Vol.66: e23220556, 2023
https://doi.org/10.1590/1678-4324-2023220556
ISSN 1678-4324 Online Edition

Article - Engineering, Technology and Techniques

Studying Intelligent Techniques Acting in Large Power Transformer Monitoring

Elvis Ricardo de Oliveira[1]
https://orcid.org/0000-0002-6290-9055

Vanias de Araujo Junior[1,2]
https://orcid.org/0000-0002-8480-801X

José Faustino da Silva Cândido[1,2]
https://orcid.org/0000-0002-0275-5285

Germano Lambert-Torres[1*]
https://orcid.org/0000-0003-3789-4696

Luiz Eduardo Borges da Silva[1]
https://orcid.org/0000-0003-2298-1017

Erik Leandro Bonaldi[1]
https://orcid.org/0000-0003-4350-9248

Gilberto Capistrano Cunha de Andrade[1]
https://orcid.org/ 0000-0001-7745-1940

Levy Ely de Lacerda de Oliveira[1]
https://orcid.org/ 0000-0001-9052-5801

Carlos Henrique Valério de Moraes[3]
https://orcid.org/0000-0003-1361-2754

Carlos Eduardo Teixeira[1]
https://orcid.org/0000-0003-3399-1878

[1]Instituto Gnarus, Itajubá, Minas Gerais, Brasil; [2]Energética Suape II, Cabo de Santo Agostinho, Pernambuco, Brasil, [3]Universidade Federal de Itajubá, Instituto de Sistemas e Tecnologia de Informação, Itajubá, Minas Gerais, Brasil.

Editor-in-Chief: Alexandre Rasi Aoki
Associate Editor: Alexandre Rasi Aoki

Received: 21-Jul-2022; Accepted: 24-Mar-2023

*Correspondence: germanoltorres@gmail.com; Tel.: +55-35- 3622-0132 (G.L.T.).

HIGHLIGHTS

- Intelligent diagnostic system for power transformers.
- Defining the operational behavior of large transformers.
- Ten intelligent classifiers were tested to establish the operational condition of power transformers.

Abstract: The presented development is an intelligent diagnostic system for transformers that studied machine learning techniques to determine the operational status of these transformers. The study of these techniques is initiated by observing the quantities that define the operational behavior of large transformers, aiming to identify anomalies in their operation from data from sensors that equipment it in the functioning environment. This large power transformer has a theoretical service life of above 20 years and a low failure rate. Thus, obtaining failure values, which have their evolution monitored for large transformers, is almost nil. Therefore, a supervised machine training methodology to diagnose these cases is practically unfeasible. The study carried out with several traditional intelligent techniques can verify this. Several supervised methods (Closest Neighbor K-th Neighbor, Support Vector Machine, Radial Base Function, Decision Trees, Random Forest, Neural Network, AdaBoost, Gaussian Naive Bayes, and Quadratic Discriminant Analysis) were studied.

Keywords: Artificial Intelligence; Power Transformers; Data Analytics; Intelligent Techniques.

INTRODUCTION

The power transformer is a critical element in the connections of the electrical power system. In addition, it is one of the most expensive and essential equipment of the substations, so the management of this asset is vital for maintaining the reliability of the operation of the entire system [1].

It is responsible for transforming voltage levels in the electrical system, either by raising a voltage (when it is called an elevator transformer and is usually positioned in a substation of a plant) or when reduced by the voltage. This latter type of transformer is located in lowering substations or urban distributions when placed on poles or in the power input booth [2, 3].

In addition to this classification, transformers may receive other ratings linked to their position in the system, voltage classes and function in the power grid. In this work, the transformers of interest are those called large size, those that conduct large blocks of power of the order of megawatts.

The reliability of these large transformers not only affects the availability of large blocks of electricity but can also lead to technical-economic losses with consequences that can be significant in commercial and environmental terms, often generating substantial fines and outgoing profit problems [4-6]. Thus, the need for detecting and identifying anomalies in its initial stage of development for possible preventive action is justified, which is essentially achieved with continuous monitoring of the transformer. Monitoring this type of transformer costs very little to the price of a unit of this equipment, which can easily reach the order of millions of dollars [7].

Thus, it should be noted in this monitoring that during any failure or defect of the transformer, the integrity of its mechanical, dielectric, and thermal parts can be affected, alone or jointly, causing its electrical parameters to vary to those considered normal of its operating state. Incipient anomalies can evolve and, for example, cause deterioration of dielectric insulation, such as in cases of short circuits and electrical overvoltages in transients and their operative maneuvers.

This procedure of detecting and monitoring defects in any asset is inserted in predictive maintenance, in which trend analyses and predictions of possible asset downtimes are made.

This paper presents another step toward an extensive transformer monitoring system, where several methods based on Artificial Intelligence techniques are analyzed, with the mission of defining the operational state of the transformer. These methods are compared in light of several criteria using existing data in public databases to determine the best characteristics of the data types usually found in these monitoring systems.

Problems with the existing monitoring systems

Despite all these facts, which unequivocally show the advantages and the need to monitor large transformers, not all have operational monitoring systems. Most of them have only monitoring systems restricted to specific transformer functions, for example, the quality of insulating oil, several of which are offline.

Insulating oil for any transformer (but much more for large ones) is essential to know where in internal connections, corrosion of metal parts, and degradation of insulating paper, among many other aspects [8]. It is said in the electrical sector that "the analysis of insulating oil is for the transformer, as the blood test is for the analysis of a human patient."

The condition and analysis of the oil are only one of several that can be studied in a transformer. Needs of its high voltage bushings, work at high temperatures, overloads, large transients, operations outside the standard operating regimes, and presence of harmonics, among many others, are essential observation points for monitoring large transformers. However, these monitoring and their consequent survey of measured values generate a wide range of data and information that must be processed to respond to the observed transformer's operational state.

The complexity of the information available to detect incipient defects and the simultaneous existence of various defects are the main reasons for the search for more efficient diagnostic systems, such as those based on pattern recognition techniques or strategies with machine learning.

Unfortunately, transformer monitoring systems available in the market have two characteristics that negatively affect the present work. The first is linked to being these systems primarily proprietary. The data is acquired, processed and delivered to the user only as the result of the analysis. Nothing is said about the data, which is recorded in databases with its format and that can only be accessed by the computer program of the company that built the monitoring system. Even the power utility, which owns the transformer and the monitoring system, does not have clear access to the measurement database. And in the event of having this access, the second harmful characteristic comes to this work, which is the non-public disclosure of these data, which are treated as confidential and considered almost a competitive secret by the concessionaires. Thus, few public transformer databases are available for testing data analytics techniques.

The development of this paper is part of a broader project aimed at developing a monitoring and diagnostic system for significant power transformers of Brazil's most prominent fuel oil power plant.

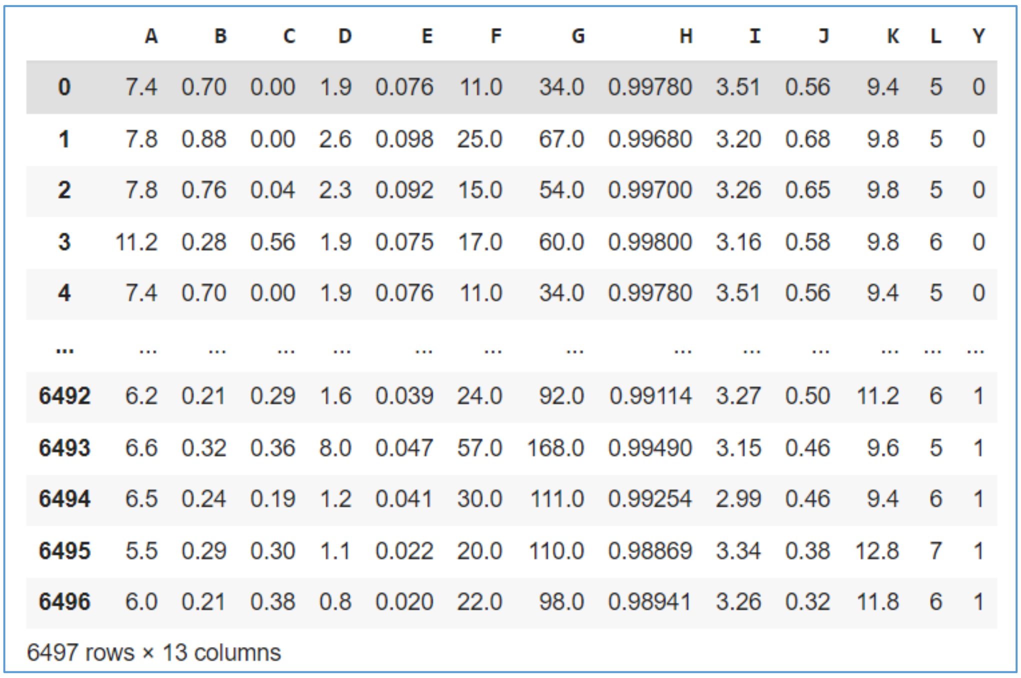
This work should develop a computational tool for analyzing the data available on the server, which will be integrated into the asset management system to issue an output related to the degree of criticality of the asset.

Several techniques of extracting characteristics were studied, using public databases from tests and transformer monitoring. These bases do not contain the same largeness found in this project, but the challenge is establishing the best technique, which will be implemented in the monitoring system.

Presentation of the used database

This section shows the features of the used database. This database is a complete database that has all sectors of the hypercube with at least one representative example, while an incomplete database has some sectors without examples. The complete database is separable, meaning hyperplanes can be perfectly divided into the hypercube.

The public database used in this study has 6437 rows (samples) and 13 columns, 12 columns of input data (attributes) and 1 column output with the results. Figure 1 shows part of the used database.



	A	B	C	D	E	F	G	H	I	J	K	L	Y
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	1
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	1
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	1
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	1
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	1

6497 rows × 13 columns

Figure 1. Part of the public database used in this study

Developed classification system

In this study, a classification system was developed to evaluate transformer databases. Nine traditional intelligent classification techniques were used: k-Nearest Neighbors (kNN) [9], Support Vector Machine (SVM) [10], Support Vector Machine using Radial Base Function (RBF-SVM) [11], Decision Tree Learning (DT) [12], Random Forest (RF) [13], Multilayer Perceptron Neural Network (MPNN) [14], AdaBoost (AB) [15], Naive Bayes (NB) [16], and Quadratic Discriminant Analysis (QDA) [17].

