927	0.114	5.6	0.2	0.2	105.6
Tomek	0.939	0.077	0.983	0.052	0.908	0.118	0.898	0.125	0.954	0.059	0.884	0.142	5.3	0.5	0.1	105.7
ENN	0.934	0.077	0.958	0.073	0.919	0.112	0.886	0.128	0.958	0.056	0.868	0.149	5.3	0.5	0.2	105.5
OneSided	0.940	0.077	0.984	0.051	0.908	0.118	0.900	0.125	0.954	0.059	0.886	0.142	5.3	0.5	0.1	105.7
NeighClean	0.938	0.075	0.974	0.061	0.914	0.118	0.894	0.122	0.956	0.059	0.879	0.140	5.3	0.5	0.2	105.6
SMOTE+Tomek	0.963	0.062	0.966	0.068	0.965	0.083	0.935	0.104	0.981	0.042	0.923	0.125	5.6	0.2	0.2	105.6
SMOTE+ENN	0.871	0.107	0.809	0.163	0.974	0.080	0.786	0.162	0.979	0.039	0.660	0.335	5.7	0.1	1.7	104.1
SMOTE+OneSided	0.951	0.072	0.966	0.066	0.943	0.102	0.915	0.119	0.970	0.051	0.901	0.138	5.5	0.3	0.2	105.6
SMOTE+NeighClean	0.964	0.060	0.965	0.069	0.968	0.077	0.937	0.101	0.983	0.039	0.924	0.123	5.6	0.2	0.2	105.6
BorderSMOTE+Tomek	0.962	0.057	0.966	0.065	0.963	0.077	0.933	0.098	0.980	0.039	0.921	0.115	5.6	0.2	0.2	105.6
BorderSMOTE+ENN	0.875	0.104	0.820	0.157	0.969	0.087	0.793	0.157	0.977	0.043	0.680	0.316	5.6	0.2	1.6	104.2
BorderSMOTE+OneSided	0.952	0.069	0.961	0.077	0.951	0.093	0.917	0.115	0.974	0.047	0.901	0.144	5.5	0.3	0.3	105.5
BorderSMOTE+NeighClean	0.956	0.066	0.963	0.069	0.956	0.094	0.924	0.110	0.977	0.047	0.910	0.129	5.5	0.2	0.2	105.6
ADASYN+Tomek	0.964	0.055	0.965	0.067	0.968	0.074	0.937	0.096	0.983	0.037	0.925	0.114	5.6	0.2	0.2	105.6
ADASYN+ENN	0.904	0.102	0.865	0.159	0.971	0.072	0.840	0.160	0.980	0.035	0.752	0.307	5.6	0.2	1.2	104.6
ADASYN+OneSided	0.958	0.064	0.963	0.076	0.959	0.083	0.927	0.109	0.978	0.042	0.912	0.134	5.6	0.2	0.2	105.5
ADASYN+NeighClean	0.941	0.070	0.928	0.106	0.966	0.075	0.898	0.118	0.981	0.037	0.867	0.170	5.6	0.2	0.5	105.3
Tomek+SMOTE	0.953	0.066	0.953	0.078	0.960	0.086	0.918	0.113	0.978	0.043	0.902	0.137	5.6	0.2	0.3	105.5
ENN+SMOTE	0.936	0.069	0.914	0.100	0.970	0.083	0.887	0.115	0.982	0.041	0.856	0.155	5.6	0.2	0.6	105.2
OneSided+ SMOTE	0.954	0.067	0.952	0.080	0.962	0.085	0.919	0.114	0.979	0.043	0.902	0.139	5.6	0.2	0.3	105.5
NeighClean+SMOTE	0.945	0.065	0.928	0.088	0.969	0.075	0.902	0.112	0.982	0.038	0.879	0.140	5.6	0.2	0.5	105.3
Tomek+BorderSMOTE	0.954	0.066	0.952	0.084	0.963	0.081	0.919	0.113	0.980	0.041	0.901	0.143	5.6	0.2	0.3	105.5
ENN+BorderSMOTE	0.938	0.066	0.917	0.095	0.972	0.085	0.891	0.110	0.983	0.042	0.863	0.142	5.6	0.2	0.6	105.2
OneSided+BorderSMOTE	0.950	0.066	0.948	0.083	0.959	0.083	0.912	0.114	0.978	0.042	0.894	0.141	5.6	0.2	0.3	105.5
NeighClean+BorderSMOTE	0.950	0.066	0.937	0.088	0.970	0.076	0.911	0.112	0.983	0.038	0.890	0.143	5.6	0.2	0.4	105.4
Tomek+ADASYN	0.953	0.064	0.946	0.081	0.965	0.076	0.916	0.111	0.981	0.038	0.898	0.135	5.6	0.2	0.3	105.5
ENN+ADASYN	0.935	0.069	0.912	0.100	0.970	0.082	0.886	0.116	0.982	0.041	0.854	0.154	5.6	0.2	0.6	105.2
OneSided+ADASYN	0.953	0.063	0.948	0.082	0.965	0.076	0.917	0.110	0.981	0.039	0.898	0.136	5.6	0.2	0.3	105.5
NeighClean+ADASYN	0.941	0.068	0.923	0.095	0.968	0.074	0.896	0.115	0.982	0.037	0.870	0.153	5.6	0.2	0.5	105.3

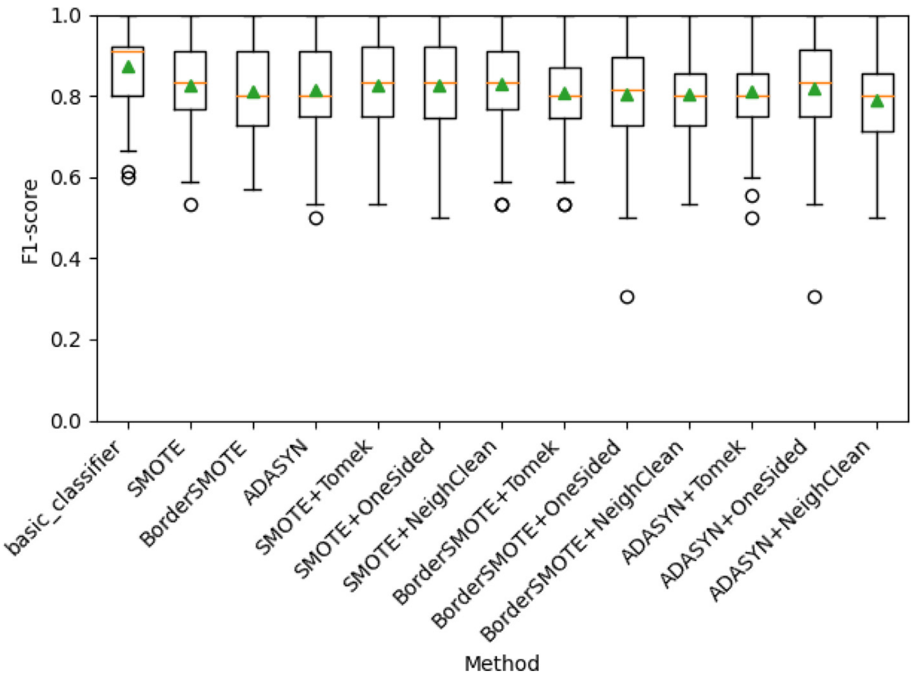


Fig. 12. The effects of several oversampling and undersampling combinations on the LDA classifier performance (F1-score), tested on the log transformed + RFE dataset.

4.2.4. Combinations of oversampling and undersampling

Finally, the Adaboost classifier was tested with combinations of the oversampling methods SMOTE, Borderline-SMOTE, and ADASYN and the undersampling methods Tomek Links, Edited Nearest Neighbor Rule, One-Sided Selection, and Neighborhood Cleaning Rule. The mean F1 scores were plotted in a box–whisker plot, as shown in Fig. 14. The effects of several oversampling and undersampling combinations on the performance of AdaBoost tested on the log-transformed + RFE dataset are shown in Table 16.

Again, a Wilcoxon's signed-rank test was performed to determine whether the observed differences were significant.

This test showed that Adaboost combined with Borderline-SMOTE+NeighborCleaningRule performed significantly better on the log transformed+ RFE dataset. AdaBoost combined with SMOTE+One-Sided Selection performed significantly poorer. All

Table 14
The effects of several oversampling and undersampling combinations on the performance of the LDA classifier, tested on the log transformed + RFE dataset.

On log transformed + RFE dataset

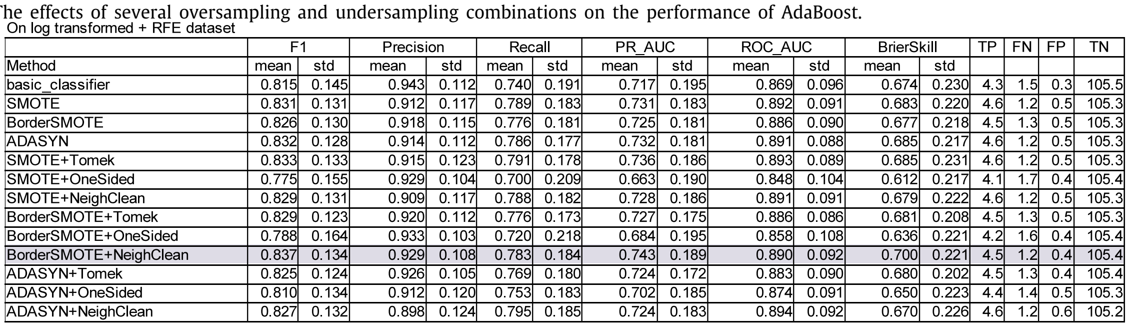
Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
basic_classifier	0.872	0.097	0.919	0.110	0.847	0.140	0.787	0.153	0.921	0.070	0.745	0.193	4.9	0.9	0.5	105.3
SMOTE	0.826	0.101	0.757	0.151	0.940	0.105	0.713	0.150	0.960	0.052	0.557	0.309	5.5	0.3	2.1	103.7
BorderSMOTE	0.811	0.107	0.732	0.155	0.939	0.106	0.691	0.162	0.959	0.053	0.513	0.310	5.5	0.3	2.3	103.5
ADASYN	0.814	0.112	0.719	0.151	0.963	0.092	0.697	0.165	0.970	0.047	0.500	0.352	5.6	0.2	2.5	103.3
SMOTE+Tomek	0.824	0.108	0.752	0.154	0.942	0.102	0.711	0.162	0.961	0.051	0.549	0.321	5.5	0.3	2.1	103.7
SMOTE+OneSided	0.825	0.114	0.781	0.153	0.905	0.133	0.712	0.172	0.944	0.066	0.578	0.310	5.2	0.6	1.8	104.0
SMOTE+NeighClean	0.829	0.103	0.763	0.155	0.937	0.103	0.717	0.153	0.959	0.051	0.565	0.313	5.4	0.4	2.0	103.8
BorderSMOTE+Tomek	0.808	0.110	0.731	0.158	0.936	0.110	0.688	0.164	0.957	0.055	0.506	0.322	5.4	0.4	2.3	103.5
BorderSMOTE+OneSided	0.803	0.124	0.765	0.169	0.882	0.153	0.682	0.178	0.932	0.076	0.527	0.330	5.1	0.7	1.9	103.9
BorderSMOTE+NeighClean	0.805	0.105	0.728	0.147	0.931	0.118	0.680	0.154	0.954	0.059	0.503	0.304	5.4	0.4	2.3	103.5
ADASYN+Tomek	0.812	0.103	0.716	0.144	0.965	0.088	0.693	0.152	0.970	0.044	0.498	0.324	5.6	0.2	2.5	103.3
ADASYN+OneSided	0.818	0.120	0.742	0.149	0.937	0.134	0.703	0.172	0.959	0.068	0.544	0.317	5.4	0.4	2.1	103.7
ADASYN+NeighClean	0.789	0.108	0.682	0.146	0.966	0.085	0.661	0.155	0.969	0.043	0.421	0.347	5.6	0.2	3.0	102.8

Table 15
The effects of several oversampling and undersampling combinations on the performance of the GaussianProc, tested on the log transformed + RFE dataset.

On log transformed + RFE dataset

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
basic_classifier	0.839	0.114	0.801	0.150	0.903	0.123	0.732	0.178	0.944	0.062	0.622	0.286	5.2	0.6	1.5	104.3
SMOTE	0.688	0.097	0.551	0.114	0.940	0.104	0.524	0.125	0.948	0.052	0.066	0.363	5.5	0.3	4.8	101.0
BorderSMOTE	0.723	0.097	0.598	0.121	0.940	0.102	0.567	0.129	0.951	0.051	0.209	0.350	5.5	0.3	4.0	101.8
ADASYN	0.695	0.094	0.561	0.113	0.940	0.102	0.532	0.123	0.948	0.050	0.098	0.355	5.5	0.3	4.6	101.2
SMOTE+Tomek	0.694	0.097	0.559	0.114	0.942	0.098	0.531	0.125	0.949	0.049	0.089	0.371	5.5	0.3	4.6	101.2
SMOTE+OneSided	0.727	0.104	0.611	0.140	0.931	0.109	0.573	0.140	0.948	0.054	0.229	0.356	5.4	0.4	3.8	102.0
SMOTE+NeighClean	0.690	0.099	0.553	0.113	0.940	0.104	0.526	0.126	0.948	0.053	0.076	0.367	5.5	0.3	4.7	101.1
BorderSMOTE+Tomek	0.724	0.099	0.601	0.127	0.940	0.102	0.569	0.132	0.951	0.051	0.212	0.355	5.5	0.3	4.0	101.8
BorderSMOTE+OneSided	0.739	0.104	0.631	0.137	0.926	0.116	0.588	0.143	0.946	0.057	0.278	0.362	5.4	0.4	3.5	102.3
BorderSMOTE+NeighClean	0.719	0.101	0.596	0.127	0.934	0.108	0.561	0.134	0.948	0.054	0.197	0.357	5.4	0.4	4.0	101.8
ADASYN+Tomek	0.695	0.095	0.561	0.118	0.942	0.098	0.532	0.122	0.949	0.049	0.092	0.365	5.5	0.3	4.6	101.2
ADASYN+OneSided	0.708	0.098	0.579	0.120	0.939	0.102	0.548	0.129	0.949	0.051	0.149	0.367	5.5	0.3	4.3	101.5
ADASYN+NeighClean	0.673	0.102	0.534	0.126	0.950	0.096	0.510	0.125	0.950	0.047	−0.028	0.450	5.5	0.3	5.3	100.5

Table 16
The effects of several oversampling and undersampling combinations on the performance of AdaBoost.