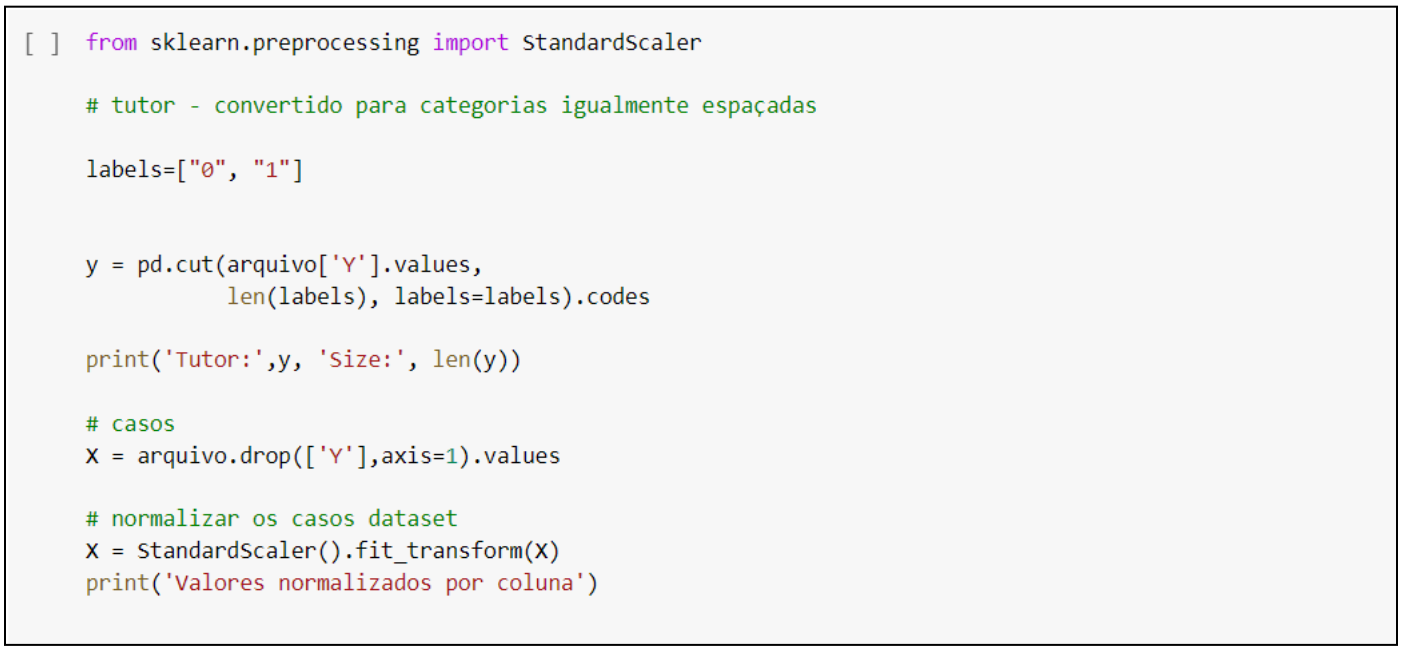
Presentation of the Classification System Program

The classification system was developed using the Python language on the Google Collaboratory platform. An interactive environment called the Colab notebook allows you to write and execute code, which does not need to be configured and runs in the cloud. The system features five blocks of code or notebooks, described below:

Block 1: In this block, the Pandas' library (data analysis and manipulation tool developed based on the Python programming language), the reading of a CSV file and assigning the data from that file to a dataset variable, is imported.

Block 2: The panda *describe()* function, which presents basic statistical details of a series of numerical values, was used to evaluate the data of each column of the dataset, such as the amount of data, the mean of the values, the standard deviation, and the distribution of the data in quartiles (minimum, 25%, 50%, 75% and maximum).

Block 3: This block converts the output column into classes and is assigned to the y vector using the *cut()* function. It also removes the column from the dataset y vector through the drop function. It normalizes the data from dataset x through the *StandardScaler* function since some classifiers work only with normalized values, as shown in Figure 2.

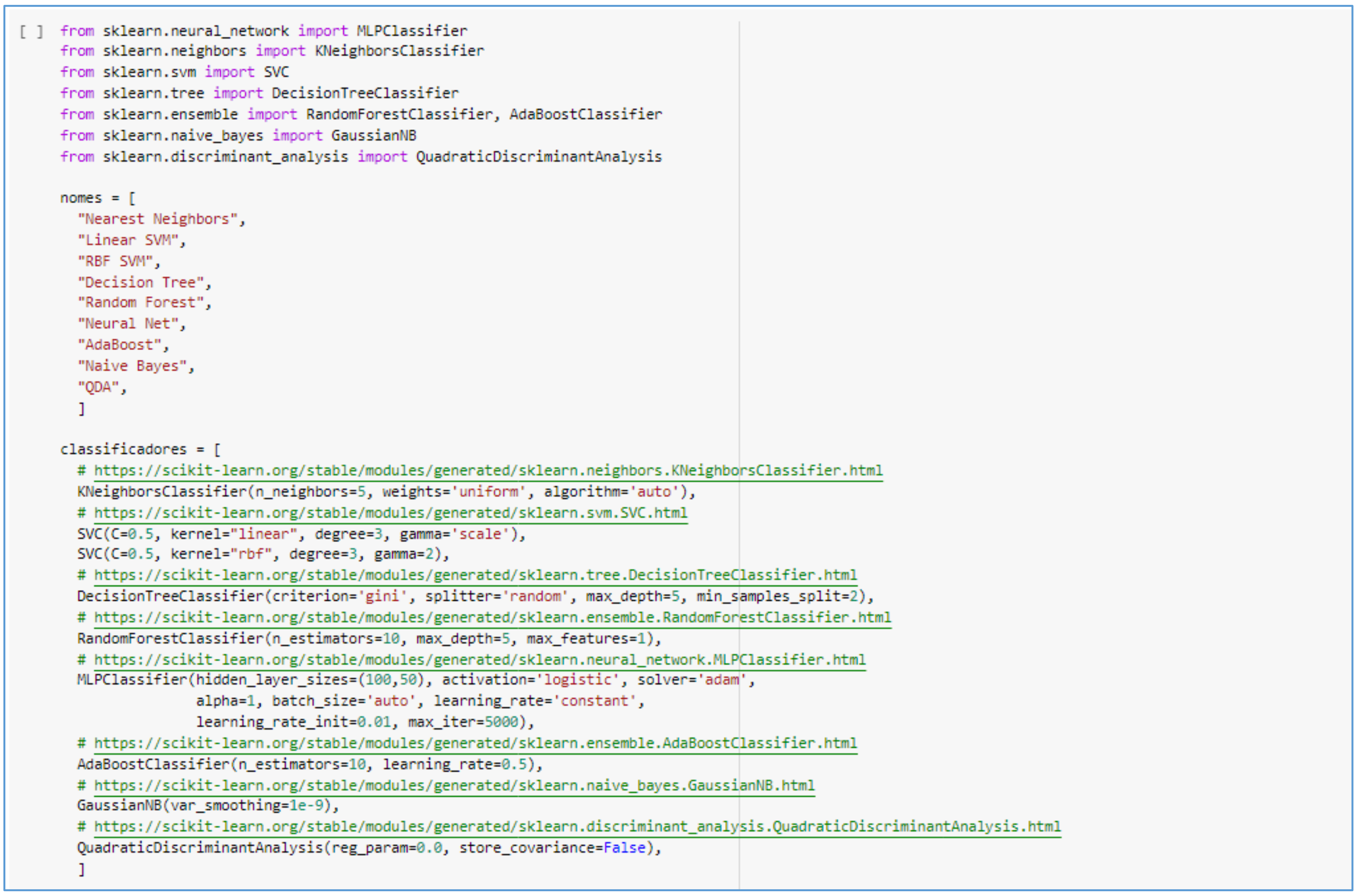


```
[ ]  from sklearn.preprocessing import StandardScaler

     # tutor - convertido para categorias igualmente espaçadas

     labels=["0", "1"]


     y = pd.cut(arquivo['Y'].values,
                len(labels), labels=labels).codes

     print('Tutor:',y, 'Size:', len(y))

     # casos
     X = arquivo.drop(['Y'],axis=1).values

     # normalizar os casos dataset
     X = StandardScaler().fit_transform(X)
     print('Valores normalizados por coluna')
```

Figure 2. Normalization of input data and classification of outputs as classes

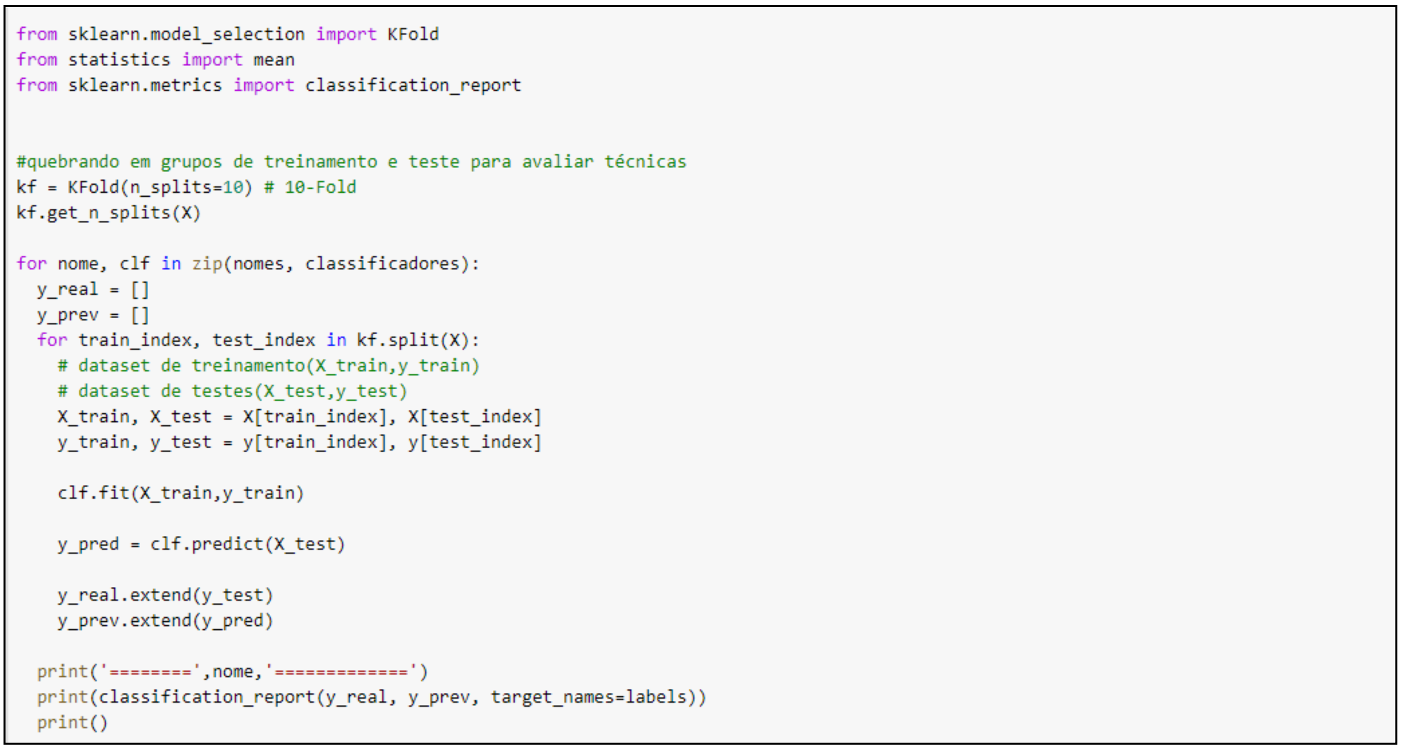
Block 4: Traditional intelligent classification techniques from the sklearn library are imported: MLPClassifier, GaussianNB, SVC, DecisionTreeClassifier, RandomForestClassifier, KNeighborsClassifier, Quadratic Discriminant Analysis, and AdaBoostClassifier. Two vectors were used, one with the names of the techniques (KNN, SVM, RBF-SVM, DT, RF, MPNN, AB, NB, and QDA) and the other of the classifiers and their parameters, as shown in Figure 3.