On log transformed + RFE dataset

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
basic_classifier	0.815	0.145	0.943	0.112	0.740	0.191	0.717	0.195	0.869	0.096	0.674	0.230	4.3	1.5	0.3	105.5
SMOTE	0.831	0.131	0.912	0.117	0.789	0.183	0.731	0.183	0.892	0.091	0.683	0.220	4.6	1.2	0.5	105.3
BorderSMOTE	0.826	0.130	0.918	0.115	0.776	0.181	0.725	0.181	0.886	0.090	0.677	0.218	4.5	1.3	0.5	105.3
ADASYN	0.832	0.128	0.914	0.112	0.786	0.177	0.732	0.181	0.891	0.088	0.685	0.217	4.6	1.2	0.5	105.3
SMOTE+Tomek	0.833	0.133	0.915	0.123	0.791	0.178	0.736	0.186	0.893	0.089	0.685	0.231	4.6	1.2	0.5	105.3
SMOTE+OneSided	0.775	0.155	0.929	0.104	0.700	0.209	0.663	0.190	0.848	0.104	0.612	0.217	4.1	1.7	0.4	105.4
SMOTE+NeighClean	0.829	0.131	0.909	0.117	0.788	0.182	0.728	0.186	0.891	0.091	0.679	0.222	4.6	1.2	0.5	105.3
BorderSMOTE+Tomek	0.829	0.123	0.920	0.112	0.776	0.173	0.727	0.175	0.886	0.086	0.681	0.208	4.5	1.3	0.5	105.3
BorderSMOTE+OneSided	0.788	0.164	0.933	0.103	0.720	0.218	0.684	0.195	0.858	0.108	0.636	0.221	4.2	1.6	0.4	105.4
BorderSMOTE+NeighClean	0.837	0.134	0.929	0.108	0.783	0.184	0.743	0.189	0.890	0.092	0.700	0.221	4.5	1.2	0.4	105.4
ADASYN+Tomek	0.825	0.124	0.926	0.105	0.769	0.180	0.724	0.172	0.883	0.090	0.680	0.202	4.5	1.3	0.4	105.4
ADASYN+OneSided	0.810	0.134	0.912	0.120	0.753	0.183	0.702	0.185	0.874	0.091	0.650	0.223	4.4	1.4	0.5	105.3
ADASYN+NeighClean	0.827	0.132	0.898	0.124	0.795	0.185	0.724	0.183	0.894	0.092	0.670	0.226	4.6	1.2	0.6	105.2

other combinations provided no significant differences in performance.

4.2.5. Effect of cost-sensitive learning on several classifiers

Several classifiers have a built-in option to use cost-sensitive learning. These classifiers can use the scale of imbalance within a dataset to provide different penalties/rewards for the correct/incorrect predictions of a sample's class. In this test, these methods were tested on the log-transformed + RFE (LR version) dataset and compared to their basic, non-cost-sensitive version.

The cost-sensitive versions of the classifiers LR (solver='liblinear', class_weight='balanced', max_iter=500), SVM((gamma='scale', class_weight='balanced')), RandomForrest (class_weight='balanced', n_estimators=1000), DecisionTree (class_weight='balanced'), ExtraTrees (class_weight='balanced', n_estimators=1000), and XGBoost (booster='gblinear', scale_pos_weight=81.5) were compared to the performance of the basic LR classifier (solver='liblinear', max_iter=500). The classifier's training and validation was performed using crossvalidation with the RepeatedStratifiedKFold method (n_splits=10, n_repeats=10, random_state=1).

For all classifiers, except for XGBoost, the cost-sensitive learning option was set with the hyperparameter *class_weight='balanced'*. For XGBoost, the cost-sensitive learning option was set with the hyperparameter *scale_pos_weight=81.5*. This optimal hyperparameter value was determined by performing a grid search with the function xgboost_grid_search, for which the strategy mentioned at https://machinelearningmastery.com/xgboost-for-imbalanced-classification was used.

The mean F1 scores with their standard deviations and medians were plotted in a box–whisker plot, as shown in Fig. 15. Table 17 shows the performance of several classifiers with cost-sensitive learning, and their basic, non-cost-sensitive versions, tested on the log-transformed + RFE dataset. Table 18 shows

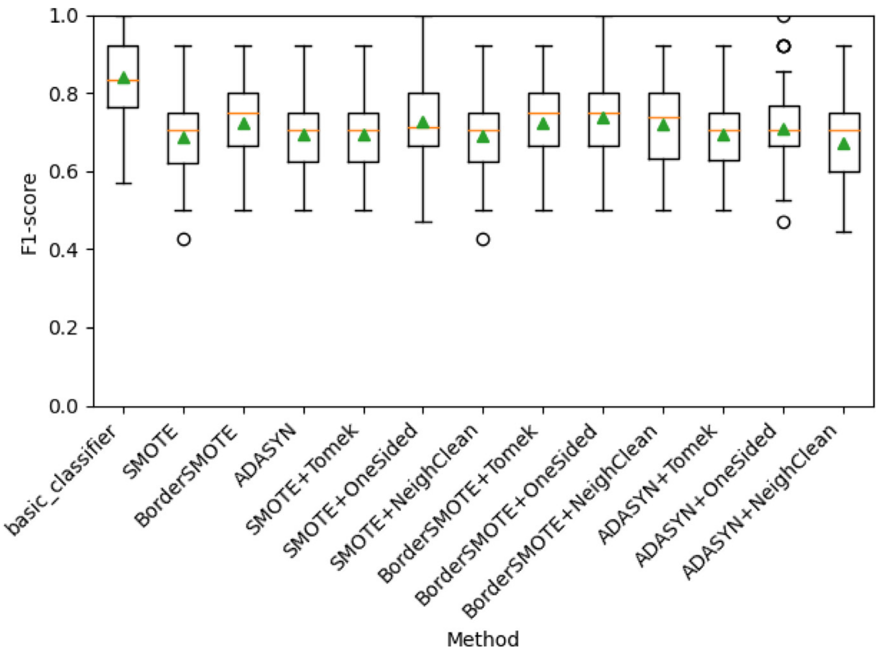


Fig. 13. The effects of several oversampling and undersampling combinations on the GaussianProc performance (F1-score), tested on the log transformed + RFE dataset.

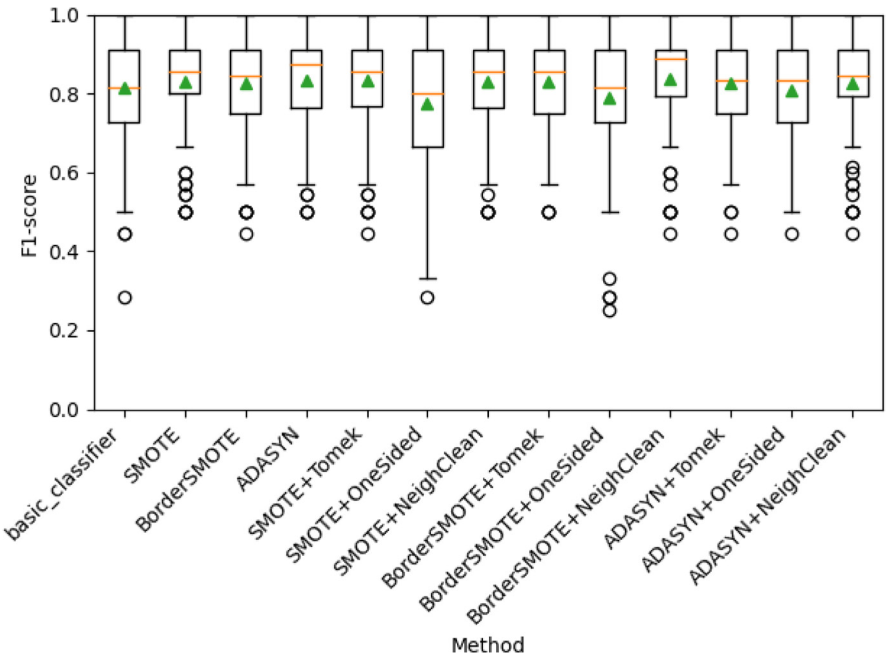


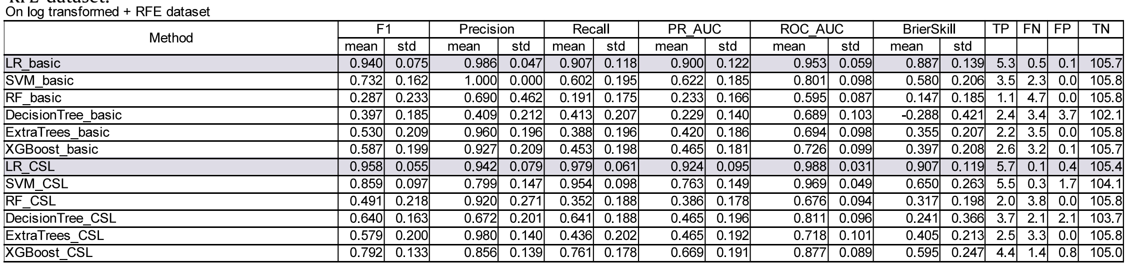
Fig. 14. The effects of several oversampling and undersampling combinations on the AdaBoost classifier performance (F1-score), tested on the log transformed + RFE dataset.

the performance of several classifiers with cost-sensitive learning, and their basic, non-cost-sensitive versions, tested on the normalized + log-transformed + RFE dataset.

Since again a Shapiro–Wilk test showed that the samples were non-Gaussian, a Wilcoxon's Signed Rank test was performed on the F1-scores produced by these cost-sensitive classifiers and their basic, non-cost-sensitive versions, on both the log transformed+ RFE dataset, and the normalized+log transformed dataset. Wilcoxon's signed-rank test was performed on the F1-measures of several cost-sensitive classifiers (CSL), and their basic, non-cost-sensitive versions (basic), when tested on the log transformed+RFE dataset or the normalized+ log transformed+ RFE dataset (n_l_rfe).

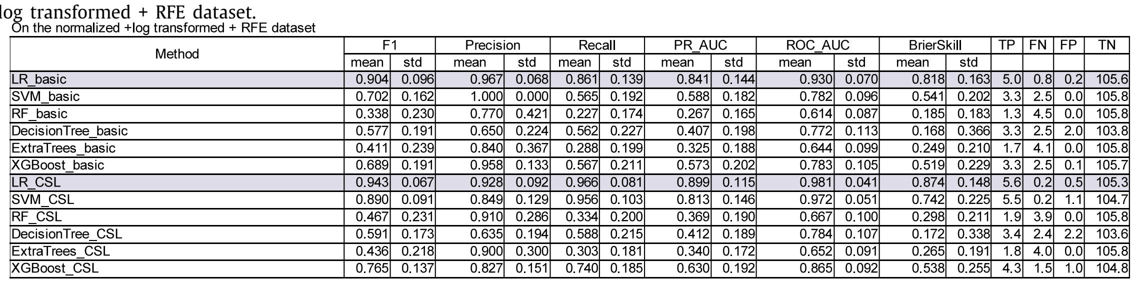
These results showed that LR remains the best-performing classifier and that its cost-sensitive version provided significantly better performance than its basic classifier. In all cases, the cost-sensitive version of a classifier provided better performance than its basic version. Using the log-transformed + RFE dataset provided significantly better performance for the LR, SVM, and ExtraTrees classifiers. However, the normalized + log-transformed + RFE dataset turns out to provide significantly better performance for the DecisionTree and XGBoost classifier. RandomForest is the

Table 17
The performance of several classifiers with cost-sensitive learning, and their basic, non-cost-sensitive versions, tested on the log transformed + RFE dataset.


On log transformed + RFE dataset

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR_basic	0.940	0.075	0.986	0.047	0.907	0.118	0.900	0.122	0.953	0.059	0.887	0.139	5.3	0.5	0.1	105.7
SVM_basic	0.732	0.162	1.000	0.000	0.602	0.195	0.622	0.185	0.801	0.098	0.580	0.206	3.5	2.3	0.0	105.8
RF_basic	0.287	0.233	0.690	0.462	0.191	0.175	0.233	0.166	0.595	0.087	0.147	0.185	1.1	4.7	0.0	105.8
DecisionTree_basic	0.397	0.185	0.409	0.212	0.413	0.207	0.229	0.140	0.689	0.103	-0.288	0.421	2.4	3.4	3.7	102.1
ExtraTrees_basic	0.530	0.209	0.960	0.196	0.388	0.196	0.420	0.186	0.694	0.098	0.355	0.207	2.2	3.5	0.0	105.8
XGBoost_basic	0.587	0.199	0.927	0.209	0.453	0.198	0.465	0.181	0.726	0.099	0.397	0.208	2.6	3.2	0.1	105.7
LR_CSL	0.958	0.055	0.942	0.079	0.979	0.061	0.924	0.095	0.988	0.031	0.907	0.119	5.7	0.1	0.4	105.4
SVM_CSL	0.859	0.097	0.799	0.147	0.954	0.098	0.763	0.149	0.969	0.049	0.650	0.263	5.5	0.3	1.7	104.1
RF_CSL	0.491	0.218	0.920	0.271	0.352	0.188	0.386	0.178	0.676	0.094	0.317	0.198	2.0	3.8	0.0	105.8
DecisionTree_CSL	0.640	0.163	0.672	0.201	0.641	0.188	0.465	0.196	0.811	0.096	0.241	0.366	3.7	2.1	2.1	103.7
ExtraTrees_CSL	0.579	0.200	0.980	0.140	0.436	0.202	0.465	0.192	0.718	0.101	0.405	0.213	2.5	3.3	0.0	105.8
XGBoost_CSL	0.792	0.133	0.856	0.139	0.761	0.178	0.669	0.191	0.877	0.089	0.595	0.247	4.4	1.4	0.8	105.0

Table 18
The performance of several classifiers with cost-sensitive learning, and their basic, non-cost-sensitive versions, tested on the normalized + log transformed + RFE dataset.


On the normalized +log transformed + RFE dataset

Method	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
LR_basic	0.904	0.096	0.967	0.068	0.861	0.139	0.841	0.144	0.930	0.070	0.818	0.163	5.0	0.8	0.2	105.6
SVM_basic	0.702	0.162	1.000	0.000	0.565	0.192	0.588	0.182	0.782	0.096	0.541	0.202	3.3	2.5	0.0	105.8
RF_basic	0.338	0.230	0.770	0.421	0.227	0.174	0.267	0.165	0.614	0.087	0.185	0.183	1.3	4.5	0.0	105.8
DecisionTree_basic	0.577	0.191	0.650	0.224	0.562	0.227	0.407	0.198	0.772	0.113	0.168	0.366	3.3	2.5	2.0	103.8
ExtraTrees_basic	0.411	0.239	0.840	0.367	0.288	0.199	0.325	0.188	0.644	0.099	0.249	0.210	1.7	4.1	0.0	105.8
XGBoost_basic	0.689	0.191	0.958	0.133	0.567	0.211	0.573	0.202	0.783	0.105	0.519	0.229	3.3	2.5	0.1	105.7
LR_CSL	0.943	0.067	0.928	0.092	0.966	0.081	0.899	0.115	0.981	0.041	0.874	0.148	5.6	0.2	0.5	105.3
SVM_CSL	0.890	0.091	0.849	0.129	0.956	0.103	0.813	0.146	0.972	0.051	0.742	0.225	5.5	0.2	1.1	104.7
RF_CSL	0.467	0.231	0.910	0.286	0.334	0.200	0.369	0.190	0.667	0.100	0.298	0.211	1.9	3.9	0.0	105.8
DecisionTree_CSL	0.591	0.173	0.635	0.194	0.588	0.215	0.412	0.189	0.784	0.107	0.172	0.338	3.4	2.4	2.2	103.6
ExtraTrees_CSL	0.436	0.218	0.900	0.300	0.303	0.181	0.340	0.172	0.652	0.091	0.265	0.191	1.8	4.0	0.0	105.8
XGBoost_CSL	0.765	0.137	0.827	0.151	0.740	0.185	0.630	0.192	0.865	0.092	0.538	0.255	4.3	1.5	1.0	104.8

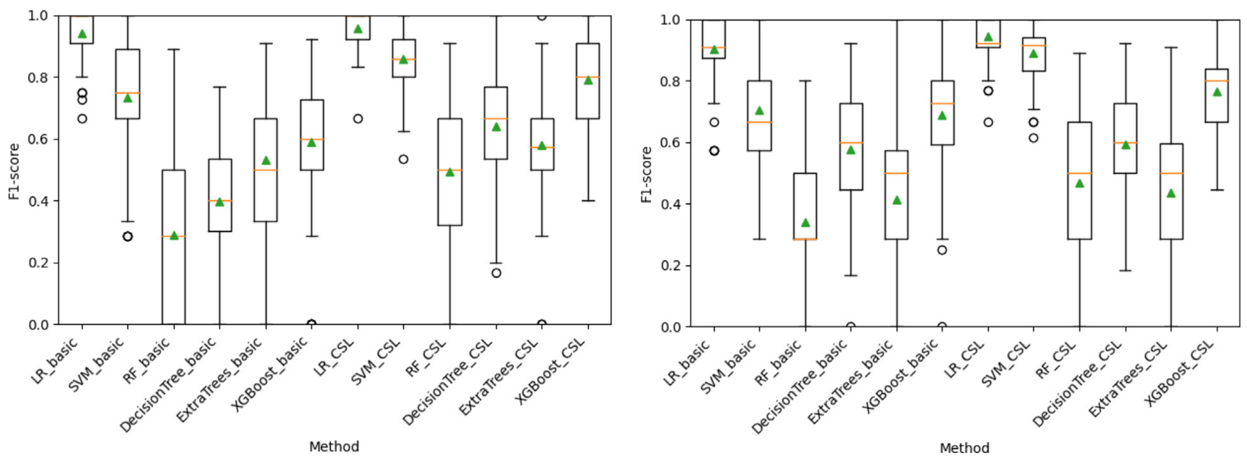


Fig. 15. The F1 performance measure of several classifiers with cost-sensitive learning, and their basic, non-cost-sensitive versions, tested on the log transformed + RFE dataset (left) and the normalized + log transformed + RFE dataset (right).

only classifier found to provide no significantly different performance between tests on the log transformed+RFE dataset and the normalized+log transformed+RFE dataset.

4.3. Investigating extra approaches to improve the performance of the second-best classifiers

LR was so far found to be the best performing classifier. However, the lesser performing classifiers, LDA, GaussianProc, and AdaBoost, could possibly still be improved.

4.3.1. Hyperparameter tuning for LinearDiscriminantAnalysis

As described by Brownlee (https://machinelearningmastery.com/linear-discriminant-analysis-with-python), LDA's performance can possibly be improved by performing a grid search on its 'solver' hyperparameter. As variations, the solvers 'svd' (Singular Value Decomposition), 'lsqr' (least-squares), and 'eigen' were tested. A RepeatedStratifiedKFold cross-validation approach was used (10 repeats, 10 folds) using accuracy for scoring, and the log transformed+RFE dataset as input. The 'svd' solver was found to provide the best performance with a mean accuracy of 0.988.

The test was repeated using an F1 score for scoring, which resulted in the selection of the same solver, but with a mean F1 score of 0.872. Test of this new LDA(solver='svd') setting with all combinations of over/undersampling, using the log transformed+RFE dataset resulted in a mean F1-score of 0.872 (sd 0.097). Adding an oversampler/undersampler provided significantly poorer performance in all cases, as such, these results are not shown.

4.3.2. Hyperparameter tuning for GaussianProcessClassifier

As described by Brownlee (https://machinelearningmastery.com/gaussian-processes-for-classification-with-python), GaussianProc'sperformance can possibly be improved by performing a grid search on the 'kernel' hyperparameter. As variations, the kernels RBF, DotProduct, Matern, RationalQuadratic, and WhiteKernel were tested. A RepeatedStratifiedKFold cross-validation approach was used (10 repeats, 10 folds) using accuracy for scoring and using the log transformed+RFE dataset as

Table 19
The performance of GaussianProc(kernel=1*DotProduct(sigma_0=1) with several combinations of oversampling and undersampling on the log transformed + RFE dataset.

Log transformed + RFE dataset

	F1		Precision		Recall		PR_AUC		ROC_AUC		BrierSkill		TP	FN	FP	TN
Method	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std				
GPC	0.934	0.078	0.984	0.051	0.898	0.121	0.889	0.126	0.948	0.060	0.875	0.143	5.2	0.6	0.1	105.7
+SMOTE	0.965	0.062	0.974	0.063	0.961	0.085	0.939	0.104	0.980	0.043	0.928	0.124	5.6	0.2	0.2	105.6
+BorderSMOTE	0.962	0.059	0.969	0.062	0.959	0.083	0.932	0.102	0.979	0.042	0.921	0.119	5.6	0.2	0.2	105.6
+ADASYN	0.962	0.061	0.970	0.067	0.961	0.083	0.934	0.103	0.979	0.042	0.922	0.124	5.6	0.2	0.2	105.6
+SMOTE+Tomek	0.963	0.062	0.969	0.065	0.963	0.085	0.936	0.105	0.980	0.043	0.924	0.124	5.6	0.2	0.2	105.6
+SMOTE+OneSided	0.965	0.062	0.978	0.060	0.957	0.087	0.939	0.104	0.978	0.044	0.929	0.124	5.5	0.2	0.1	105.7
+SMOTE+NeighClean	0.966	0.059	0.973	0.060	0.964	0.081	0.941	0.101	0.981	0.041	0.931	0.118	5.6	0.2	0.2	105.6
+BorderSMOTE+Tomek	0.963	0.056	0.969	0.062	0.961	0.079	0.934	0.098	0.980	0.040	0.923	0.114	5.6	0.2	0.2	105.6
+BorderSMOTE+OneSided	0.952	0.065	0.970	0.062	0.941	0.097	0.916	0.109	0.969	0.048	0.903	0.126	5.5	0.3	0.2	105.6
+BorderSMOTE+NeighClean	0.959	0.064	0.969	0.062	0.957	0.093	0.930	0.105	0.977	0.047	0.918	0.122	5.5	0.2	0.2	105.6
+ADASYN+Tomek	0.964	0.060	0.971	0.065	0.963	0.082	0.937	0.101	0.980	0.041	0.926	0.121	5.6	0.2	0.2	105.6
+ADASYN+OneSided	0.955	0.068	0.967	0.067	0.950	0.095	0.922	0.113	0.974	0.048	0.909	0.133	5.5	0.3	0.2	105.6
+ADASYN+NeighClean	0.947	0.072	0.938	0.101	0.965	0.080	0.907	0.119	0.980	0.040	0.881	0.167	5.6	0.2	0.5	105.3

input. The DotProduct(sigma_0=1) kernel was found to provide the best performance with a mean accuracy of 0.994. Test of this new GaussianProc(kernel=1*DotProduct(sigma_0=1)) setting with all combinations of over/undersampling using the log transformed+RFE dataset resulted in the following scores shown in Table 19.

These results show that GaussianProc performed best on the log transformed+RFE dataset, in combination with either the oversamplers SMOTE, SMOTE+NeigborCleaningRule, Borderline-SMOTE, Borderline-SMOTE+Tomek Links, ADASYN, or ADASYN+ Tomek Links.

A Wilcoxon's signed-rank test (i.e., samples were shown to be non-Gaussian in a Shapiro–Wilkes test) showed that these combinations were significantly different. The addition of an undersampler did not provide any further improvement.

4.3.3. Hyperparameter tuning for AdaBoost

As Brownlee mentions (https://machinelearningmastery.com/adaboost-ensemble-in-python), the AdaBoost hyperparameters learning_rate and n_estimators are good candidates for improving the classifier's performance. As optimal hyperparameters the settings AdaBoostClassifier(base_estimator =LogisticRegression(solver='liblinear'), learning_rate=1.4, n_estimators=50) was found, resulting in a mean F1-score of 0.897 (sd 0.108). As second best settings AdaBoostClassifier(learning_rate=1.1, n_estimators=1000) was found, resulting in a mean F1-score of 0.756 (sd 0.155). Both versions were tested elaborately, however, the performance could not be further improved, making it the fourth-best classifier. For this reason, the classifier was not further investigated.

4.4. Testing the best performing classifiers with their best resampling method(s) on the test dataset

The best performing classifiers (in order of performance) were identified to be LR, GaussianProc, and LDA. In this experiment, these classifiers were used to predict the classes of the test dataset samples to determine the classifiers' performances on samples it has never seen before.

The classifiers were first trained on the log-transformed + RFE dataset. LR training was performed with the solver='liblinear', max_iter=1000, with the resampling combinations SMOTE (with and without NeighborCleaningRule), Borderline-SMOTE (with and without Tomek Links), and ADASYN (with and without Tomek Links). The cost-sensitive version was trained without resampling. GaussianProc training was performed with the hyperparameter setting kernel=DotProduct(sigma_0=1), with the oversamplers SMOTE, Borderline-SMOTE, and ADASYN. LDA training was performed with the hyperparameter setting solver='svd', without the use of a resampler. After the training phase, the test dataset was log-transformed and the same features were selected as those selected during the RFE step in the training dataset. The class labels were predicted for all samples in the test dataset, and an overview table was produced, containing the predicted classes and the true class for each sample, and the confusion matrix values for each classifier. For a complete overview, the Appendix includes additional tables. Table 20 shows the TP/FN/FP/TN results of the LR, GaussianProc, and LDA with several over/undersamplers, when tested on the test dataset.

Of the tested combinations with LR as the classifier, the use of the oversampler Borderline-SMOTE provided the best results, with all positive minority samples correctly predicted. In most tests with an oversampler, two false-positive samples were found. Two false positives were found to be the lowest number, but if the predictions were repeated, SMOTE, Borderline-SMOTE, and Borderline-SMOTE+Tomek Links were seen to fluctuate between two and three false positives. The default LR classifier, trained without the use of resampling methods also performs remarkably well, and even outperformed the cost-sensitive learning LR version, with only one false negative sample (Sample_032), and one false positive (Sample_022). GaussianProc with and without the use of an over/undersampler provided identical results as LR. LDA, which was tested without an over/undersampler, also provided one false positive and one false negative sample.

The two consistently False Positives were identified as Sample_022 and Sample_045. The third frequently found False Positive was Sample_033. The appendix contains an additional table for this purpose. Checking these samples in the description files showed that for Sample_022, a comment was made that the sample was remarkably dirty. For Sample_045, no stored remarks were found in the description files. For Sample_033, the remark is that a blank sample was found, which should clearly be a negative sample. Sample_032 was found to be a false negative in three predictions (basic LR, GPC(DotProduct), and LDA (svd)) while being identified correctly in all other cases. Since this sample is described as treated with the hormone Stanozolol, the false-negative predictions are likely to be indeed false negatives.

5. Discussion

5.1. General discussion

The LC-HRMS dataset used in this study, which is both imbalanced, contains a large number of features, and a large number of missing data points is rather unique compared to datasets used in literature (ref). Our strategy to replace the missing data points with random values close to the detection threshold of the LC–MS (10.000 +/- 5.000) worked well. However, combined with the imbalance, and a large number of features in our dataset, the traits of our dataset seem to complicate the learning skill of many machine learning algorithms. In hindsight, choosing random values with a larger difference from the 'real' data, and with less variation, might have made the dataset less noisy, and simplified the situation for the classifiers' learning skills.

Table 20
Predicted numbers of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN) by LR, GaussianProc, and LDA with several over/ undersamplers, when tested on the test dataset.