Block 5: K-fold cross-validation is used, an intensive computational technique that uses all available samples as training and test samples. More accurate results can be achieved with this to other methods of cross-validation. Given a hypothetical database, and by setting the k=10, the database will be divided into ten subsets. After splitting into subsets, a subset will be used in model validation, and the remaining sets will be used as training. The cross-validation process is then repeated K (10) times so that each of the K subsets is used exactly once as a test for model validation. For example, for data 10 subsets B1, B2, ..., and B10, the first step of K-Fold is to use B1 for testing and from B2 to B10 for training. In the second step, B2 is used for testing, and everything else for training, including B1 used for testing in the first step, the third step until the tenth will be applied the same logic successively. The final result of K-Fold validation is the average classifier performance in the K tests. Repeating the tests several times increases the reliability of the estimation of the accuracy of the classifier. With the defined sets, the classifiers are trained where a report of each classifier is printed, showing the main classification metrics, as shown in Figure 4.



```
[ ] from sklearn.neural_network import MLPClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.svm import SVC
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

    nomes = [
      "Nearest Neighbors",
      "Linear SVM",
      "RBF SVM",
      "Decision Tree",
      "Random Forest",
      "Neural Net",
      "AdaBoost",
      "Naive Bayes",
      "QDA",
      ]

    classificadores = [
      # https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
      KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto'),
      # https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
      SVC(C=0.5, kernel="linear", degree=3, gamma='scale'),
      SVC(C=0.5, kernel="rbf", degree=3, gamma=2),
      # https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
      DecisionTreeClassifier(criterion='gini', splitter='random', max_depth=5, min_samples_split=2),
      # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
      RandomForestClassifier(n_estimators=10, max_depth=5, max_features=1),
      # https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
      MLPClassifier(hidden_layer_sizes=(100,50), activation='logistic', solver='adam',
                    alpha=1, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.01, max_iter=5000),
      # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
      AdaBoostClassifier(n_estimators=10, learning_rate=0.5),
      # https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
      GaussianNB(var_smoothing=1e-9),
      # https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html
      QuadraticDiscriminantAnalysis(reg_param=0.0, store_covariance=False),
      ]
```

Figure 3. Loading the classifier libraries and adjusting the parameters



```
from sklearn.model_selection import KFold
from statistics import mean
from sklearn.metrics import classification_report


#quebrando em grupos de treinamento e teste para avaliar técnicas
kf = KFold(n_splits=10) # 10-Fold
kf.get_n_splits(X)

for nome, clf in zip(nomes, classificadores):
  y_real = []
  y_prev = []
  for train_index, test_index in kf.split(X):
    # dataset de treinamento(X_train,y_train)
    # dataset de testes(X_test,y_test)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    clf.fit(X_train,y_train)

    y_pred = clf.predict(X_test)

    y_real.extend(y_test)
    y_prev.extend(y_pred)

  print('========',nome,'=============')
  print(classification_report(y_real, y_prev, target_names=labels))
  print()
```

Figure 4. Network training using the k-fold method

In response after training, the program presents a classification report for each of the classification techniques used. For report generation, some features of the Scikit-learn library are used for the metrics, as shown below in the next topic.

Scikit-learn Library Features

Scikit-learn is an open-source machine learning library that supports supervised and unsupervised learning. It also provides several tools for model tuning, data preprocessing, model selection and evaluation, and various other utilities.

Classification of k-Nearest Neighbors

k-NN classifier (here called sklearn.neighbors) is a type of neighbor-based classification based on *instance-based learning* or *non-generalizing learning*: it does not attempt to build a general internal model but simply stores instances of training data. The ranking is calculated from a simple majority of votes from the nearest neighbors to each point: a query point is assigned to the data class with the most delegates within the nearest neighbors.

Classification of Support Vectors Machine

SVC (here called sklearn.svm) is a machine learning algorithm for classification and regression. The implementation is based on *libsvm*. The adjustment time is scaled at least to quadratic with the number of samples and can be impractical beyond tens of thousands of samples.

Classification of Decision Tree

DT classifier (here called sklearn.tree) is a nonparametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from data resources. A tree can be seen as a constant approximation.

Classification of Random Forest

RF classifier (here called sklearn.ensemble.randomforestclassifier) is a meta-estimator that fits into a series of decision tree classifiers in multiple subsamples of the dataset and uses the mean to improve predictive accuracy and control over docking. The sub-sample size is controlled with the parameter if (default); otherwise, the entire dataset is used to construct each tree.

Classification of Neural Networks

The NN classifier is a multiple-layer perceptron classifier (here called sklearn.ensemble.neuralnetclassifier). This model optimizes the log loss function using LBFGS or stochastic gradient descent. The MPNN classifier trains iteratively because, at each moment, the partial derivatives of the loss function to the model's parameters are computed to update the parameters. A regularization term can also be added to the loss function that reduces model parameters to avoid over-adaptation. This implementation works with data represented as dense NumPy arrays or sparse arrays of floating-point values.

Classification of AdaBoost

AB classifier (here called sklearn.ensemble.adaboostclassifier) is a meta-estimator that begins by fitting a classifier into the original dataset and then snaps into additional copies of the classifier in the same dataset. However, the weights of incorrectly classified instances are adjusted so that subsequent classifiers focus more on complex cases.

Classification of Naive Bayes

Gaussian NB (here called sklearn.naive_bayes) is an algorithm based on Thomas Bayes's discovery to make machine learning predictions. The term "naive" refers to the way the algorithm analyzes database resources: it assumes that the resources are independent of each other. You can perform online updates to template parameters via *partial_fit*. The probability of the characteristics is considered Gaussian. Parameters are estimated using maximum probability.

Classification of Quadratic Discriminant Analysis

QDA (here called sklearn.discriminant_analysis) is a classifier with a quadratic decision limit generated by the adequacy of class conditional densities to the data and using the Bayes rule. The model fits into a Gaussian density for each class. Linear and Quadratic Discriminant Analysis are two classifiers with, as their names suggest, a linear and quadratic decision surface, respectively. These classifiers are attractive because they have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice and do not have hyperparameters to tune into.

Classification Metrics Used in the Classification Process

The main ranking metrics, called sklearn.metrics.classification_report, create a report showing the main ranking metrics: precision, recall, f-measure, support, macro average, weighted average, and sample average. Reported averages include:

- macro average - average of the unweighted average per label;
- weighted average - mean weighted average per label; and
- sample average - available only in the multilabel classification.

The first classification metrics used precision, recall, and support concepts, here called sklearn.metrics.precision_recall_fscore_support, are calculated for each class:

- Precision is the ratio indicated in (1), where P is the number of true positives and FP s the number of false positives. This metric is intuitively the classifier's ability not to label as positive a negative sample.

$$\textit{precision} = \mathrm{TP}/((\mathrm{TP}+\mathrm{FP})) \quad (1)$$

- Recall (called Sensitivity) is the ratio indicated in (2), where TP is the number of true positives and FN is the number of false negatives. The recall is intuitively the classifier's ability to find all positive samples.

$$\textit{recall} = \mathrm{TP}/((\mathrm{TP}+\mathrm{FN})) \quad (2)$$

- Support is the number of samples in which each metric was calculated. For unbalanced classes, with a significant variation of support values, the classifier may tend to classify the new cases as being of the class that had the most examples during training.

The second classification metrics used F1 score, accuracy, and macro average.

- The harmonic mean of precision and recall gives a class's F1 score (f1-score). It combines accuracy and recalls into a metric, as indicated in (3). Since its value is high, the accuracy obtained is relevant. VP, VN, FP, and FN measured values do not present significant distortions. It can also be interpreted as a measure of accuracy and reliability.

$$\textit{F1} = 2\ (\textit{precision} * \textit{recall})\ /\ (\textit{precision} + \textit{recall}) = \mathrm{TP}\ /\ (\mathrm{TP}+\ (\mathrm{FP}+\mathrm{FN})/2) \quad (3)$$

- Accuracy is the fraction of correct predictions; that is, among all classifications, how many the model correctly classified testifies the overall performance of the model, as shown in (4).

$$\textit{accuracy} = (\mathrm{TP}+\mathrm{TN})\ /\ (\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}) \quad (4)$$

- Macro average is the average of the unweighted mean per class. Calculates the metric independently for each class and then gets the average of them (handling all classes equally). The weighted average is the average of the support-weighted average for each class. Calculates the metric independently for each class and then gets their weighted average.

Each metric has its peculiarities that should be considered when choosing how the classification model will be evaluated. One should not think of one metric as better or worse than the other in general, but rather the problem should be analyzed and the one that best fits.

One of the ways to analyze the results is to observe the accuracy, verifying that the F1 score followed the mean value of accuracy and recall, so the accuracy obtained is reliable, and the model can be classified as good or bad.

Another way is to observe for each class the different combinations of recall and precision, which have the following meanings:

- High recall + high precision: The model perfectly handles the class.
- Low recall + high accuracy: The model cannot detect the class well but is highly reliable when it does.
- High recall + low accuracy: The class is well caught, but the model includes points from other classes.
- Low recall + low accuracy: The class is mishandled by the model.

Results using the Complete Database

This section presents the results using a complete public database described above, applying intelligent techniques. This section starts with the adjustments used in each approach.