Predictions with Logistic Regression

	LR	+SMOTE	+SMOTE +NCR	+BSMOTE	+BSMOTE +Tomek	+ADASYN	+ADASYN +Tomek
TP	6	7	7	7	7	7	7
FN	1	0	0	0	0	0	0
FP	1	3	3	3	2	3	3
TN	117	115	115	115	116	115	115

Predictions with Gaussian Process Classifier and Linear Discriminant Analysis

	GPC (DotProduct)	+SMOTE	+BSMOTE	+ADASYN	LDA (svd)	y_true
TP	6	7	7	7	6	7
FN	1	0	0	0	1	0
FP	1	3	2	3	1	0
TN	117	115	116	115	117	118

Our findings have confirmed that the F1-score, Precision, Recall, PR_AUC, and to some degree the BrierSkill score are suitable measures for comparison of the classifier's performances when dealing with an imbalanced dataset. However, The ROC_AUC is found to give too positive impressions and is, therefore, an unsuitable measure for imbalanced datasets. Since the ROC_AUC is a frequently used measure in other research though, it was decided to include this measure in our overviews.

Although it is not common practice to include the confusion matrix values in a performance measure overview, we have found it quite informative to include, since it provides the viewer valuable insight into why measures are changing. Test of the raw dataset on the set of different classifiers only showed above no-skill performance for LR, XGBoost, and AdaBoost. All other classifiers either showed around no-skill performance or were not able to produce an output at all.

Our findings showed that the log transformation of our dataset was crucial for the performance of many classifiers. The classifiers LR, LDA, GaussianProc, and AdaBoost were found to provide the best performance. Although several other classifiers also showed above-skill improved performance after log transformation, they did not perform as well as the above-mentioned classifiers.

Although a claim was found that normalization of the data is a necessity [8], in our case, normalization did not provide a significant improvement over log transformation alone for our best-performing classifiers. Since the features in our dataset were shown to be far from Gaussian distribution, with our random values for the missing data points being a large cause for this, it made sense that standardization of the dataset did not work. Special scaling methods like MaxAbsScaler to deal with sparse datasets do exist, however, were beyond the scope of this research.

Of the tested feature removal methods, Recursive Feature Elimination was found to be the best performing method, when used with the LR classifier for feature importance determination and log transformation of the dataset. The Recursive Feature Elimination was also attempted with LDA for feature importance determination, and with normalization + log transformation of the dataset, but these methods were found to give a lesser performance for most classifiers.

The Recursive Feature Removal with LR for feature importance determination was performed with the hyperparameter setting class_weights='balanced' to stimulate the selection of features linked to the minority class. Although the cost-sensitive version of LR was found to give significantly better performance than its basic version, the basic LR classifier was found to perform quite well. In retrospect, it may not have been strictly necessary to use its cost-sensitive version for the Recursive Feature Elimination step. Options of using an oversampler before Recursive Feature Elimination, so that other classifiers without cost-sensitive learning could be used were not investigated during this research.

Alternative variance-based feature selection methods like Low Variance and Multicollinear feature selection did not work well for our dataset. Since we were working with an imbalanced dataset, where the missing data points, replaced by random values, formed the majority within most features, the minority class-related features were expected to be low in variance. It, therefore, makes sense that the Low Variance method was not suitable for our dataset. A good explanation of why the Multicollinear Feature Elimination method did not work well was not found. Possibly the replacement of the missing data points by random values caused the dataset to only give the impression that a multitude of features was highly correlated, resulting in a discard of most of the features when Multicollinear Feature Elimination was performed.

Finally, the Feature Removal with SelectKBest was tested in combination with Borderline-SMOTE to create a more balanced dataset. It was noted that the more features were removed with this method, the lower the performance of the LR classifier became, indicating that this method is not capable of selecting the important features within our dataset. The best results were found when 250 features were kept out of the total of 271 features in our original dataset, meaning that 21 features were discarded using this method. However, these features could also be discarded by only using the Recursive Feature Elimination method, making the SelectKBest-based method no improvement.

Repeated stratified k-fold cross-validation is commonly performed with 10 folds [26] and a varying number of repeats. We chose to follow this example by performing all cross-validation steps with 10 folds and 10 repeats. With our low number of 65 minority class samples, in retrospect, it might have been a better choice to decrease the number of folds, and increase the number of repeats, to create a test situation where misclassifications of some minority samples have less effect on the variance.

Our tests showed that the use of the oversamplers SMOTE, Borderline-SMOTE, and ADASYN on our dataset significantly improved the performance of LR, GaussianProc, and AdaBoost. The addition of an undersampler (Tomek Links, Edited Nearest Neighbor Rule, One-Sided Selection, or Neighborhood Cleaning Rule) repeatedly gave slightly, but not a significant improvement. Edited Nearest Neighbor Rule was the only undersampler consistently giving significantly poorer results. In our case, the performance of the classifiers LDA worsened when combined with a resampling method. Using an undersampler before an oversampler, in general, provided a poorer performance on the classifier except for the Edited Nearest Neighbor Rule.

The classifiers LR, SVM, RandomForest, DecisionTree, Extra Trees, and XGBoost were tested with both their basic setting and their cost-sensitive learning option. All cost-sensitive classifiers were found to provide significantly improved performance over their basic versions. LR was still found to be the best performing classifier with its cost-sensitive version providing the best performance. However, the cost-sensitive LR classifier was not found to outperform the basic LR classifier when combined with SMOTE, Borderline-SMOTE, or ADASYN.

LR was found to be the best performing classifier on our dataset. Although the basic classifier performed quite well without the need for a resampler (F1 score 0.940), the addition of the oversamplers SMOTE or ADASYN was seen to provide significantly best performance (F1-score 0.965), followed by Borderline-SMOTE+Tomek Links (F1-score 0.962) and Borderline-SMOTE (F1-score 0.961). Predictions on the test dataset showed no false negatives when a resampler was used, and one false negative when no resampler was used. The best resampling combinations (SMOTE, ADASYN, and Borderline-SMOTE) showed three false positives, while Borderline-SMOTE showed only two false positives. Why the best performing classifier/resampling combinations showed one more false positive is most likely caused by the (test) dataset. The use of the basic LR classifier in combination with a resampler seemed to provide better performance than using LR's cost-sensitive version.

GaussianProc was found to be the second-best performing classifier although hyperparameter tuning was needed to achieve these results. Best results were found when the hyperparameter setting solver='svd' was used, in combination with the oversampler SMOTE (F1-score 0.965), followed by the addition of the oversamplers Borderline-SMOTE or ADASYN (F1-score 0.962). Predictions on the test dataset showed quite similar results as LR: the classifier without a resampler shows one false positive and one false negative. All combinations with an oversampler showed no false negatives, three false positives when SMOTE or ADASYN was used, and two false positives when Borderline-SMOTE was used.

LDA was found to be the third-best classifier. Here hyperparameter tuning was crucial to achieving the best performance as well. With the hyperparameter setting solver='svd', this classifier was found to perform optimally without the addition of a resampler, resulting in a mean F1 score of 0.872. Predictions with this classifier on the test dataset showed similar results as the non-resampled versions of LR and GaussianProc.

AdaBoost was found to perform reasonably well, but compared with the other classifiers, it was considered not well enough. Hyperparameter tuning was shown to improve performance. Best results were found with the following settings:

- AdaBoost(base_estimator=LogisticRegression (solver='liblinear'), learning_rate=1.4, n_estimators=50), resulting in a mean F1-score of 0.897
- Second best results were found with the settings AdaBoost(learning_rate=1.1, n_estimators=1000), resulting in a mean F1-score of 0.756

All three classifiers trained without resampling showed similar results, showing one false positive (Sample_022) and one false negative (Sample_032). Classifiers trained with the addition of a resampler repeatedly showed Sample_022, Sample_033, and Sample_045 to be false positives. Sample_032 was found to be a false negative in three cases. This sample was claimed to be treated with the hormone Stanozolol. No irregularities in the sample preparation and/or LC–MS analysis were found for these samples, which could explain the incorrect predictions for some resampling/classifier combinations. The false-positive/ negative predictions for these samples are, therefore, indeed considered to be incorrect predictions of our classifiers.

Of the hormone-treated samples in our dataset, it could be guaranteed that they were hormone-treated. However, of the samples that were labeled as being non-hormone-treated, this was only an assumption. The fact that our dataset may have contained hormone-treated samples that were labeled as non-hormone-treated, due to lack of information, may have had negative effects on the training of the classifiers. Outlier removal by use of a one-class classifier like IsolationForest and/or OneClassSVM might offer a solution, but this option could not be fully investigated.

- *RQ1: What is the state of the art in data balancing algorithms, to deal with imbalanced datasets?*
  From our literature search, we conclude that the oversamplers SMOTE, Borderline-SMOTE and ADASYN are the state-of-the-art data balancing methods used to cope with an imbalance in a dataset. RUS was the only undersampler mentioned that was found in our search. Although technically not data balancing methods, feature reduction by Recursive Feature Elimination or a RandomForest approach were mentioned in articles to aid in the reduction of the dataset complexity and thereby improving a classifier's learning skill.
- *RQ 1a: What is the state of the art in dealing with missing data points within a dataset?*
  Brownlee [26] mentions three strategies for dealing with missing data points; discarding of all features containing missing data, imputation of missing data values from neighboring samples, and use of a classifier that can cope with missing data points. To our knowledge, our strategy of replacing the below the LC–MS detection threshold missing data points with artificial random values has never been attempted before.
- *RQ 1b: Which state-of-the-art data balancing algorithms are presently being used for imbalanced datasets derived from LC–MS data?*
  No articles mentioning data balancing methods being used on LC–MS-derived datasets were found during our literature search.
- *RQ2: Which data balancing methods work best for training machine learning algorithms to detect illegal hormone usage in LC–MS analysis results of bovine urine samples?*
  Our findings confirm that the oversamplers SMOTE, Borderline-SMOTE, and ADASYN provide the best performance on our tested classifiers when trained on our LC–MS derived dataset. In some cases, the addition of the undersamplers Tomek Links or Neighbor Cleaning Rule was found to improve a classifier's performance even further. LR was found to be the best-performing classifier, followed by GaussianProc and LDA.
- *RQ2a: Which data imputation strategy can be used for our LC–MS-data derived imbalanced dataset?*
  Since the three missing data strategies mentioned in the answer of RQ-1a were not an option for our specific situation, the replacement of the missing data points with random values close to the detection threshold of the LC–MS instrument was investigated in this research. Our strategy was found to work and resulted in the identification of three promising classifiers for the detection of illegal hormone abuse in the cattle industry.
- *RQ 2b: Which combination of feature selection, data balancing methods, and classification algorithm work best for detecting illegal hormone usage in LC–MS analysis results of bovine urine samples?*
  Our results show that log transformation followed by Recursive Feature Elimination (with LR for feature importance determination) of the dataset was necessary.

5.2. Threats to validity

Construct validity: As measures of a classifier's performance, the measures F1-score, Precision, Recall, PR_AUC, ROC_AUC, Brier-SkillScore, and the (mean) confusion matrix values were chosen. ROC-AUC is frequently mentioned in the literature to give an overly positive impression of a classifier's performance when used on imbalanced datasets, it was nevertheless decided to be included as a measure for comparison purposes with other papers. All other chosen metrics have been proven in other studies to be valid measures.

Although deliberate attempts were made to perform all tests as unprejudiced as possible, include as many alternative approaches as possible, and test a broad selection of classification algorithms, resampling strategies, and feature selection methods, the authors of this paper were aware that by the choice to use LR for several performance improvement attempts, this may have steered the selection of the best performing classifier towards LR. By no means was the selection of strategies a complete list of all classification, resampling, and feature selection methods.

There can be different combinations for ensemble methods so that the performance might be better when ensemble methods are applied.

Internal validity: As can be seen in the Related Work section of this paper, the approach described in this article to reach a well-performing classifier is based on the approach suggested by Brownlee [26] and similar approaches have been described in other articles as well. To our knowledge, the approach to replace below-the detection-threshold missing data points in our dataset with random values has never been attempted before. If the impression of collinearity for many features and the poor learning skill for several classifiers can be explained by our random value, the approach cannot be excluded.

Conclusion validity: We do not claim that the combinations of feature selection, resampling, and classifiers that were found to perform well on our dataset are the only combinations that can work. Other combinations may perform equally well after further hyperparameter tuning, but investigation of all these combinations was beyond the scope of this research. Statistical tests were frequently performed to determine if the effects of a test approach were significant. If variants of a certain approach were not found to be significantly different, in general, the simplest approach was selected for continued experiments.

External validity: It should be noted that the experimental approach described in this paper was performed on a dataset prepared by the last two authors of this article. Similar datasets that combine the traits of being imbalanced, high in a number of features, and high in a number of below-the-threshold missing data points, to our knowledge were not publicly available, and were, therefore beyond the scope of this research.

6. Conclusions

This research aimed to identify a classification model that can discriminate between hormone-treated and non-treated animals. The dataset used for this research was produced by performing an LC-HRMS analysis on a large number of bovine urine samples. Only a small number were from studies whereby animals were treated with hormones, making it an imbalanced dataset. Furthermore, the dataset contained many features and a large number of 'missing data points' that resulted from certain signals not rising above the LC-HRMS instrument's detection threshold in certain samples. To our knowledge, the combination of these dataset traits has never been analyzed with classifiers before. A strategy of replacing the missing data points with random values close to the LC-HRMS instrument's detection threshold was applied. Log transformation followed by Recursive Feature Elimination (with LR for feature importance determination) of the dataset was found to be crucial. Data balancing with SMOTE or ADASYN was found to provide the best performance for LR (max F1-score of 0.965). Data balancing with SMOTE + NeighborCleaningRule was found to provide the best performance for GaussianProc (max F1-score of 0.966). LDA was found not to benefit from data balancing (max F1-score of 0.872). Test of the three classifiers with their preferred resampling strategy on a test dataset provided similar classification results with minimal false positives and false negatives.

Based on the findings of this research, we make the following suggestions for future research: (see Table 21,Table 22,Table 23,Table 24,Table 25a,Table 25b,Table 26b).