Changes in the Parameters of Intelligent Classification Techniques

For the KNeighborsClassifier, the parameter n_neighbors (number of neighbors to be used by default) was changed because the parameter with the possibility of higher results and the range of values used in the tests are presented.
Default Parameters
KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
Original code parameters
KNeighborsClassifier(n_neighbors=3, weights='uniform', algorithm='auto')
Test Parameters:
n_neighbors = varying from 3 to 10, with step of1

For the Linear SVM classifier, the parameter C_Regularization_parameter was changed, as it is the parameter with the possibility of the greatest change in results and the ranges of values used in the tests.
Default Parameters
SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)
Original code parameters
SVC(C=0.025, kernel='linear', degree=3, gamma='scale')
Test Parameters:
C= 0.025 til 1, with step of 0.025
degree = linear

For the RBF SVM classifier, the parameters C_Regularization and training_Gama_parameter were changed, as they are the parameters with the possibility of greater changes in the results and their ranges of values used in the tests.
Default Parameters
SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)
Original code parameters
SVC(C=1.0, kernel='rbf', degree=3, gamma=2)
Test Parameters:
C = 0.25 til 2, with step of 0.25
degree = RBF
gamma = 1 a 10, with step of1

For the classifier, the parameters max_depth_parameter maximum depth of the tree and min_samples_split_parameter (of the minimum number of samples required to be on a node) were changed, as they are the parameters with the possibility of further changes in the results and their ranges of values used in the tests.DecisionTreeClassifier
Default Parameters
DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alpha=0.0)

Original code parameters

DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=5, min_samples_split=2)

Test Parameters:

max_depth = 1 til 10

min_samples_split = 2 til 10, with step of1

For the RandomForestClassifier classifier, the parameters n_estimators_parameter (with the number of trees in the forest), max_depth_parameter (with the maximum depth of the tree), and max_features_parameter (with the number of resources to be considered when searching for the best division) were changed, because they are the parameters with the possibility of greater changes in the results and their ranges of values used in the tests.

Default Parameters

RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)

Original code parameters

RandomForestClassifier(n_estimators=10, max_depth=5, max_features=1)

Test Parameters:

n_estimators = 1 til 100, with step of 1

max_depth = 3 til 7, with step of 1

max_features = 1 til 10, with step of 1

For the Neural Net classifier, the parameters hidden_layer_sizes_parameter (which represents the number of neurons in the hidden layer), alpha_parameter penalty (regularization term), learning_rate_init_parameter (which means the initial rate of learning used), and max_iter_parameter (with the maximum number of iterations) were changed, because they are the parameters with the possibility of significant changes in the results and their ranges of values used in the tests.

Default Parameters

MLPClassifier(hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)

Original code parameters

MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=1, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, max_iter=1000)

Test Parameters:

hidden_layer_sizes = 100 til 10000, with step of 1000.

alpha = 0 til 0.001, with step of 0.0002

learning_rate_init = 0.0001 til 0.001, with step of 0.0002

max_iter = 1000 til 10000, with step of 1000

For the AdaBoost classifier, the parameters n_estimators_parameter (where the maximum number of estimators in which the perfect fit is terminated) and learning_rate_parameter (where the weight is applied to each classifier in each iteration) were changed. It occurs because they are the parameters with the possibility of greater changes in the results and their ranges of values used in the tests.

Default Parameters

AdaBoostClassifier(base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)

Original code parameters

AdaBoostClassifier(n_estimators=50, earning_rate=1.0)

Test Parameters:

n_estimators = 1 til 100, with step of 1

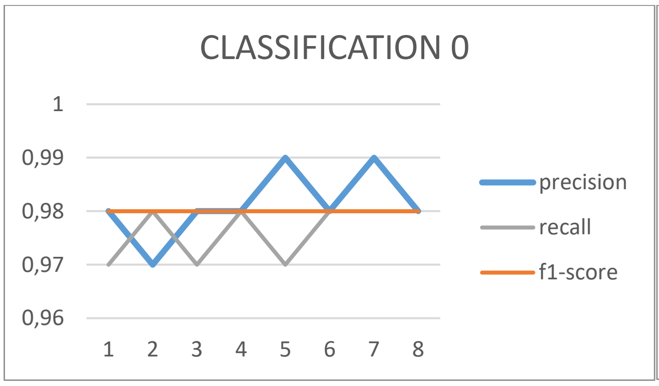
learning_rate = 0,5 til 2,0, with step of 0,5

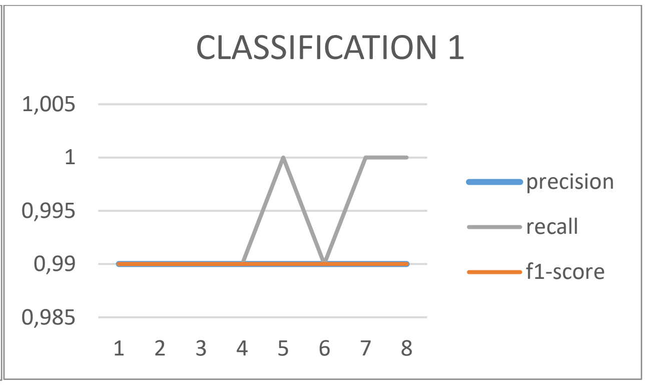
For the Naive Bayes classifier, the changed parameter var_smoothing_parameter (with the largest variance of all the features that is added to the variances for calculation stability) was changed, as it is the parameter with the possibility of greater change in the results and the ranges of values used in the tests.
Default Parameters
GaussianNB(*, priors=None, var_smoothing=1e-09)
Original code parameters
GaussianNB(var_smoothing = 1e-9)
Test Parameters:
var_smoothing = 1e-9 til 1e-6, with step of 2e-8

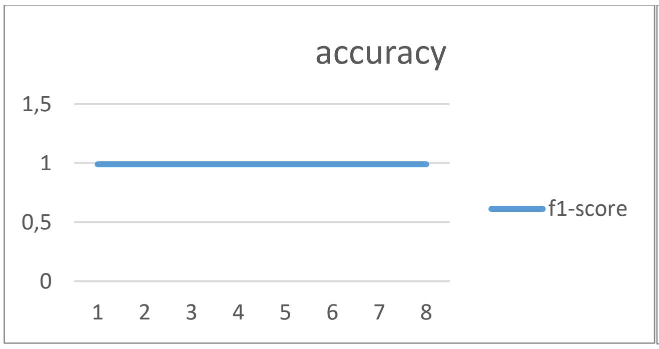
For the QuadraticDiscriminantAnalysis classifier, the parameters reg_param_Parameter (that regularizes the estimates of covariance by class) and store_covariance_parameter (where ever true, class covariance matrices are explicitly calculated and stored in the self.covariance attribute) was changed, because they are the parameters with the possibility of significant changes in the results and their ranges of values used in the tests.
Default Parameters
QuadraticDiscriminantAnalysis(*, priors=None, reg_param=0.0, store_covariance=False, tol=0.0001)
Original code parameters
QuadraticDiscriminantAnalysis(reg_param=0.0, store_covariance=False)
Test Parameters:
reg_param = 0,0 til 1,0, with step of 0,25
store_covariance = [False, True]

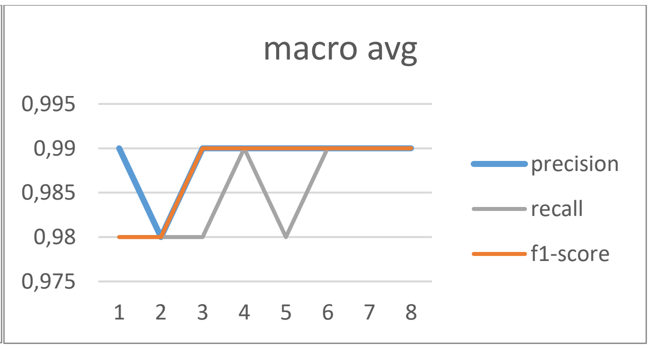
Final Results Applying the Intelligent Techniques in the Complete Database

All the traditional intelligent techniques mentioned in the previous subsection were applied to arrive at the results of the tests of the complete base. Figures 5 to 13 show the main results.