1. **Use of alternative random values for the missing data:** For this research, the missing data points were replaced by values close to the detection threshold of the LC–MS instrument. In retrospect, this might have created a rather noisy dataset that complicated the learning skill of many classifiers. Using random values that are separated further from the 'true' data values might improve the learning skill of these classifiers.
2. **Use of pre-data balancing and alternative classifiers for Recursive Feature Elimination:** Recursive Feature Elimination was said to need a classifier that produces either a coef, or a feature_importances_ attribute. Since we were dealing with an imbalanced dataset, it was our assumption that we would need a classifier with cost-sensitive learning to stimulate the selection of minority class-related features. This combination of requirements seriously limited the options for such a classifier. Not only did it turn out that several classifiers performed quite well without cost-sensitive learning or a resampling strategy, but the option of first creating a more balanced dataset by use of an oversampler, followed by Recursive Feature Elimination with alternative (non-cost-sensitive learning) classifiers was not considered during the development of the Recursive Feature Elimination function. It is, therefore, our recommendation to investigate the option to perform data balancing before feature selection.
3. **Removal of outliers by the use of a one-class classifier:** A potential threat in this research was the fact that samples that were labeled as non-hormone-treated could not be guaranteed to be non-hormone-treated. Our tests on the test dataset showed that several samples were consistently predicted to be hormone-treated, however, this prediction could not be confirmed. Furthermore, the presence of incorrectly labeled samples can seriously inhibit the learning skill of many classifiers. A one-class classifier can aid in such cases by identifying and discarding outliers within each class before the dataset is used for the classifiers' training phase. It is, therefore, our recommendation to investigate outlier removal from the majority class samples before resampling and training the classifiers.
4. **Use of alternative settings for StratifiedKFold cross-validation:** During all tests of this research, the StratifiedKFold cross-validation was performed with 10 folds and 10 repeats. With the rather small number of minority class samples in our dataset, in retrospect, it might have been a better choice to decrease the number of folds and increase the number of repeats. This way, the number of minority class samples in each fold would increase, resulting in a lessened impact of possible outliers. It is,

Table 21
Recursive Feature Elimination results of selected and discarded features on the log transformed dataset, using LR for feature importance determination.

Selected features:
[‘C18H22O5S1’, ‘C18H22O6S1’, ‘C18H24O3’, ‘C18H24O4’, ‘C18H24O6S1’, ‘C18H24O8S2’, ‘C18H26O6’, ‘C18H26O9S1’, ‘C18H28O4’, ‘C18H28O6’, ‘C18H30O6S1’, ‘C18H30O8S1’, ‘C18H32O6S1’, ‘C19H26O2’, ‘C19H26O3’, ‘C19H26O5S1’, ‘C19H28O2’, ‘C19H28O3’, ‘C19H30O3’, ‘C19H30O5S1’, ‘C19H32O6S1’, ‘C19H32O8S2’, ‘C20H30O3’, ‘C20H30O4’, ‘C20H30O6’, ‘C20H30O9S1’, ‘C20H32O4’, ‘C20H32O5’, ‘C20H32O6S1’, ‘C20H32O7S1’, ‘C20H32O8S1’, ‘C20H34O7S1’, ‘C20H34O9S1’, ‘C20H36O5’, ‘C21H28O6’, ‘C21H30O12S2’, ‘C21H30O2’, ‘C21H30O8S1’, ‘C21H30O9S1’, ‘C21H32O11S2’, ‘C21H32O2’, ‘C21H32O5S1’, ‘C21H32O6’, ‘C21H32O7S1’, ‘C21H32O9S2’, ‘C21H34O3’, ‘C21H34O4’, ‘C21H34O5’, ‘C21H34O5S1’, ‘C21H34O6S1’, ‘C21H36O5S1’, ‘C21H36O9S2’, ‘C23H38O4’, ‘C23H38O7S1’, ‘C24H30O8’, ‘C24H30O9’, ‘C24H32O10’, ‘C24H32O11S1’, ‘C24H32O9’, ‘C24H34O12’, ‘C24H34O5’, ‘C24H36O10’, ‘C24H36O12’, ‘C24H36O13’, ‘C24H38O12’, ‘C24H38O4’, ‘C24H40O4’, ‘C24H40O5’, ‘C24H40O6S1’, ‘C24H40O8S1’, ‘C25H32O9’, ‘C25H34O8’, ‘C25H36O12S1’, ‘C25H38O11S1’, ‘C25H38O12S1’, ‘C25H40O11S1’, ‘C26H36O12’, ‘C26H38O9’, ‘C26H40O10’, ‘C26H40O11’, ‘C26H40O9’, ‘C26H42O10’, ‘C26H42O12’, ‘C26H43N1O6’, ‘C26H43N1O8S1’, ‘C26H43N1O9S1’, ‘C26H44O11’, ‘C26H45N1O5S1’, ‘C26H45N1O6S1’, ‘C26H45N1O8S2’, ‘C26H45N1O9S2’, ‘C27H36O14S1’, ‘C27H38O10’, ‘C27H38O12’, ‘C27H38O13S1’, ‘C27H38O14S1’, ‘C27H38O8’, ‘C27H38O9’, ‘C27H40O10’, ‘C27H40O11’, ‘C27H40O12’, ‘C27H40O12S1’, ‘C27H40O9’, ‘C27H42O10’, ‘C27H42O11’, ‘C27H42O13S1’, ‘C27H44O9’, ‘C30H46O10’, ‘C30H46O11’, ‘C30H47N3O9S1’, ‘C30H48O10’, ‘C33H46O16’, ‘C33H48O17’, ‘C33H50O15’]
Discarded Features:
[’C18H22O2’, ‘C18H22O3’, ‘C18H24O10S2’, ‘C18H24O2’, ‘C18H24O5S1’, ‘C18H24O7S1’, ‘C18H24O9S2’, ‘C18H28O10S1’, ‘C18H28O7’, ‘C18H28O7S1’, ‘C18H28O9S1’, ‘C18H30O3’, ‘C18H30O5’, ‘C18H30O6’, ‘C18H30O9S1’, ‘C18H32O3’, ‘C19H24O3’, ‘C19H24O6S1’, ‘C19H26O6S1’, ‘C19H28O5S1’, ‘C19H28O6S1’, ‘C19H28O9S2’, ‘C19H30O2’, ‘C19H30O6S1’, ‘C19H30O8S2’, ‘C19H30O9S2’, ‘C19H32O2’, ‘C19H32O3’, ‘C19H32O5S1’, ‘C19H32O9S2’, ‘C20H28O6’, ‘C20H28O9S1’, ‘C20H30O5’, ‘C20H30O6S1’, ‘C20H30O7S1’, ‘C20H30O8S1’, ‘C20H32O3’, ‘C20H34O10S1’, ‘C20H34O4’, ‘C20H34O5’, ‘C20H34O6’, ‘C20H34O7’, ‘C20H34O8S1’, ‘C20H36O8S1’, ‘C21H28O11S2’, ‘C21H28O5’, ‘C21H28O8S1’, ‘C21H28O9S1’, ‘C21H30O10S2’, ‘C21H30O11S2’, ‘C21H30O3’, ‘C21H30O4’, ‘C21H30O5’, ‘C21H30O5S1’, ‘C21H30O6’, ‘C21H30O6S1’, ‘C21H30O7S1’, ‘C21H32O10S2’, ‘C21H32O12S2’, ‘C21H32O3’, ‘C21H32O4’, ‘C21H32O5’, ‘C21H32O6S1’, ‘C21H32O8S1’, ‘C21H32O9S1’, ‘C21H34O10S2’, ‘C21H34O11S2’, ‘C21H34O2’, ‘C21H34O7S1’, ‘C21H34O8S1’, ‘C21H34O9S2’, ‘C21H36O2’, ‘C21H36O3’, ‘C21H36O6S1’, ‘C23H37N1O5S1’, ‘C23H37N1O8S2’, ‘C24H32O12S1’, ‘C24H32O13S1’, ‘C24H32O8’, ‘C24H34O8S1’, ‘C24H38O11’, ‘C24H38O5’, ‘C24H38O7S1’, ‘C24H38O8S1’, ‘C24H38O9’, ‘C24H40O3’, ‘C24H40O7S1’, ‘C24H40O9’, ‘C25H34O9’, ‘C25H36O8’, ‘C25H36O9’, ‘C25H38O8’, ‘C25H38O9’, ‘C25H39N1O6S1’, ‘C25H39N1O9S2’, ‘C25H40N2O6S1’, ‘C25H40N2O9S2’, ‘C25H40O12S1’, ‘C25H40O8’, ‘C25H40O9’, ‘C26H38O10’, ‘C26H38O11’, ‘C26H38O12’, ‘C26H42O11’, ‘C26H42O13’, ‘C26H43N1O4’, ‘C26H43N1O5’, ‘C26H43N1O7S1’, ‘C26H45N1O10S2’, ‘C26H45N1O7S1’, ‘C27H36O11’, ‘C27H36O12’, ‘C27H36O15S1’, ‘C27H38O11’, ‘C27H38O15S1’, ‘C27H40O13S1’, ‘C27H40O14S1’, ‘C27H40O15S1’, ‘C27H40O8’, ‘C27H42O12S1’, ‘C27H42O14S1’, ‘C27H42O8’, ‘C27H42O9’, ‘C27H44O12S1’, ‘C27H44O8’, ‘C29H45N1O11S1’, ‘C29H46O10’, ‘C30H40O14’, ‘C30H40O15’, ‘C30H40O16’, ‘C30H42O11’, ‘C30H47N3O12S2’, ‘C30H48O11’, ‘C30H48O9’, ‘C31H44O15’, ‘C31H46O14’, ‘C31H46O15’, ‘C31H47N1O12S1’, ‘C31H48N2O12S1’, ‘C31H48O14’, ‘C31H48O15’, ‘C32H51N1O10’, ‘C32H51N1O11’, ‘C32H51N1O12’, ‘C32H53N1O11S1’, ‘C32H53N1O12S1’, ‘C32H53N1O13S1’, ‘C33H44O17’, ‘C33H44O18’, ‘C33H46O17’, ‘C33H46O18’, ‘C33H48O15’, ‘C33H48O16’, ‘C33H48O18’, ‘C33H50O16’, ‘C33H50O17’, ‘C33H52O15’]
Used model for feature importance determination: LogisticRegression() With log transformation. Number of selected features: 114 Number of discarded features: 157

Table 22
Recursive Feature Elimination results of selected and discarded features on the log transformed dataset, using LDA for feature importance determination. Marked in bold: Features that were selected for discard both by LR and LDA.

Selected features:
[’C18H22O2’, ‘C18H22O3’, ‘C18H22O5S1’, ‘C18H22O6S1’, ‘C18H24O10S2’, ‘C18H24O2’, ‘C18H24O3’, ‘C18H24O4’, ‘C18H24O5S1’, ‘C18H24O6S1’, ‘C18H24O8S2’, ‘C18H24O9S2’, ‘C18H26O6’, ‘C18H26O9S1’, ‘C18H28O10S1’, ‘C18H28O4’, ‘C18H28O6’, ‘C18H28O7’, ‘C18H30O5’, ‘C18H30O6S1’, ‘C18H30O8S1’, ‘C18H30O9S1’, ‘C18H32O3’, ‘C18H32O6S1’, ‘C19H24O3’, ‘C19H26O2’, ‘C19H26O3’, ‘C19H26O5S1’, ‘C19H26O6S1’, ‘C19H28O2’, ‘C19H28O3’, ‘C19H28O6S1’, ‘C19H28O9S2’, ‘C19H30O3’, ‘C19H30O5S1’, ‘C19H30O6S1’, ‘C19H30O8S2’, ‘C19H32O2’, ‘C19H32O5S1’, ‘C19H32O6S1’, ‘C19H32O8S2’, ‘C19H32O9S2’, ‘C20H28O6’, ‘C20H28O9S1’, ‘C20H30O3’, ‘C20H30O4’, ‘C20H30O5’, ‘C20H30O6’, ‘C20H30O6S1’, ‘C20H30O9S1’, ‘C20H32O3’, ‘C20H32O4’, ‘C20H32O5’, ‘C20H32O6S1’, ‘C20H32O8S1’, ‘C20H34O4’, ‘C20H34O5’, ‘C20H34O6’, ‘C20H34O7’, ‘C20H34O7S1’, ‘C20H34O8S1’, ‘C20H34O9S1’, ‘C20H36O5’, ‘C20H36O8S1’, ‘C21H28O11S2’, ‘C21H28O5’, ‘C21H28O6’, ‘C21H28O8S1’, ‘C21H30O11S2’, ‘C21H30O12S2’, ‘C21H30O2’, ‘C21H30O3’, ‘C21H30O5’, ‘C21H30O5S1’, ‘C21H30O6S1’, ‘C21H30O8S1’, ‘C21H30O9S1’, ‘C21H32O10S2’, ‘C21H32O11S2’, ‘C21H32O12S2’, ‘C21H32O2’, ‘C21H32O3’, ‘C21H32O4’, ‘C21H32O5’, ‘C21H32O5S1’, ‘C21H32O6’, ‘C21H32O6S1’, ‘C21H32O7S1’, ‘C21H32O8S1’, ‘C21H32O9S1’, ‘C21H32O9S2’, ‘C21H34O10S2’, ‘C21H34O11S2’, ‘C21H34O2’, ‘C21H34O3’, ‘C21H34O4’, ‘C21H34O5S1’, ‘C21H34O6S1’, ‘C21H34O7S1’, ‘C21H34O8S1’, ‘C21H36O2’, ‘C21H36O3’, ‘C21H36O5S1’, ‘C21H36O6S1’, ‘C21H36O9S2’, ‘C23H37N1O5S1’, ‘C23H37N1O8S2’, ‘C23H38O7S1’, ‘C24H30O8’, ‘C24H30O9’, ‘C24H32O10’, ‘C24H32O11S1’, ‘C24H32O12S1’, ‘C24H32O13S1’, ‘C24H32O8’, ‘C24H32O9’, ‘C24H34O12’, ‘C24H34O5’, ‘C24H34O8S1’, ‘C24H36O10’, ‘C24H36O12’, ‘C24H36O13’, ‘C24H38O11’, ‘C24H38O12’, ‘C24H38O5’, ‘C24H38O7S1’, ‘C24H38O8S1’, ‘C24H38O9’, ‘C24H40O4’, ‘C24H40O5’, ‘C24H40O6S1’, ‘C24H40O7S1’, ‘C24H40O8S1’, ‘C24H40O9’, ‘C25H34O8’, ‘C25H34O9’, ‘C25H36O12S1’, ‘C25H36O8’, ‘C25H36O9’, ‘C25H38O11S1’, ‘C25H38O12S1’, ‘C25H38O9’, ‘C25H39N1O6S1’, ‘C25H39N1O9S2’, ‘C25H40N2O6S1’, ‘C25H40O11S1’, ‘C25H40O12S1’, ‘C25H40O8’, ‘C25H40O9’, ‘C26H36O12’, ‘C26H38O10’, ‘C26H38O12’, ‘C26H38O9’, ‘C26H40O10’, ‘C26H40O11’, ‘C26H40O9’, ‘C26H42O10’, ‘C26H42O12’, ‘C26H42O13’, ‘C26H43N1O4’, ‘C26H43N1O6’, ‘C26H43N1O7S1’, ‘C26H43N1O8S1’, ‘C26H43N1O9S1’, ‘C26H44O11’, ‘C26H45N1O10S2’, ‘C26H45N1O5S1’, ‘C26H45N1O6S1’, ‘C26H45N1O7S1’, ‘C26H45N1O8S2’, ‘C26H45N1O9S2’, ‘C27H36O11’, ‘C27H36O14S1’, ‘C27H36O15S1’, ‘C27H38O10’, ‘C27H38O11’, ‘C27H38O12’, ‘C27H38O13S1’, ‘C27H38O14S1’, ‘C27H38O15S1’,

(*continued on next page*)

Table 22 (*continued*).