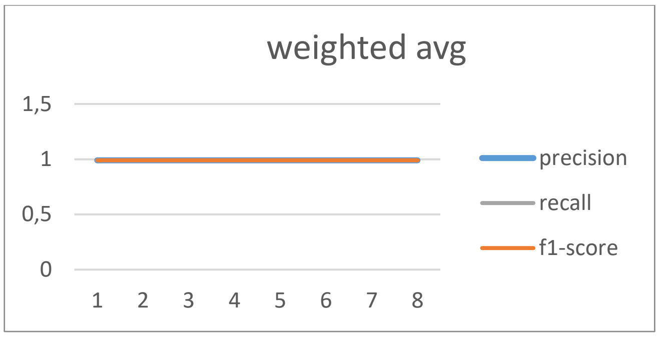


Figure 5. Results for the technique in neighbours KNeighbors

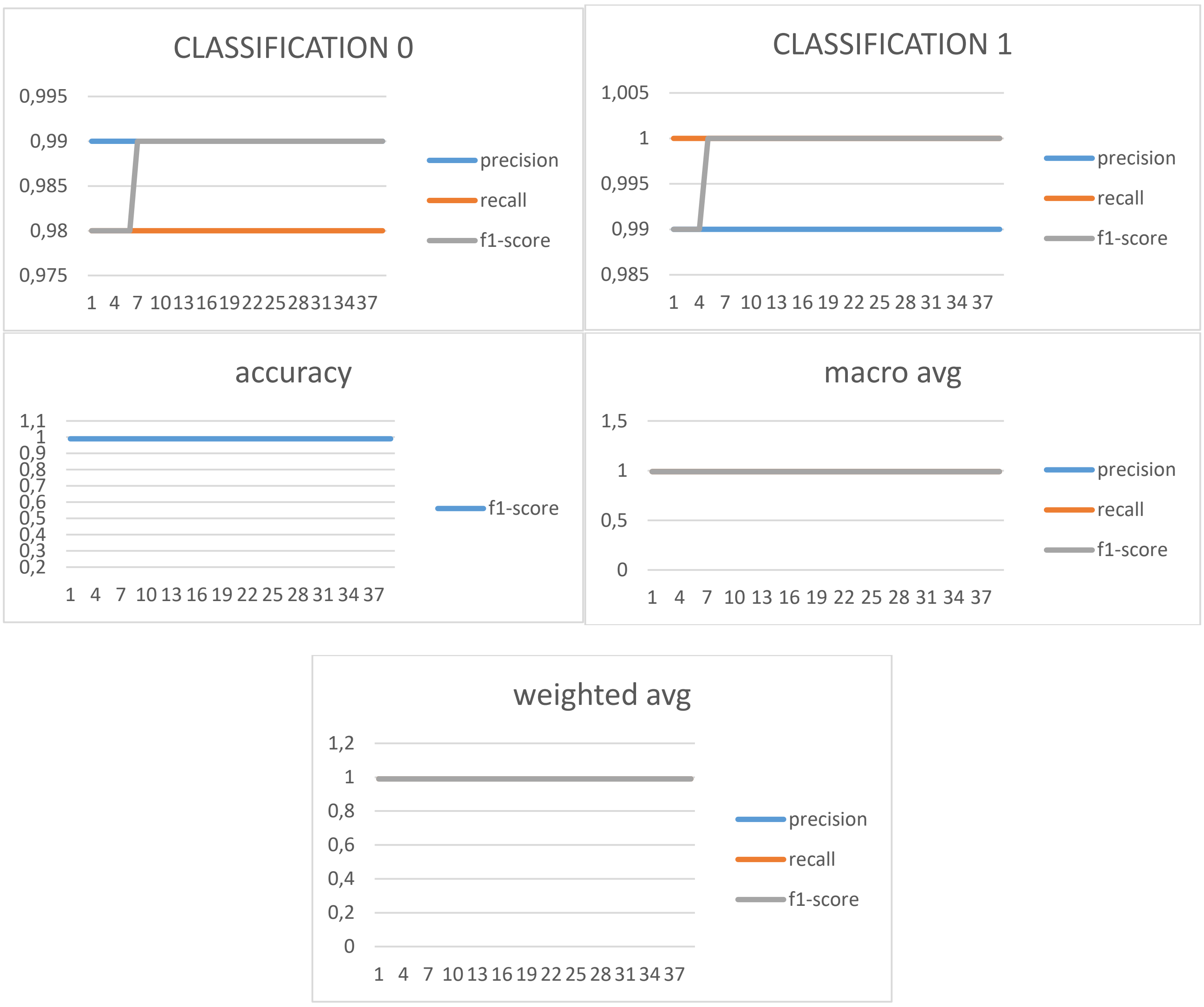


Figure 6. Results for the Linear SVM Classifier

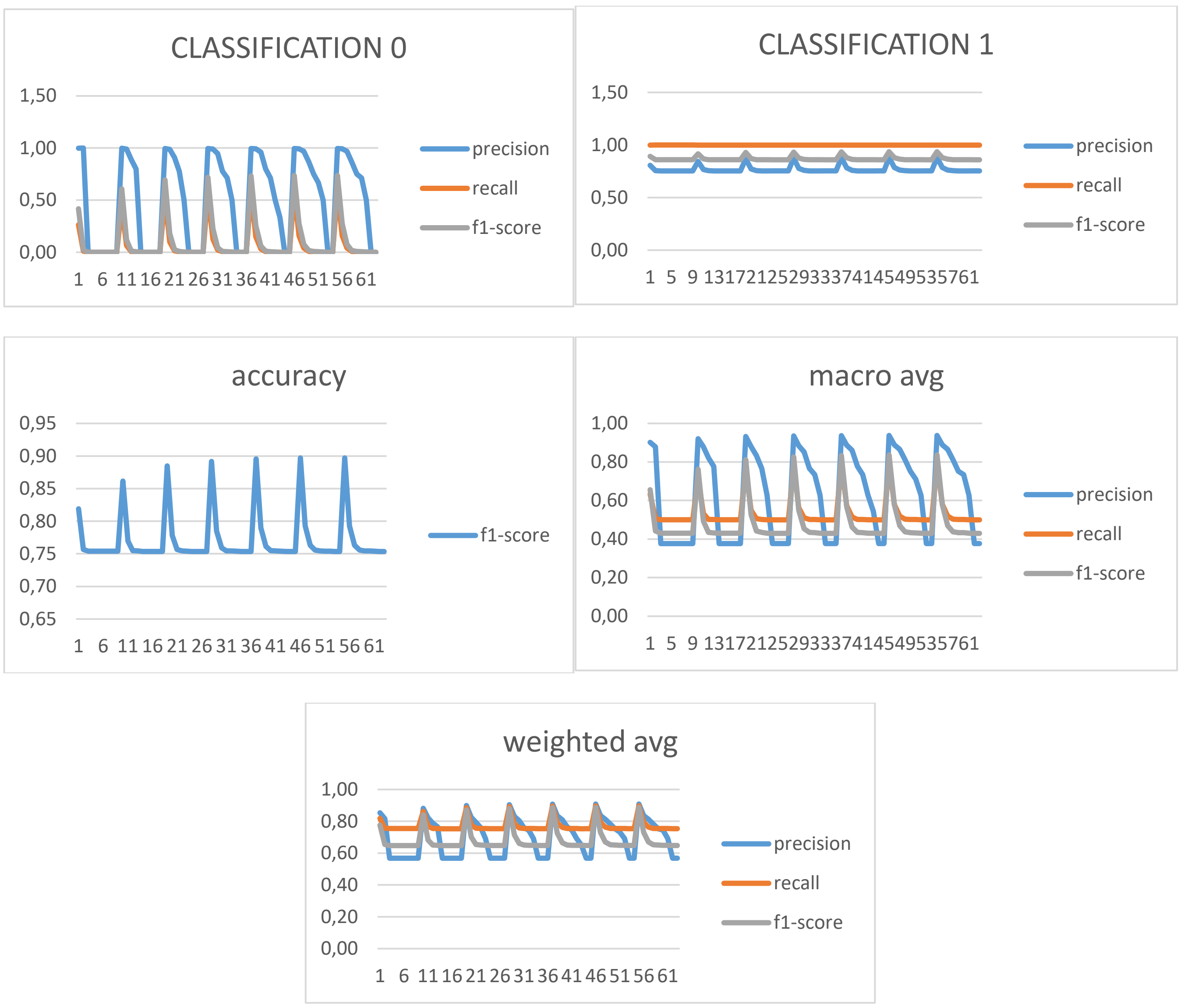
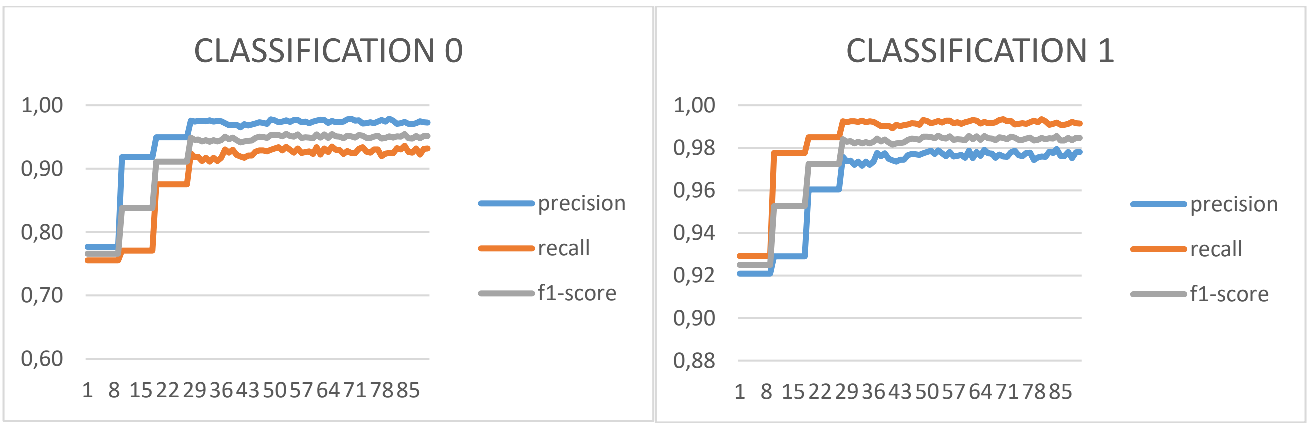