Selected features:
‘C27H38O8’, ‘C27H38O9’, ‘C27H40O10’, ‘C27H40O11’, ‘C27H40O12’, ‘C27H40O12S1’, ‘C27H40O13S1’, ‘C27H40O14S1’, ‘C27H40O15S1’, ‘C27H40O8’, ‘C27H40O9’, ‘C27H42O11’, ‘C27H42O12S1’, ‘C27H42O13S1’, ‘C27H42O14S1’, ‘C27H42O8’, ‘C27H42O9’, ‘C27H44O12S1’, ‘C27H44O8’, ‘C27H44O9’, ‘C29H45N1O11S1’, ‘C29H46O10’, ‘C30H40O14’, ‘C30H40O15’, ‘C30H46O11’, ‘C30H47N3O9S1’, ‘C30H48O10’, ‘C30H48O11’, ‘C30H48O9’, ‘C31H44O15’, ‘C31H46O14’, ‘C31H46O15’, ‘C31H47N1O12S1’, ‘C31H48N2O12S1’, ‘C31H48O14’, ‘C31H48O15’, ‘C32H51N1O10’, ‘C32H51N1O11’, ‘C32H51N1O12’, ‘C32H53N1O11S1’, ‘C32H53N1O12S1’, ‘C32H53N1O13S1’, ‘C33H44O17’, ‘C33H44O18’, ‘C33H46O16’, ‘C33H46O17’, ‘C33H46O18’, ‘C33H48O15’, ‘C33H48O17’, ‘C33H48O18’, ‘C33H50O15’, ‘C33H50O16’, ‘C33H50O17’, ‘C33H52O15’]
Discarded features:
[’**C18H24O7S1**’, ‘C18H28O7S1’, ‘**C18H28O9S1**’, **’C18H30O3**’, **’C18H30O6**’, ‘**C19H24O6S1**’, **’C19H28O5S1**’, **’C19H30O2**’, **’C19H30O9S2**’, **’C19H32O3**’, **’C20H30O7S1**’, ‘**C20H30O8S1**’, ’C20H32O7S1’, **’C20H34O10S1**’, ‘**C21H28O9S1**’, ‘**C21H30O10S2**’, **’C21H30O4**’, ‘**C21H30O6**’, **’C21H30O7S1**’, ‘C21H34O5’, **’C21H34O9S2**’, ‘C23H38O4’, ‘C24H38O4’, ‘**C24H40O3**’, ‘C25H32O9’, ‘**C25H38O8’**, **’C25H40N2O9S2**’, ‘**C26H38O11**’, ‘**C26H42O11**’, ‘**C26H43N1O5**’, ‘**C27H36O12**’, ‘C27H42O10’, ‘**C30H40O16**’, ‘**C30H42O11**’, ‘C30H46O10’, ‘**C30H47N3O12S2**’, ‘**C33H48O16**’]
Used model for feature importance determination: LinearDiscriminantAnalysis () With log transformation. Number of selected features: 234 Number of discarded features: 37

Table 23
Recursive Feature Elimination results of selected and discarded features on the normalized + log transformed dataset, using LR for feature importance determination.

Selected features:
[’C18H22O2’, ‘C18H22O5S1’, ‘C18H22O6S1’, ‘C18H24O2’, ‘C18H24O4’, ‘C18H24O6S1’, ‘C18H24O8S2’, ‘C18H26O6’, ‘C18H26O9S1’, ‘C18H28O4’, ‘C18H28O6’, ‘C18H28O7S1’, ‘C18H30O6S1’, ‘C18H30O8S1’, ‘C18H32O6S1’, ‘C19H26O2’, ‘C19H26O3’, ‘C19H26O5S1’, ‘C19H28O2’, ‘C19H28O3’, ‘C19H30O3’, ‘C19H30O5S1’, ‘C19H32O6S1’, ‘C20H30O3’, ‘C20H30O4’, ‘C20H30O6’, ‘C20H30O9S1’, ‘C20H32O3’, ‘C20H32O4’, ‘C20H32O5’, ‘C20H32O6S1’, ‘C20H32O7S1’, ‘C20H32O8S1’, ‘C20H34O4’, ‘C20H34O7S1’, ‘C20H34O9S1’, ‘C20H36O5’, ‘C21H28O6’, ‘C21H30O12S2’, ‘C21H30O2’, ‘C21H30O3’, ‘C21H30O8S1’, ‘C21H30O9S1’, ‘C21H32O10S2’, ‘C21H32O11S2’, ‘C21H32O2’, ‘C21H32O5S1’, ‘C21H32O6’, ‘C21H32O7S1’, ‘C21H32O9S2’, ‘C21H34O3’, ‘C21H34O4’, ‘C21H34O5’, ‘C21H34O5S1’, ‘C21H34O6S1’, ‘C21H34O8S1’, ‘C21H36O5S1’, ‘C21H36O9S2’, ‘C23H37N1O8S2’, ‘C23H38O4’, ‘C23H38O7S1’, ‘C24H30O8’, ‘C24H30O9’, ‘C24H32O10’, ‘C24H32O11S1’, ‘C24H32O9’, ‘C24H34O12’, ‘C24H34O5’, ‘C24H36O10’, ‘C24H36O12’, ‘C24H36O13’, ‘C24H38O12’, ‘C24H38O4’, ‘C24H38O5’, ‘C24H40O4’, ‘C24H40O5’, ‘C24H40O6S1’, ‘C24H40O8S1’, ‘C25H32O9’, ‘C25H34O8’, ‘C25H36O12S1’, ‘C25H38O11S1’, ‘C25H38O12S1’, ‘C25H38O8’, ‘C25H40O11S1’, ‘C26H36O12’, ‘C26H38O9’, ‘C26H40O10’, ‘C26H40O11’, ‘C26H40O9’, ‘C26H42O10’, ‘C26H42O12’, ‘C26H42O13’, ‘C26H43N1O8S1’, ‘C26H43N1O9S1’, ‘C26H44O11’, ‘C26H45N1O5S1’, ‘C26H45N1O6S1’, ‘C26H45N1O8S2’, ‘C26H45N1O9S2’, ‘C27H36O11’, ‘C27H36O14S1’, ‘C27H38O10’, ‘C27H38O12’, ‘C27H38O13S1’, ‘C27H38O14S1’, ‘C27H38O8’, ‘C27H38O9’, ‘C27H40O10’, ‘C27H40O11’, ‘C27H40O12’, ‘C27H40O12S1’, ‘C27H40O8’, ‘C27H40O9’, ‘C27H42O10’, ‘C27H42O11’, ‘C27H42O13S1’, ‘C27H44O8’, ‘C27H44O9’, ‘C30H40O15’, ‘C30H46O10’, ‘C30H46O11’, ‘C30H47N3O9S1’, ‘C30H48O10’, ‘C31H47N1O12S1’, ‘C33H46O16’, ‘C33H48O15’, ‘C33H48O17’, ‘C33H50O15’, ‘C33H50O16’]
Discarded features:
[’C18H22O3’, ‘C18H24O10S2’, ‘C18H24O3’, ‘C18H24O5S1’, ‘C18H24O7S1’, ‘C18H24O9S2’, ‘C18H28O10S1’, ‘C18H28O7’, ‘C18H28O9S1’, ‘C18H30O3’, ‘C18H30O5’, ‘C18H30O6’, ‘C18H30O9S1’, ‘C18H32O3’, ‘C19H24O3’, ‘C19H24O6S1’, ‘C19H26O6S1’, ‘C19H28O5S1’, ‘C19H28O6S1’, ‘C19H28O9S2’, ‘C19H30O2’, ‘C19H30O6S1’, ‘C19H30O8S2’, ‘C19H30O9S2’, ‘C19H32O2’, ‘C19H32O3’, ‘C19H32O5S1’, ‘C19H32O8S2’, ‘C19H32O9S2’, ‘C20H28O6’, ‘C20H28O9S1’, ‘C20H30O5’, ‘C20H30O6S1’, ‘C20H30O7S1’, ‘C20H30O8S1’, ‘C20H34O10S1’, ‘C20H34O5’, ‘C20H34O6’, ‘C20H34O7’, ‘C20H34O8S1’, ‘C20H36O8S1’, ‘C21H28O11S2’, ‘C21H28O5’, ‘C21H28O8S1’, ‘C21H28O9S1’, ‘C21H30O10S2’, ‘C21H30O11S2’, ‘C21H30O4’, ‘C21H30O5’, ‘C21H30O5S1’, ‘C21H30O6’, ‘C21H30O6S1’, ‘C21H30O7S1’, ‘C21H32O12S2’, ‘C21H32O3’, ‘C21H32O4’, ‘C21H32O5’, ‘C21H32O6S1’, ‘C21H32O8S1’, ‘C21H32O9S1’, ‘C21H34O10S2’, ‘C21H34O11S2’, ‘C21H34O2’, ‘C21H34O7S1’, ‘C21H34O9S2’, ‘C21H36O2’, ‘C21H36O3’, ‘C21H36O6S1’, ‘C23H37N1O5S1’, ‘C24H32O12S1’, ‘C24H32O13S1’, ‘C24H32O8’, ‘C24H34O8S1’, ‘C24H38O11’, ‘C24H38O7S1’, ‘C24H38O8S1’, ‘C24H38O9’, ‘C24H40O3’, ‘C24H40O7S1’, ‘C24H40O9’, ‘C25H34O9’, ‘C25H36O8’, ‘C25H36O9’, ‘C25H38O9’, ‘C25H39N1O6S1’, ‘C25H39N1O9S2’, ‘C25H40N2O6S1’, ‘C25H40N2O9S2’, ‘C25H40O12S1’, ‘C25H40O8’, ‘C25H40O9’, ‘C26H38O10’, ‘C26H38O11’, ‘C26H38O12’, ‘C26H42O11’, ‘C26H43N1O4’, ‘C26H43N1O5’, ‘C26H43N1O6’, ‘C26H43N1O7S1’, ‘C26H45N1O10S2’, ‘C26H45N1O7S1’, ‘C27H36O12’, ‘C27H36O15S1’, ‘C27H38O11’, ‘C27H38O15S1’, ‘C27H40O13S1’, ‘C27H40O14S1’, ‘C27H40O15S1’, ‘C27H42O12S1’, ‘C27H42O14S1’, ‘C27H42O8’, ‘C27H42O9’, ‘C27H44O12S1’, ‘C29H45N1O11S1’, ‘C29H46O10’, ‘C30H40O14’, ‘C30H40O16’, ‘C30H42O11’, ‘C30H47N3O12S2’, ‘C30H48O11’, ‘C30H48O9’, ‘C31H44O15’, ‘C31H46O14’, ‘C31H46O15’, ‘C31H48N2O12S1’, ‘C31H48O14’, ‘C31H48O15’, ‘C32H51N1O10’, ‘C32H51N1O11’, ‘C32H51N1O12’, ‘C32H53N1O11S1’, ‘C32H53N1O12S1’, ‘C32H53N1O13S1’, ‘C33H44O17’, ‘C33H44O18’, ‘C33H46O17’, ‘C33H46O18’, ‘C33H48O16’, ‘C33H48O18’, ‘C33H50O17’, ‘C33H52O15’]
Used model for feature importance determination: LogisticRegression() With normalization + log transformation Number of selected features: 130 Number of discarded features: 141

Table 24
Recursive Feature Elimination results of selected and discarded features on the normalized + log transformed dataset, using LDA for feature importance determination.

Selected features:
['C18H22O2', 'C18H22O3', 'C18H22O5S1', 'C18H24O10S2', 'C18H24O2', 'C18H24O3', 'C18H24O4', 'C18H24O5S1', 'C18H24O6S1', 'C18H24O7S1', 'C18H24O8S2', 'C18H24O9S2', 'C18H26O6', 'C18H26O9S1', 'C18H28O10S1', 'C18H28O4', 'C18H28O6', 'C18H28O7', 'C18H30O5', 'C18H30O6S1', 'C18H30O8S1', 'C18H30O9S1', 'C18H32O3', 'C18H32O6S1', 'C19H24O3', 'C19H24O6S1', 'C19H26O2', 'C19H26O3', 'C19H26O5S1', 'C19H26O6S1', 'C19H28O2', 'C19H28O3', 'C19H28O6S1', 'C19H28O9S2', 'C19H30O3', 'C19H30O5S1', 'C19H30O6S1', 'C19H30O8S2', 'C19H30O9S2', 'C19H32O2', 'C19H32O5S1', 'C19H32O6S1', 'C19H32O8S2', 'C19H32O9S2', 'C20H28O6', 'C20H28O9S1', 'C20H30O3', 'C20H30O4', 'C20H30O5', 'C20H30O6', 'C20H30O6S1', 'C20H30O7S1', 'C20H30O8S1', 'C20H30O9S1', 'C20H32O3', 'C20H32O4', 'C20H32O5', 'C20H32O6S1', 'C20H32O8S1', 'C20H34O4', 'C20H34O5', 'C20H34O6', 'C20H34O7', 'C20H34O7S1', 'C20H34O8S1', 'C20H34O9S1', 'C20H36O5', 'C20H36O8S1', 'C21H28O11S2', 'C21H28O5', 'C21H28O6', 'C21H28O8S1', 'C21H30O11S2', 'C21H30O12S2', 'C21H30O2', 'C21H30O3', 'C21H30O5', 'C21H30O5S1', 'C21H30O6S1', 'C21H30O8S1', 'C21H30O9S1', 'C21H32O10S2', 'C21H32O11S2', 'C21H32O12S2', 'C21H32O2', 'C21H32O3', 'C21H32O4', 'C21H32O5', 'C21H32O5S1', 'C21H32O6', 'C21H32O6S1', 'C21H32O7S1', 'C21H32O8S1', 'C21H32O9S1', 'C21H32O9S2', 'C21H34O10S2', 'C21H34O11S2', 'C21H34O2', 'C21H34O3', 'C21H34O4', 'C21H34O5S1', 'C21H34O6S1', 'C21H34O7S1', 'C21H34O8S1', 'C21H36O2', 'C21H36O3', 'C21H36O5S1', 'C21H36O6S1', 'C21H36O9S2', 'C23H37N1O5S1', 'C23H37N1O8S2', 'C23H38O4', 'C23H38O7S1', 'C24H30O8', 'C24H30O9', 'C24H32O10', 'C24H32O11S1', 'C24H32O12S1', 'C24H32O13S1', 'C24H32O8', 'C24H32O9', 'C24H34O12', 'C24H34O5', 'C24H34O8S1', 'C24H36O10', 'C24H36O12', 'C24H36O13', 'C24H38O11', 'C24H38O12', 'C24H38O5', 'C24H38O7S1', 'C24H38O8S1', 'C24H38O9', 'C24H40O3', 'C24H40O4', 'C24H40O5', 'C24H40O6S1', 'C24H40O7S1', 'C24H40O8S1', 'C24H40O9', 'C25H34O8', 'C25H34O9', 'C25H36O12S1', 'C25H36O8', 'C25H36O9', 'C25H38O11S1', 'C25H38O12S1', 'C25H38O8', 'C25H38O9', 'C25H39N1O6S1', 'C25H39N1O9S2', 'C25H40N2O6S1', 'C25H40O11S1', 'C25H40O12S1', 'C25H40O8', 'C25H40O9', 'C26H36O12', 'C26H38O10', 'C26H38O12', 'C26H38O9', 'C26H40O10', 'C26H40O11', 'C26H40O9', 'C26H42O10', 'C26H42O11', 'C26H42O12', 'C26H42O13', 'C26H43N1O4', 'C26H43N1O5', 'C26H43N1O6', 'C26H43N1O7S1', 'C26H43N1O8S1', 'C26H43N1O9S1', 'C26H44O11', 'C26H45N1O10S2', 'C26H45N1O5S1', 'C26H45N1O6S1', 'C26H45N1O7S1', 'C26H45N1O8S2', 'C26H45N1O9S2', 'C27H36O11', 'C27H36O12', 'C27H36O14S1', 'C27H36O15S1', 'C27H38O10', 'C27H38O11', 'C27H38O12', 'C27H38O13S1', 'C27H38O14S1', 'C27H38O8', 'C27H38O9', 'C27H40O10', 'C27H40O11', 'C27H40O12', 'C27H40O12S1', 'C27H40O13S1', 'C27H40O14S1', 'C27H40O15S1', 'C27H40O8', 'C27H40O9', 'C27H42O11', 'C27H42O12S1', 'C27H42O13S1', 'C27H42O14S1', 'C27H42O8', 'C27H42O9', 'C27H44O12S1', 'C27H44O8', 'C27H44O9', 'C29H45N1O11S1', 'C29H46O10', 'C30H40O14', 'C30H40O15', 'C30H42O11', 'C30H46O10', 'C30H46O11', 'C30H47N3O12S2', 'C30H47N3O9S1', 'C30H48O10', 'C30H48O11', 'C30H48O9', 'C31H44O15', 'C31H46O14', 'C31H46O15', 'C31H47N1O12S1', 'C31H48N2O12S1', 'C31H48O14', 'C31H48O15', 'C32H51N1O10', 'C32H51N1O11', 'C32H51N1O12', 'C32H53N1O11S1', 'C32H53N1O12S1', 'C32H53N1O13S1', 'C33H44O17', 'C33H44O18', 'C33H46O16', 'C33H46O17', 'C33H46O18', 'C33H48O15', 'C33H48O17', 'C33H48O18', 'C33H50O15', 'C33H50O16', 'C33H50O17', 'C33H52O15']
Discarded features:
['C18H22O6S1', 'C18H28O7S1', 'C18H28O9S1', 'C18H30O3', 'C18H30O6', 'C19H28O5S1', 'C19H30O2', 'C19H32O3', 'C20H32O7S1', 'C20H34O10S1', 'C21H28O9S1', 'C21H30O10S2', 'C21H30O4', 'C21H30O6', 'C21H30O7S1', 'C21H34O5', 'C21H34O9S2', 'C24H38O4', 'C25H32O9', 'C25H40N2O9S2', 'C26H38O11', 'C27H38O15S1', 'C27H42O10', 'C30H40O16', 'C33H48O16']
Used model for feature importance determination: LinearDiscriminantAnalysis() With normalization + log transformation Number of selected features: 246 Number of discarded features: 25

therefore, our recommendation that for datasets with a small number of minority class samples, the number of folds should be kept low, while the number of repeats should be increased.

5. **Using an alternative dataset:** The traits of the dataset used for this research determined the strategy to eventually identify three classifiers. Our proposed strategy is general and robust to work on other similar datasets as well. However, we could not investigate whether our strategy works on other datasets or not due to a lack of comparable datasets with similar traits. It is, therefore, our recommendation that the proposed strategy should be tested on similar other datasets whenever they are publicly available or created.
6. **Using Deep Learning algorithms:** There is a growing interest in the use of deep learning algorithms that are based on Artificial Neural Networks. We, therefore, recommend this alternative as an option to investigate for future research.

CRediT authorship contribution statement

Paul Mooijman: Conceptualization, Methodology, Software, Writing – review & editing. **Cagatay Catal:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Bedir Tekinerdogan:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Arjen Lommen:** Data curation, Methodology, Writing – review & editing. **Marco Blokland:** Data curation, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data are available at the following webpage: https://github.com/mooym001/machine_learning

Acknowledgment

Open Access funding provided by the Qatar National Library.

Appendix

See Table 26a.

Table 25a
Predicted classes by LR. combined with several over/ undersamplers, for all samples of the test dataset, with the total number of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN).

sample	LR	+SMOTE	+SMOTE +NCR	+BSMOTE	+BSMOTE +Tomek	+ADASYN	+ADASYN +Tomek	LR(CSL)	y_true
TP	6	7	7	7	7	7	7	7	7
FN	1	0	0	0	0	0	0	0	0
FP	1	3	3	3	2	3	3	4	0
TN	117	115	115	115	116	115	115	114	118
Sample_001	0	0	0	0	0	0	0	0	0
Sample_002	0	0	0	0	0	0	0	0	0
Sample_003	0	0	0	0	0	0	0	0	0
Sample_004	0	0	0	0	0	0	0	0	0
Sample_005	0	0	0	0	0	0	0	0	0
Sample_006	0	0	0	0	0	0	0	0	0
Sample_007	0	0	0	0	0	0	0	0	0
Sample_008	0	0	0	0	0	0	0	0	0
Sample_009	0	0	0	0	0	0	0	0	0
Sample_010	0	0	0	0	0	0	0	0	0
Sample_011	0	0	0	0	0	0	0	0	0
Sample_012	0	0	0	0	0	0	0	0	0
Sample_013	0	0	0	0	0	0	0	0	0
Sample_014	0	0	0	0	0	0	0	0	0
Sample_015	0	0	0	0	0	0	0	0	0
Sample_016	0	0	0	0	0	0	0	0	0
Sample_017	0	0	0	0	0	0	0	0	0
Sample_018	0	0	0	0	0	0	0	0	0
Sample_019	0	0	0	0	0	0	0	0	0
Sample_020	0	0	0	0	0	0	0	0	0
Sample_021	0	0	0	0	0	0	0	0	0
Sample_022	1	1	1	1	1	1	1	1	0
Sample_023	0	0	0	0	0	0	0	0	0
Sample_024	1	1	1	1	1	1	1	1	1
Sample_025	0	0	0	0	0	0	0	0	0
Sample_026	1	1	1	1	1	1	1	1	1
Sample_027	0	0	0	0	0	0	0	0	0
Sample_028	0	0	0	0	0	0	0	0	0
Sample_029	0	0	0	0	0	0	0	0	0
Sample_030	0	0	0	0	0	0	0	0	0
Sample_031	0	0	0	0	0	0	0	0	0
Sample_032	0	1	1	1	1	1	1	1	1
Sample_033	0	1	1	0	0	1	1	1	0
Sample_034	1	1	1	1	1	1	1	1	1
Sample_035	0	0	0	0	0	0	0	0	0
Sample_036	0	0	0	0	0	0	0	0	0
Sample_037	0	0	0	0	0	0	0	0	0
Sample_038	0	0	0	0	0	0	0	0	0
Sample_039	0	0	0	0	0	0	0	0	0
Sample_040	0	0	0	0	0	0	0	0	0
Sample_041	0	0	0	0	0	0	0	0	0
Sample_042	0	0	0	0	0	0	0	0	0
Sample_043	0	0	0	0	0	0	0	0	0
Sample_044	0	0	0	0	0	0	0	0	0
Sample_045	0	1	1	1	1	1	1	1	0
Sample_046	0	0	0	0	0	0	0	0	0
Sample_047	0	0	0	0	0	0	0	0	0
Sample_048	0	0	0	0	0	0	0	0	0
Sample_049	0	0	0	0	0	0	0	0	0
Sample_050	0	0	0	0	0	0	0	0	0
Sample_051	0	0	0	0	0	0	0	0	0
Sample_052	0	0	0	0	0	0	0	0	0
Sample_053	0	0	0	0	0	0	0	0	0
Sample_054	0	0	0	0	0	0	0	0	0
Sample_055	0	0	0	0	0	0	0	0	0
Sample_056	0	0	0	0	0	0	0	0	0
Sample_057	0	0	0	0	0	0	0	0	0

Table 25b
Continuation of the predicted classes by LR with several over/ undersamplers, for all samples of the test dataset.

sample	LR	+SMOTE	+SMOTE +NCR	+BSMOTE	+BSMOTE +Tomek	+ADASYN	+ADASYN +Tomek	LR(CSL)	y_true
TP	6	7	7	7	7	7	7	7	7
FN	1	0	0	0	0	0	0	0	0
FP	1	3	3	3	2	3	3	4	0
TN	117	115	115	115	116	115	115	114	118
Sample_058	0	0	0	0	0	0	0	0	0
Sample_059	0	0	0	0	0	0	0	0	0
Sample_060	0	0	0	0	0	0	0	0	0
Sample_061	0	0	0	0	0	0	0	0	0
Sample_062	0	0	0	0	0	0	0	0	0
Sample_063	0	0	0	0	0	0	0	0	0
Sample_064	0	0	0	0	0	0	0	0	0
Sample_065	0	0	0	0	0	0	0	0	0
Sample_066	0	0	0	0	0	0	0	0	0
Sample_067	0	0	0	0	0	0	0	0	0
Sample_068	0	0	0	0	0	0	0	0	0
Sample_069	0	0	0	0	0	0	0	0	0
Sample_070	0	0	0	0	0	0	0	0	0
Sample_071	0	0	0	0	0	0	0	0	0
Sample_072	0	0	0	0	0	0	0	0	0
Sample_073	0	0	0	0	0	0	0	0	0
Sample_074	0	0	0	0	0	0	0	0	0
Sample_075	0	0	0	0	0	0	0	0	0
Sample_076	0	0	0	0	0	0	0	0	0
Sample_077	0	0	0	0	0	0	0	0	0
Sample_078	0	0	0	0	0	0	0	0	0
Sample_079	0	0	0	0	0	0	0	0	0
Sample_080	0	0	0	0	0	0	0	0	0
Sample_081	0	0	0	0	0	0	0	0	0
Sample_082	0	0	0	0	0	0	0	0	0
Sample_083	0	0	0	0	0	0	0	0	0
Sample_084	0	0	0	0	0	0	0	0	0
Sample_085	0	0	0	0	0	0	0	0	0
Sample_086	0	0	0	0	0	0	0	0	0
Sample_087	0	0	0	0	0	0	0	0	0
Sample_088	0	0	0	0	0	0	0	0	0
Sample_089	0	0	0	0	0	0	0	0	0
Sample_090	0	0	0	0	0	0	0	0	0
Sample_091	0	0	0	0	0	0	0	0	0
Sample_092	0	0	0	0	0	0	0	0	0
Sample_093	0	0	0	0	0	0	0	0	0
Sample_094	0	0	0	0	0	0	0	0	0
Sample_095	0	0	0	0	0	0	0	0	0
Sample_096	0	0	0	0	0	0	0	0	0
Sample_097	0	0	0	0	0	0	0	0	0
Sample_098	0	0	0	0	0	0	0	0	0
Sample_099	0	0	0	0	0	0	0	0	0
Sample_100	0	0	0	0	0	0	0	0	0
Sample_101	0	0	0	0	0	0	0	0	0
Sample_102	0	0	0	0	0	0	0	0	0
Sample_103	0	0	0	0	0	0	0	0	0
Sample_104	0	0	0	0	0	0	0	0	0
Sample_105	0	0	0	0	0	0	0	0	0
Sample_106	0	0	0	0	0	0	0	0	0
Sample_107	0	0	0	0	0	0	0	0	0
Sample_108	0	0	0	0	0	0	0	0	0
Sample_109	0	0	0	0	0	0	0	0	0
Sample_110	0	0	0	1	0	0	0	1	0
Sample_111	0	0	0	0	0	0	0	0	0
Sample_112	0	0	0	0	0	0	0	0	0
Sample_113	0	0	0	0	0	0	0	0	0
Sample_114	0	0	0	0	0	0	0	0	0
Sample_115	0	0	0	0	0	0	0	0	0
Sample_116	0	0	0	0	0	0	0	0	0
Sample_117	0	0	0	0	0	0	0	0	0
Sample_118	0	0	0	0	0	0	0	0	0
Sample_119	0	0	0	0	0	0	0	0	0
Sample_120	0	0	0	0	0	0	0	0	0
Sample_121	0	0	0	0	0	0	0	0	0
Sample_122	0	0	0	0	0	0	0	0	0
Sample_123	1	1	1	1	1	1	1	1	1
Sample_124	1	1	1	1	1	1	1	1	1
Sample_125	1	1	1	1	1	1	1	1	1