Figure 7. Results for the RBF SVM classifier



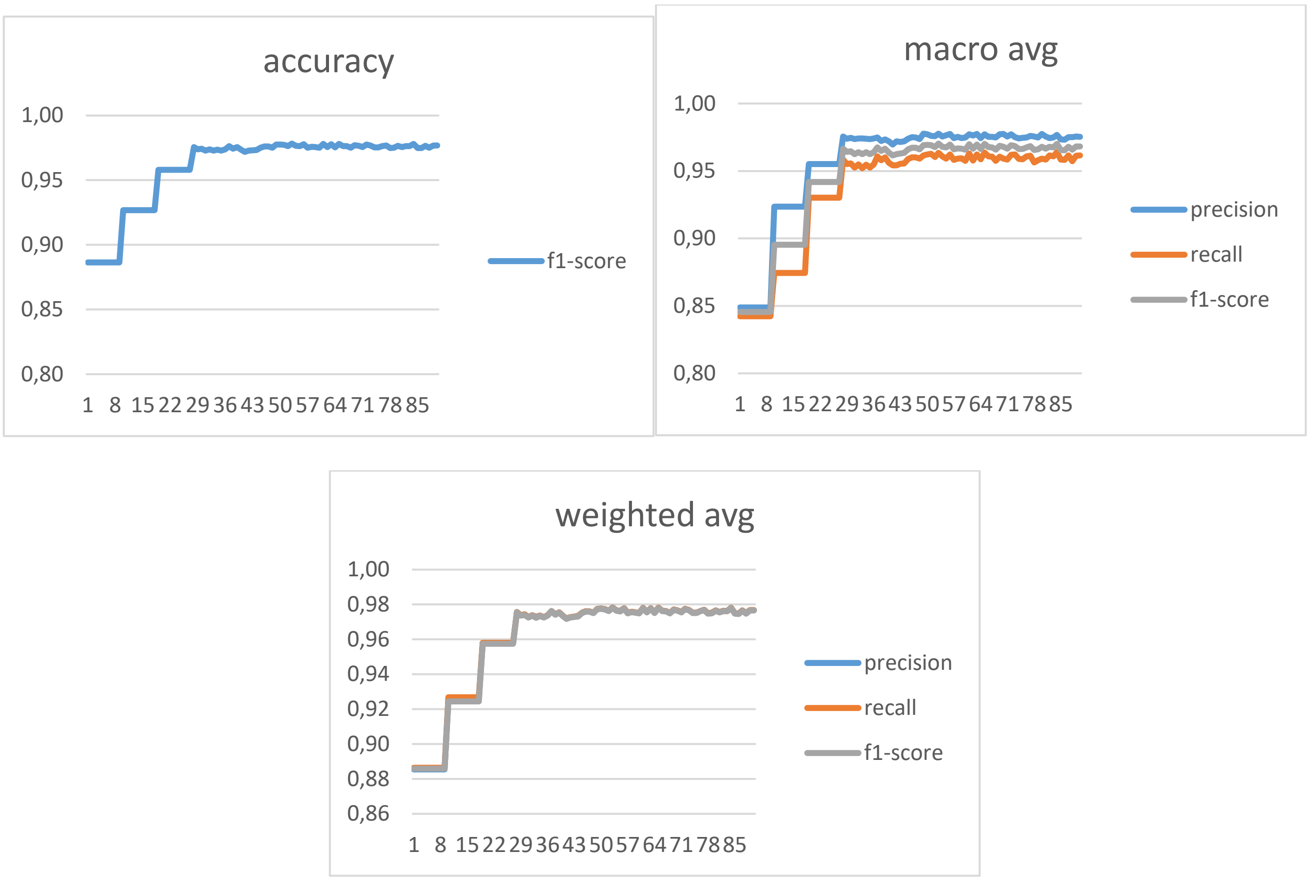
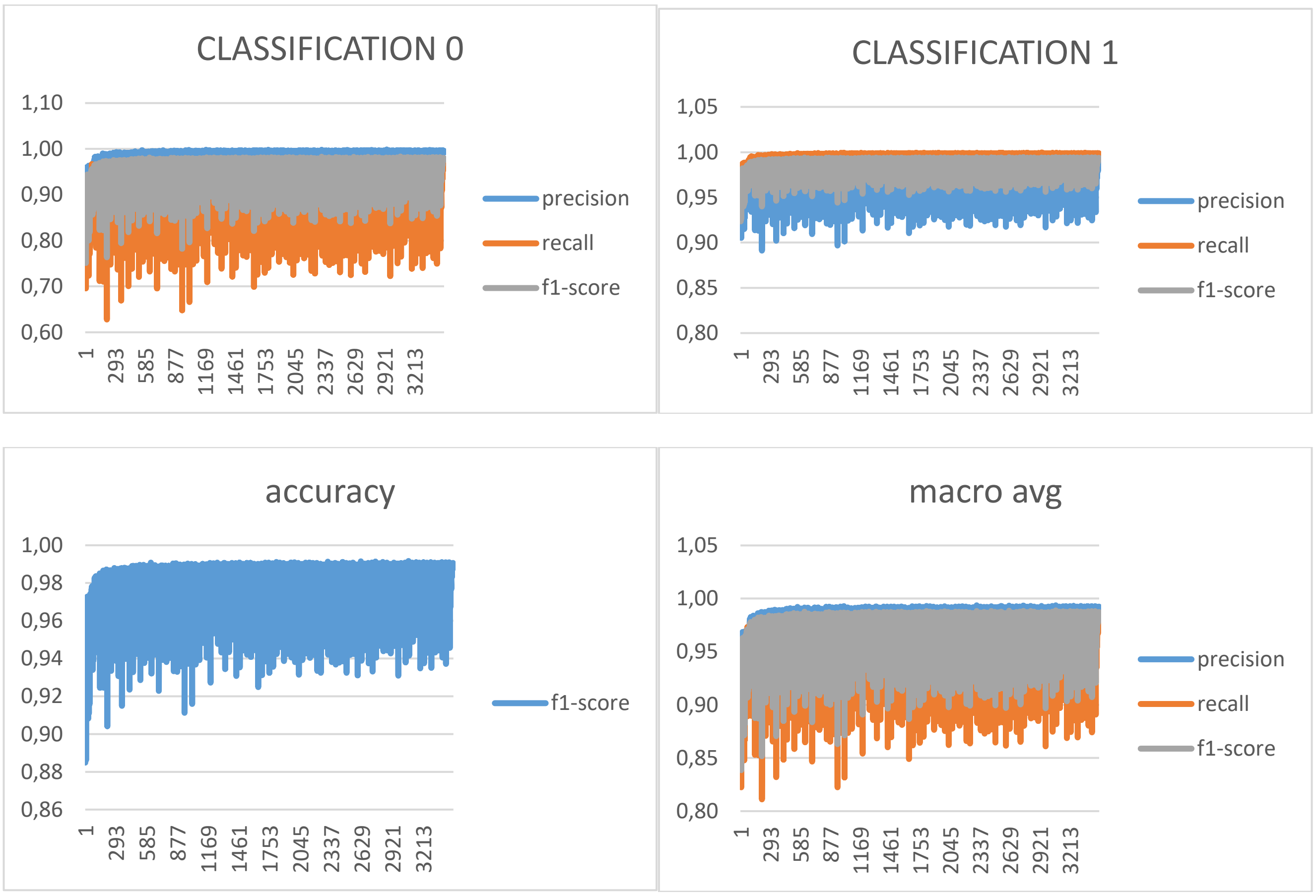


Figure 8. Results for the Decision Tree classifier



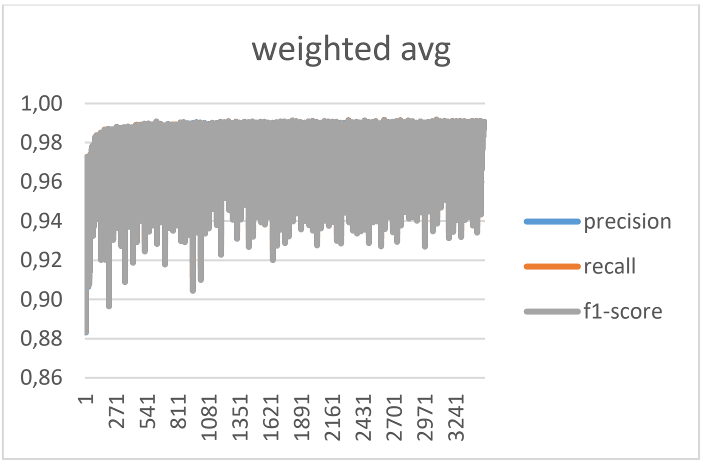
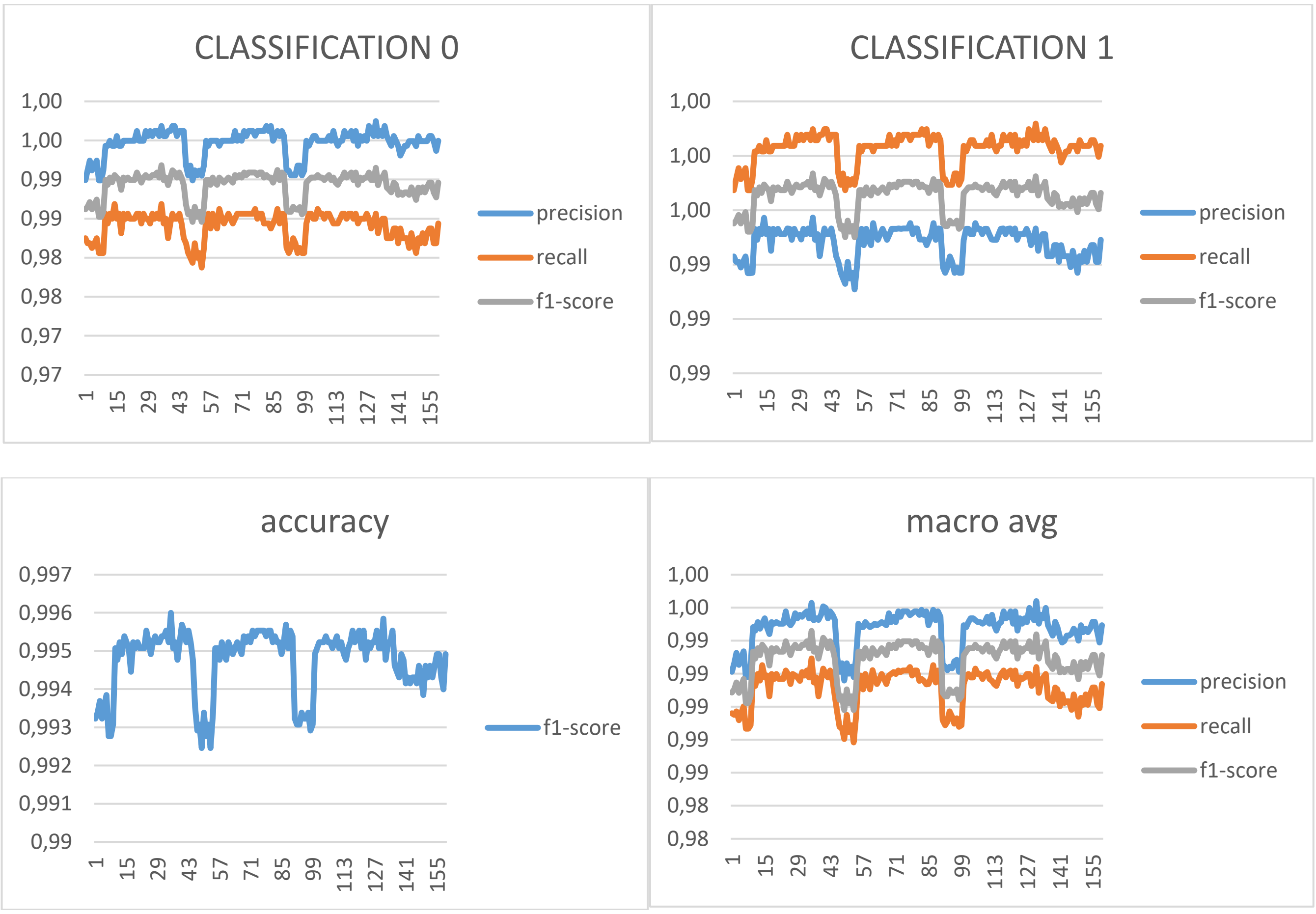