Table 26a
Predicted classes by GaussianProc (GPC) and LDA combined with several over/ undersamplers, for all samples of the test dataset, with the total number of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN).

sample	GPC (DotProduct)	+SMOTE	+BSMOTE	+ADASYN	LDA (svd)	y_true
TP	6	7	7	7	6	7
FN	1	0	0	0	1	0
FP	1	3	2	3	1	0
TN	117	115	116	115	117	118
Sample_001	0	0	0	0	0	0
Sample_002	0	0	0	0	0	0
Sample_003	0	0	0	0	0	0
Sample_004	0	0	0	0	0	0
Sample_005	0	0	0	0	0	0
Sample_006	0	0	0	0	0	0
Sample_007	0	0	0	0	0	0
Sample_008	0	0	0	0	0	0
Sample_009	0	0	0	0	0	0
Sample_010	0	0	0	0	0	0
Sample_011	0	0	0	0	0	0
Sample_012	0	0	0	0	0	0
Sample_013	0	0	0	0	0	0
Sample_014	0	0	0	0	0	0
Sample_015	0	0	0	0	0	0
Sample_016	0	0	0	0	0	0
Sample_017	0	0	0	0	0	0
Sample_018	0	0	0	0	0	0
Sample_019	0	0	0	0	0	0
Sample_020	0	0	0	0	0	0
Sample_021	0	0	0	0	0	0
Sample_022	1	1	1	1	1	0
Sample_023	0	0	0	0	0	0
Sample_024	1	1	1	1	1	1
Sample_025	0	0	0	0	0	0
Sample_026	1	1	1	1	1	1
Sample_027	0	0	0	0	0	0
Sample_028	0	0	0	0	0	0
Sample_029	0	0	0	0	0	0
Sample_030	0	0	0	0	0	0
Sample_031	0	0	0	0	0	0
Sample_032	0	1	1	1	0	1
Sample_033	0	1	0	1	0	0
Sample_034	1	1	1	1	1	1
Sample_035	0	0	0	0	0	0
Sample_036	0	0	0	0	0	0
Sample_037	0	0	0	0	0	0
Sample_038	0	0	0	0	0	0
Sample_039	0	0	0	0	0	0
Sample_040	0	0	0	0	0	0
Sample_041	0	0	0	0	0	0
Sample_042	0	0	0	0	0	0
Sample_043	0	0	0	0	0	0
Sample_044	0	0	0	0	0	0
Sample_045	0	1	1	1	0	0
Sample_046	0	0	0	0	0	0
Sample_047	0	0	0	0	0	0
Sample_048	0	0	0	0	0	0
Sample_049	0	0	0	0	0	0
Sample_050	0	0	0	0	0	0
Sample_051	0	0	0	0	0	0
Sample_052	0	0	0	0	0	0
Sample_053	0	0	0	0	0	0
Sample_054	0	0	0	0	0	0
Sample_055	0	0	0	0	0	0
Sample_056	0	0	0	0	0	0
Sample_057	0	0	0	0	0	0

Table 26b
Continuation of predicted classes by GaussianProc (GPC) and LDA combined with several over/ undersamplers.

sample	GPC (DotProduct)	+SMOTE	+BSMOTE	+ADASYN	LDA (svd)	y_true
Sample_058	0	0	0	0	0	0
Sample_059	0	0	0	0	0	0
Sample_060	0	0	0	0	0	0
Sample_061	0	0	0	0	0	0
Sample_062	0	0	0	0	0	0
Sample_063	0	0	0	0	0	0
Sample_064	0	0	0	0	0	0
Sample_065	0	0	0	0	0	0
Sample_066	0	0	0	0	0	0
Sample_067	0	0	0	0	0	0
Sample_068	0	0	0	0	0	0
Sample_069	0	0	0	0	0	0
Sample_070	0	0	0	0	0	0
Sample_071	0	0	0	0	0	0
Sample_072	0	0	0	0	0	0
Sample_073	0	0	0	0	0	0
Sample_074	0	0	0	0	0	0
Sample_075	0	0	0	0	0	0
Sample_076	0	0	0	0	0	0
Sample_077	0	0	0	0	0	0
Sample_078	0	0	0	0	0	0
Sample_079	0	0	0	0	0	0
Sample_080	0	0	0	0	0	0
Sample_081	0	0	0	0	0	0
Sample_082	0	0	0	0	0	0
Sample_083	0	0	0	0	0	0
Sample_084	0	0	0	0	0	0
Sample_085	0	0	0	0	0	0
Sample_086	0	0	0	0	0	0
Sample_087	0	0	0	0	0	0
Sample_088	0	0	0	0	0	0
Sample_089	0	0	0	0	0	0
Sample_090	0	0	0	0	0	0
Sample_091	0	0	0	0	0	0
Sample_092	0	0	0	0	0	0
Sample_093	0	0	0	0	0	0
Sample_094	0	0	0	0	0	0
Sample_095	0	0	0	0	0	0
Sample_096	0	0	0	0	0	0
Sample_097	0	0	0	0	0	0
Sample_098	0	0	0	0	0	0
Sample_099	0	0	0	0	0	0
Sample_100	0	0	0	0	0	0
Sample_101	0	0	0	0	0	0
Sample_102	0	0	0	0	0	0
Sample_103	0	0	0	0	0	0
Sample_104	0	0	0	0	0	0
Sample_105	0	0	0	0	0	0
Sample_106	0	0	0	0	0	0
Sample_107	0	0	0	0	0	0
Sample_108	0	0	0	0	0	0
Sample_109	0	0	0	0	0	0
Sample_110	0	0	0	0	0	0
Sample_111	0	0	0	0	0	0
Sample_112	0	0	0	0	0	0
Sample_113	0	0	0	0	0	0
Sample_114	0	0	0	0	0	0
Sample_115	0	0	0	0	0	0
Sample_116	0	0	0	0	0	0
Sample_117	0	0	0	0	0	0
Sample_118	0	0	0	0	0	0
Sample_119	0	0	0	0	0	0
Sample_120	0	0	0	0	0	0
Sample_121	0	0	0	0	0	0
Sample_122	0	0	0	0	0	0
Sample_123	1	1	1	1	1	1
Sample_124	1	1	1	1	1	1
Sample_125	1	1	1	1	1	1

References

[1] A. Singh, R.K. Ranjan, A. Tiwari, Credit card fraud detection under extreme imbalanced data: A comparative study of data-level algorithms, J. Exp. Theor. Artif. Intell. (2021) 1–28, http://dx.doi.org/10.1080/0952813X.2021.1907795.

[2] O. Alan, C. Catal, Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets, Expert Syst. Appl. 38 (4) (2011) 3440–3445.

[3] C. Catal, U. Sevim, B. Diri, Metrics-driven software quality prediction without prior fault data, in: Electronic Engineering and Computing Technology, Springer, Dordrecht, 2010, pp. 189–199.

[4] W.L.S. Yee, C.L. Drum, Increasing complexity to simplify clinical care: High resolution mass spectrometry as an enabler of AI guided clinical and therapeutic monitoring, Adv. Therapeut. 3 (4) (2020) 1900163, http://dx.doi.org/10.1002/adtp.201900163.

[5] N. Liu, X. Li, E. Qi, M. Xu, L. Li, B. Gao, A novel ensemble learning paradigm for medical diagnosis with imbalanced data, IEEE Access 8 (2020) 171263–171280, http://dx.doi.org/10.1109/ACCESS.2020.3014362.

[6] S.L. Bartelt-Hunt, D.D. Snow, W.L. Kranz, T.L. Mader, C.A. Shapiro, S.J.V. Donk, D.P. Shelton, D.D. Tarkalson, T.C. Zhang, Effect of growth promotants on the occurrence of endogenous and synthetic steroid hormones on feedlot soils and in runoff from beef cattle feeding operations, Environ. Sci. Technol. 46 (3) (2012) 1352–1360, http://dx.doi.org/10.1021/es202680q.

[7] R. Stella, D. Bovo, E. Mastrorilli, M. Pezzolato, E. Bozzetta, G. Biancotto, Anabolic treatments in bovines: quantification of plasma protein markers of dexamethasone administration, Proteomics (2021) http://dx.doi.org/10.1002/pmic.202000238, 2000238.

[8] U.W. Liebal, A.N.T. Phan, M. Sudhakar, K. Raman, L.M. Blank, Machine learning applications for mass spectrometry-based metabolomics, Metabolites 10 (6) (2020) 1–23, http://dx.doi.org/10.3390/metabo10060243.

[9] R. Bouwmeester, R. Gabriels, T. Van Den Bossche, L. Martens, S. Degroeve, The age of data-driven proteomics: How machine learning enables novel workflows, Proteomics 20 (21–22) (2020) 1–6, http://dx.doi.org/10.1002/pmic.201900351.

[10] D.G. Rocha, M.A.G. Lana, D.C. de Assis, A.N. de Macedo, J.M. Corrêa, R. Augusti, A.F. Faria, A novel strategy for the detection of boldenone undecylenate misuse in cattle using ultra-high performance liquid chromatography coupled to high resolution orbitrap mass spectrometry: From non-targeted to targeted, Drug Test. Anal. 14 (4) (2022) 667–675.

[11] A. Benedetto, M. Pezzolato, E. Robotti, E. Biasibetti, A. Poirier, G. Dervilly, et al., Profiling of transcriptional biomarkers in FFPE liver samples: PLS-DA applications for detection of illicit administration of sex steroids and clenbuterol in veal calves, Food Control 128 (2021a) 108149.

[12] A. Benedetto, M. Pezzolato, E. Biasibetti, E. Bozzetta, Omics applications in the fight against abuse of anabolic substances in cattle: challenges, perspectives and opportunities, Curr. Opin. Food Sci. 40 (2021b) 112–120.

[13] R. Draisci, L. Palleschi, E. Ferretti, L. Lucentini, P. Cammarata, Quantitation of anabolic hormones and their metabolites in bovine serum and urine by liquid chromatography-tandem mass spectrometry, J. Chromatogr. A 870 (1–2) (2000) 511–522, http://dx.doi.org/10.1016/S0021-9673(99)01293-5.

[14] J.C.W. Rijk, A. Lommen, M.L. Essers, M.J. Groot, J.M. Van Hende, T.G. Doeswijk, M.W.F. Nielen, Metabolomics approach to anabolic steroid urine profiling of bovines treated with prohormones, Anal. Chem. 81 (16) (2009) 6879–6888, http://dx.doi.org/10.1021/ac900874m.

[15] R. Angeletti, L. Contiero, G. Gallina, C. Montesissa, The urinary ratio of testosterone to epitetosterone: A good marker of illegal treatment also in cattle? Veterinary Res. Commun. 30 (SUPPL. 1) (2006) 127–131, http://dx.doi.org/10.1007/s11259-006-0025-9.

[16] K. Verheyden, H. Noppe, J. Vanden Bussche, K. Wille, K. Bekaert, L. De Boever, J. Van Acker, C.R. Janssen, H.F. De Brabander, L. Vanhaecke, Characterisation of steroids in wooden crates of veal calves by accelerated solvent extraction (ASE®) and ultra-high performance liquid chromatography coupled to triple quadrupole mass spectrometry (U-HPLC-QqQ-MS-MS), Anal. Bioanal. Chem. 397 (1) (2010) 345–355, http://dx.doi.org/10.1007/s00216-010-3462-9.

[17] A.M. Richardson, B.A. Lidbury, Enhancement of hepatitis virus immunoassay outcome predictions in imbalanced routine pathology data by data balancing and feature selection before the application of support vector machines, BMC Med. Inf. Decis. Mak. 17 (1) (2017) 1–11, http://dx.doi.org/10.1186/s12911-017-0522-5.

[18] R. Low, L. Cheah, L. You, Commercial vehicle activity prediction with imbalanced class distribution using a hybrid sampling and gradient boosting approach, IEEE Trans. Intell. Transp. Syst. 22 (3) (2021) 1401–1410, http://dx.doi.org/10.1109/TITS.2020.2970229.

[19] G. Karatas, O. Demir, O.K. Sahingoz, Increasing the performance of machine learning-based IDSs on an imbalanced and up-to-date dataset, IEEE Access 8 (2020) 32150–32162, http://dx.doi.org/10.1109/ACCESS.2020.2973219.

[20] A. Kaya, A.S. Keceli, C. Catal, B. Tekinerdogan, The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models, J. Softw.: Evol. Process 31 (9) (2019) 1–25, http://dx.doi.org/10.1002/smr.2164.

[21] W.J. Lin, J.J. Chen, Class-imbalanced classifiers for high-dimensional data, Brief. Bioinform. 14 (1) (2013) 13–26, http://dx.doi.org/10.1093/bib/bbs006.

[22] J.M. Johnson, T.M. Khoshgoftaar, Survey on deep learning with class imbalance, J. Big Data 6 (1) (2019) http://dx.doi.org/10.1186/s40537-019-0192-5.

[23] N. Jiang, N. Li, A wind turbine frequent principal fault detection and localization approach with imbalanced data using an improved synthetic oversampling technique, Int. J. Electr. Power Energy Syst. 126 (PA) (2021) 106595, http://dx.doi.org/10.1016/j.ijepes.2020.106595.

[24] P.A. Patrician, Multiple imputation for missing data, Res. Nurs. Health 25 (1) (2002) 76–84, http://dx.doi.org/10.1002/nur.10015.

[25] S. Gorard, Handling missing data in numeric analyses, Int. J. Soc. Res. Methodol. 23 (6) (2020) 651–660, http://dx.doi.org/10.1080/13645579.2020.1729974.

[26] J. Brownlee, Imbalanced classification with python - choose better metrics, balance skewed classes, and apply cost-sensitive learning, in: Machine Learning Mastery, 2020, p. 463.