Figure 9. Results for the Random Forest classifier



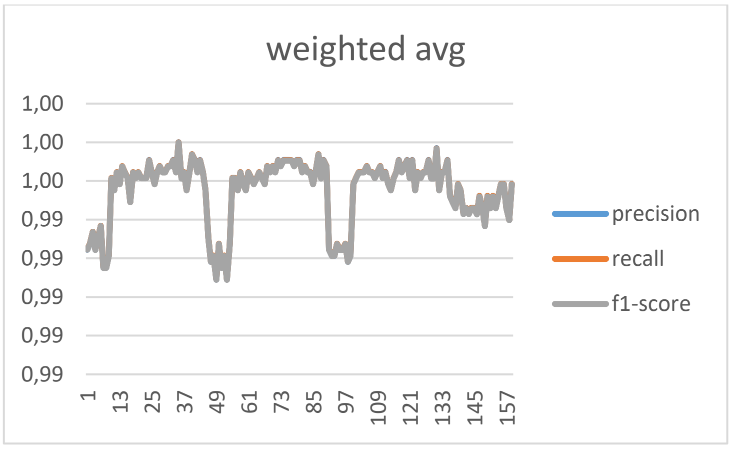
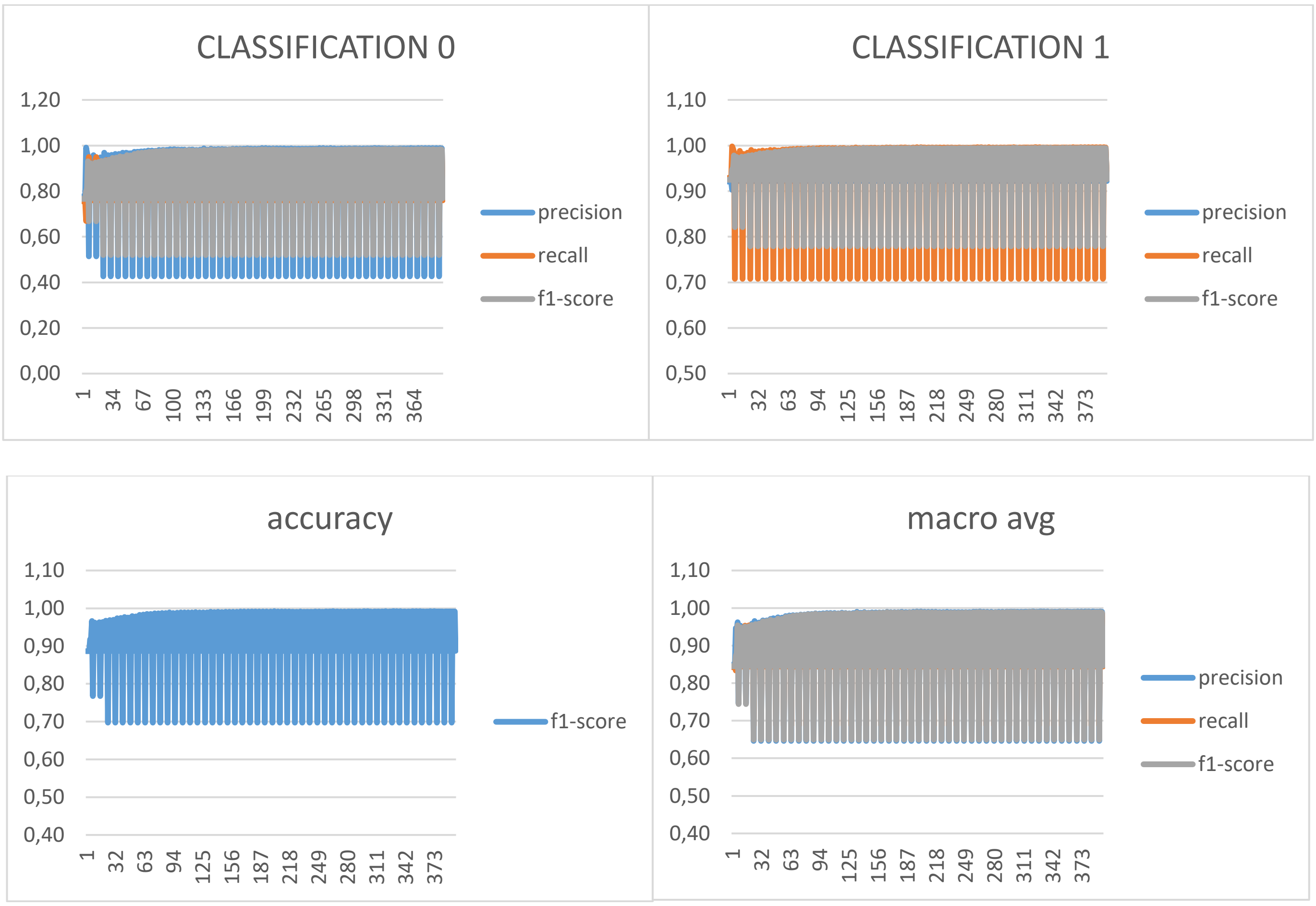


Figure 10. Results for the Neural Net classifier



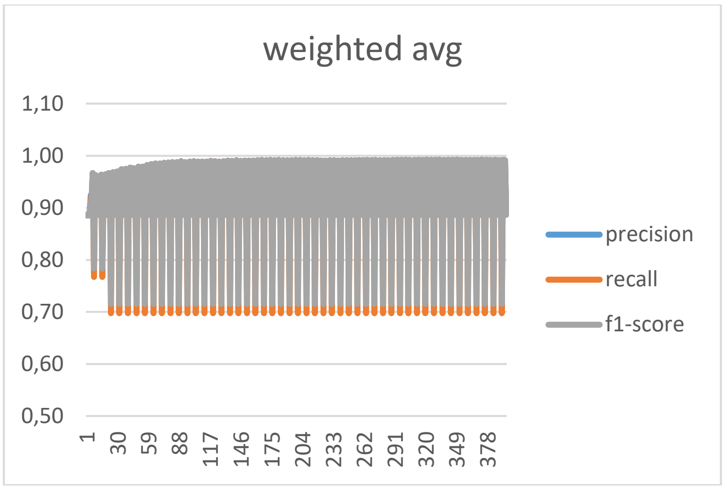
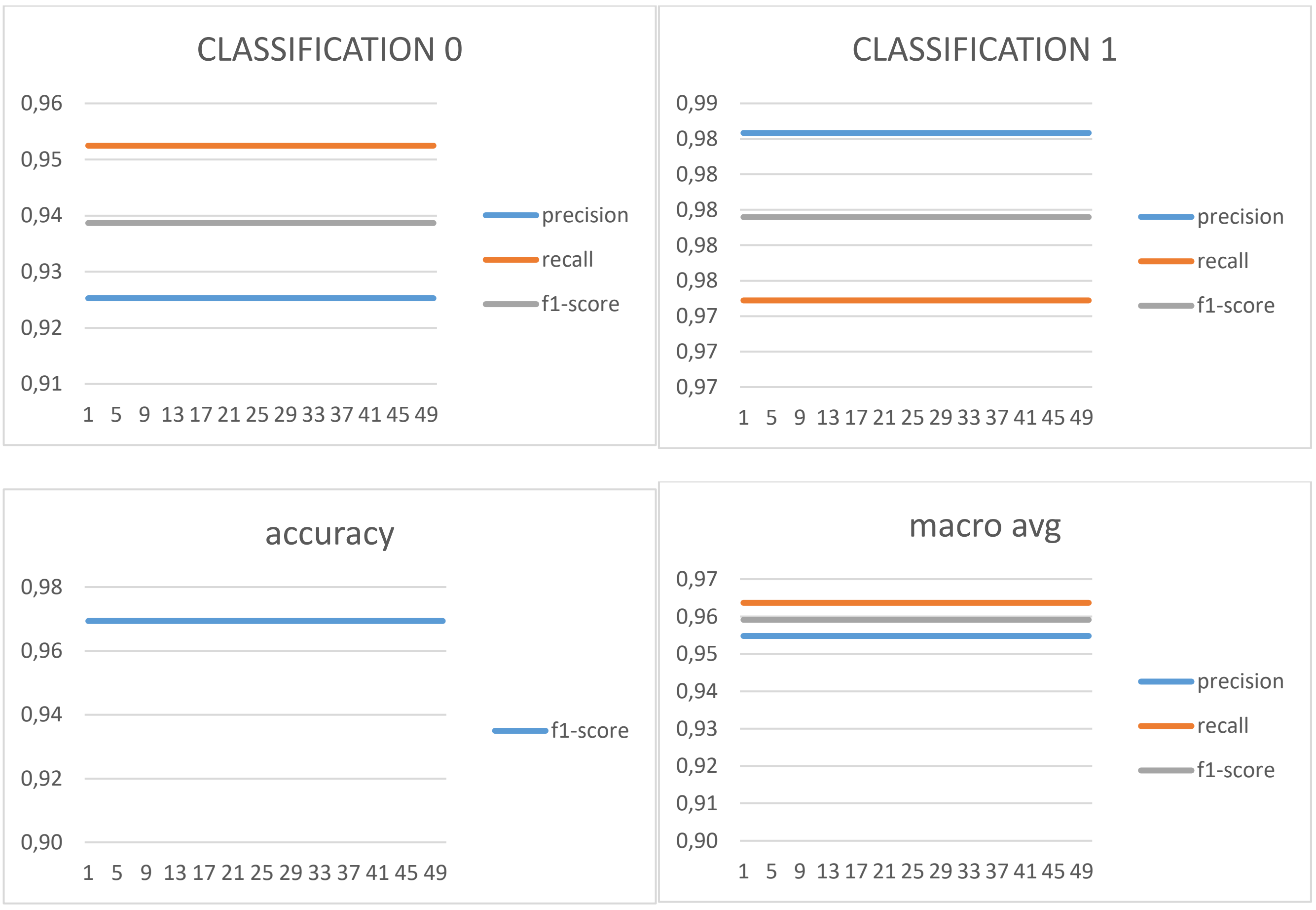


Figure 11. Results for the AdaBoost classifier



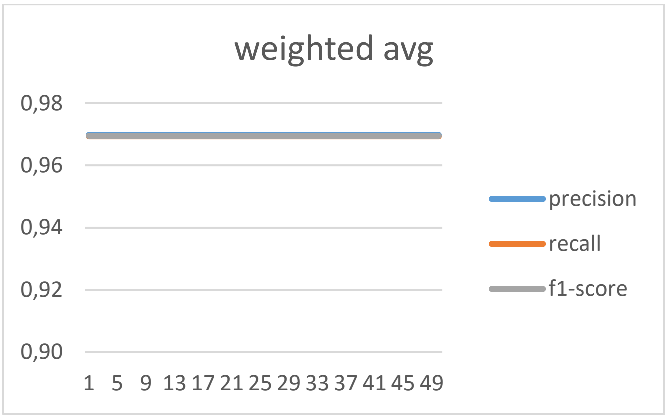
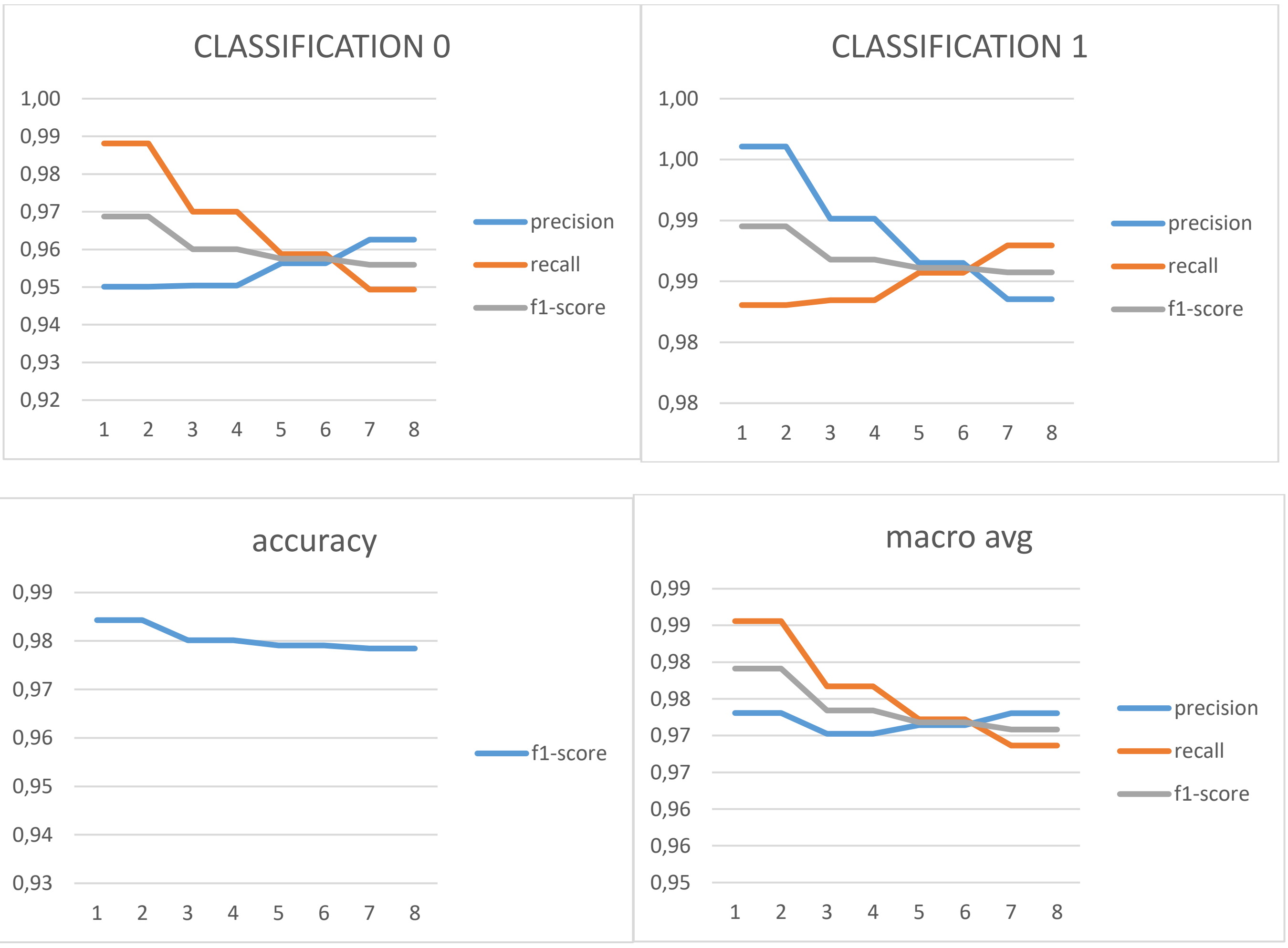


Figure 12. Results for Naive Bayes classifier



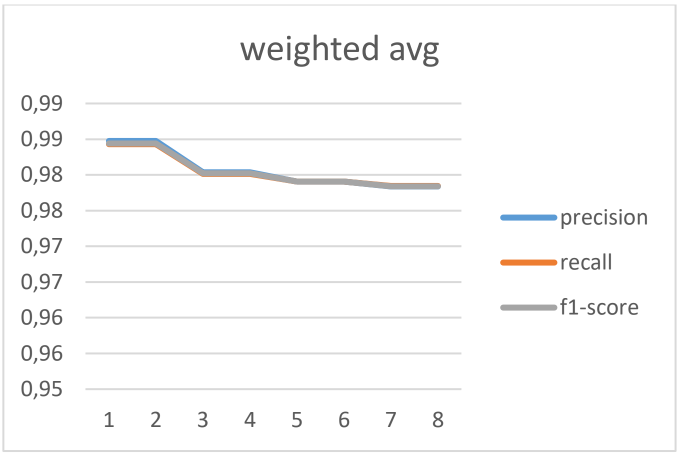


Figure 13. Results for the QDA classifier

The analysis begins by checking the support, according to Table 1.

Table 1. Attributes of each class

Types	Support
0	1599
1	4898
accuracy	6497
macro avg	6497
weighted avg	6497

Analyzing the *support values* for classes (0 and 1), it is perceived that the number of cases for each class is not well distributed; that is, there is an unbalanced database. This indicates that classifiers will be biased in their ranking. But the analysis was made in the database, and the k-fold method was used to balance the data.

Table 2 shows the maximum values found for each metric (accuracy, recall, and f1 score), followed by the respective values for the other metrics and the changed parameter(s) for the classifier.

Table 2. Maximum values for each metric

Classifiers	Accuracy	Recall	F1-score
KNeighbors	0.99	0.99	0.99
Linear SVM Classifier	0.99	0.99	0.99
RBF SVM classifier	0.90	0.88	0.89
Decision Tree	0.98	0.98	0.98
Random Forest	0.99	0.99	0.99
Neural Net	1.00	1.00	1.00
AdaBoost	0.99	0.99	0.99
Naive Bayes	0.97	0.97	0.97
QDA	0.98	0.98	0.98

Some considerations:

1) The precision and recall values for the classes must be greater than or equal to 0.9.
2) In Table 2, you cannot correlate one row with the other since each row brings maximum values for certain metrics. The other metrics will be presented for the relevant lines, if applicable, during the analysis.

Based on the assumptions that the values of precision and recall, considered acceptable, for the classes are greater than or equal to 0.9, it can be observed that most classifiers presented results higher than acceptable ones.

CONCLUSION

We also analyzed the combinations *of recall and precision*, seeking classes with high *recall* and high *precision*. In this combination, the class is ideally treated by the model or *high recall* + low *precision*, in which the class is well detected. Still, the model also includes points of other classes in it.

Initially, the precision and recall values were evaluated to verify that the metric was within the acceptable range, and most classifiers were above the so-called acceptable value of 0.9 for recall and accuracy.

Since most classifiers have reached higher than acceptable values, it can be said that the system developed in Python for the evaluation of the database sounds reasonable because, for several classifiers, the results approached.

It is concluded that except for the RBF SVM classifier, all other classifiers are good and with high confidence in the results for presenting a high performance.

Funding: This research was funded by Suape Energia Group, inside the ANEEL R&D projects.
Acknowledgments: The authors would like to express their thanks to CNPq, and CAPES**.**
Conflicts of Interest: The authors declare no conflict of interest**.**

REFERENCES

1. Arvind D, Khushdeep S, Deepak K. Condition monitoring of power transformer: A review. Proceedings of the 2008 IEEE/PES Transmission and Distribution Conference and Exposition; 2008 Apr 22-24, Chicago, IL. Piscataway (NJ): IEEE Press, c2008. p. 1-6. doi: 10.1109/TDC.2008.4517046.
2. Yuezhong L, Xiaoqiang Y, Piaoxia Z. Research on On-Line Monitoring and Fault Recognition Technology of Intelligent Power Transformer Based on the Internet. Proceedings of the 2018 Chinese Autom. Congr.; 2018 Nov. 30-Dec. 02, Xian, China. Piscataway (NJ): IEEE Press, c2019. p. 4227–4231. doi: 10.1109/CAC.2018.8623269.
3. De Moraes CHV, Boas JLV, Lambert-Torres G, Andrade GCC, Costa CIA. Intelligent Power Distribution Restoration Based on a Multi-Objective Bacterial Foraging Optimization Algorithm. Energies. 2022; 15 (paper 1445): 1-23. doi: 10.3390/en15041445, 2022.
4. Poyser TD, Lenderking BN, Templeton JB. Online Monitoring of Power Transformers. IEEE Trans. Power Appar. Syst., 1985 Jan; PAS-104(1): 207-11. doi: 10.1109/TPAS.1985.318915.
5. Barreto NEM, Rodrigues R, Schumacher R, Aoki AR, Lambert-Torres G. Artificial Neural Network Approach for Fault Detection and Identification in Power Systems with Wide Area Measurement Systems. J. Cont Aut Elect Syst. 2021; 32: 1617-26. doi: 10.1007/s40313-021-00785-y.
6. Machado M, Silva VA, Blasi TM, Kuster KK, Aoki AR, Fernandes TSP, Lambert-Torres G. Recent Research and Development of Microgrids in Parana. Braz Arch Biol Tech. 2021; 64 (e21210177): 1-15. doi: 10.1590/1678-4324-75years-2021210177.
7. Fujimoto Y, Ono T, Works I. Operation of an online substation diagnosis system. IEEE Trans. Power Deliv. 1988; 3(4): 1628-35. doi: 10.1109/61.193964.
8. Veloso GFC, Borges da Silva LE, Noronha I, Lambert Torres G. Using partial discharge as sample signal source to identify contamination moisture pattern in power transformer insulating oil. Proceedings of the 2010 Industrial Electron. Conf.; 2010 Nov 7-10, Glendale, AZ. Piscataway (NJ): IEEE Press, c2010. p. 1041-44. doi: 10.1109/IECON.2010.5675511.
9. Cunninghamacm P, Delany SJ. k-Nearest Neighbour Classifiers - A Tutorial. ACM Computing Surveys. 2021 Jul; 54(6), Paper 128. doi: 10.1145/3459665.
10. Jakkula V. Tutorial on Support Vector Machine [Internet]. Pullman (SA): Washington State University, School of EECS; 2018 [cited 2022 May 17]. Avalilable from: https://course.ccs.neu.edu/cs5100f11/resources/jakkula.pdf.
11. Razaque A, Ben Haj Frej M, Almi’ani M, Alotaibi M, Alotaibi B. Improved Support Vector Machine Enabled Radial Basis Function and Linear Variants for Remote Sensing Image Classification. Sensors.2021;21 (Paper 4431):1-26. doi: 10.3390/s21134431.
12. Charbuty A, Abdulazeez A. Classification Based on Decision Tree Algorithm for Machine Learning. J Appl Sc Tech Trends. 2021 Mar; 2(1): 20-8. doi: 10.38094/jastt20165.
13. Rodriguez-Galianoa VF, Ghimireb B, Roganb J, Chica-Olmoa M, Rigol-Sanchez JP. An assessment of the effectiveness of a random forest classifier for land-cover classification. ISPRS J Photog Rem Sens. 2012 Jan; 67: 93-104. doi: 10.1016/j.isprsjprs.2011.11.002.
14. Mohan Rai H., Chatterjee K. A unique feature extraction using MRDWT for automatic classification of abnormal heartbeat from ECG big data with Multilayered Probabilistic Neural Network classifier. Appl Soft Comp. 2018 Nov; 72: 596-608. doi: 10.1016/j.asoc.2018.04.005.

15. Zhao Y, Gong L, Zhou B, Huang Y, Liu C. Detecting tomatoes in greenhouse scenes by combining AdaBoost classifier and colour analysis. Biosyst Eng. 2016 Aug; 148: 127-37. doi: 10.1016/j.biosystemseng.2016.05.001.
16. Sumanto Y, Sugiarti Y, Supriyatna A, Carolina I, Amin R, Yani A. Model Naïve Bayes Classifiers For Detection Apple Diseases. Proceedings of the 2021 9th International Conference on Cyber and IT Service Management; 2021 Sep 22-23, Bengkulu, Indonesia. Piscataway (NJ): IEEE Press, c2008. p. 1-4. doi: 10.1109/CITSM52892.2021.9588801.
17. Tharwat A. Linear vs. quadratic discriminant analysis classifier: a tutorial. Int J App Patt Recog. 2016; 3(2): 145-80. doi: 10.1504/IJAPR.2016.079050.