Object Classification using Supervised Machine Learning

Master Information Technology
Machine Learning
Arinze Okechukwu Okpagu
Matriculation Number:1324112
arinze.okpagu@stud.fra-uas.de

Master Information Technology
Machine Learning
Emenike John Uchechukwu
Matriculation Number:1324701
john.emenike@stud.fra-uas.de

Master Information Technology
Machine Learning
Daniel Okoyo
Matriculation Number:1324604
daniel.okoyo@stud.fra-uas.de

Master Information Technology
Machine Learning
Bolutife Adisa
Matriculation Number:1400418
bolutife.adisa@stud.fra-uas.de

***Abstract*—This project explores the use of supervised machine learning for real-time object detection and classification. The SRF02 Automotive ultrasonic sensor and Red Pitaya are used as the electronic hardware tools. The model is trained using labeled data, where the correct labels are known beforehand. The focus of the project is to differentiate between an empty seat and a human subject based on the profile of the reflected ultrasonic waves. Three machine learning models - Support Vector Classifier, Decision Tree Classifier, and Multilayer Perceptron Classifier - are used on the same dataset to evaluate their performance. Hyperparameter tuning is also explored to find optimal results per model.**

***Keywords*— Machine Learning (ML), Support Vector Machine (SVM), Multi-layer Perceptron (MLP), Decision Tree (DT), Supervised learning.**

I. INTRODUCTION

Object detection has advanced significantly in recent times with the introduction of AI and machine learning, resulting in increasingly sophisticated and popular systems. Deep learning algorithms have improved the accuracy, efficiency, and scalability of object detection systems, enabling identification and localization of objects in images or videos. This has led to a range of applications in fields such as self-driving vehicles, robotics, surveillance, and security [1]. The benefits of these advanced object detection systems are enormous, and they have revolutionized industries that require precise visual data monitoring or analysis. For instance, security systems now use object detection to identify potential threats such as weapons or suspicious objects. Autonomous vehicles rely on object detection to identify and avoid obstacles, pedestrians, and other vehicles, while robotics applications use object detection to navigate dynamic environments and interact more effectively with humans [2]. Earlier object detection systems relied on handcrafted features and complex algorithms. However, AI-based systems have made significant strides in human detection using techniques like body part detection, silhouette-based detection, and pose estimation to accurately detect humans [3]. For example, some systems use pose estimation to detect humans by identifying key points such as joints and body parts, enabling them to detect abnormal movements or postures.

Machine learning is a powerful tool that leverages sophisticated algorithms to enhance decision-making capabilities. These sophisticated algorithms can learn from data without being explicitly programmed to do so. Machine learning models are trained on a dataset, which is a collection of examples, and they use statistical techniques to identify patterns and relationships within the data. Once the model has been trained, it can be used to make predictions on a new data that it has not seen before.

Machine learning can differentiate between soft and hard objects using an ultrasonic sensor. It achieves this analyzing the echoes returned by the sensor when it emits ultrasonic waves towards the object. The time taken for the echo to return to the sensor and its strength can provide information about the object's distance, size, and hardness. Soft objects are objects that are pliable or made of flexible materials such as clothing or fabrics. Hard objects, on the other hand, are objects that are rigid and typically made of materials such as wall, wood, plastics, or metals. Object detection systems can be trained to detect both soft and hard objects using machine learning algorithms such as a support vector machine (SVM), neural network, or decision tree. In the case of differentiating between summer and winter clothes due to thickness or make of the material, object detection systems can be trained to identify these differences. For example, a system can be trained to detect thicker and heavier clothing items such as jackets and coats as winter clothing, while thinner and lighter clothing items such as t-shirts and shorts can be identified as summer clothing. Depending on the use case, Machine learning offers a unique opportunity to gain insights into diverse types of data patterns.

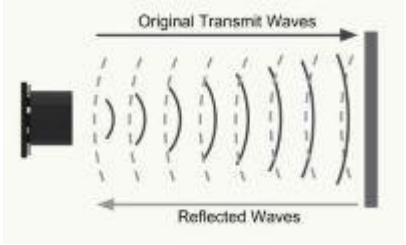


Fig. 1. Ultrasonic sensor Transducer

II. ULTRASONIC SENSOR AND RED PITAYA

A. *Ultrasonic Sensor*

An ultrasonic sensor is a device that uses sound waves with frequencies above the human audible range to detect the distance to, or presence of, an object. These sensors emit high-frequency sound waves, which bounce off an object and return to the sensor. The time it takes for the sound waves to return to the sensor is used to calculate the distance to the object.

Ultrasonic sensors are used in a variety of applications, including robotics, automation, and security systems, and are especially useful in environments where other sensors may not work well due to factors like light interference or dust. The other key benefits includes:

i. Non-contact measurement: they can measure distances without the need for physical contact. This means they can be used in applications where contact sensors would be difficult or impossible to use, such as measuring the level of liquid in a tank.

ii. High Accuracy: they are highly accurate and can measure distances with an accuracy of up to 0.1% of the measured distance. This makes them ideal for applications that require precise measurements.

iii. Robustness: they demonstrate great ability to detect objects regardless of their color or texture.

iv. Low cost: they are relatively very low cost compared to other sensors such as radar and can be easily integrated with microcontrollers and other devices.

There are still worries that the accuracy of ultrasonic sensors may be impacted by shifts in temperature or wind. Furthermore, having several ultrasonic sensors working close to each other may result in interference, leading to imprecise distance measurements [5].

B. *Red Pitaya*

Redpitaya is a low-cost open-source software-defined radio (SDR) platform used for creating different measurement and test applications. It operates on a field-programmable gate array (FPGA) which can be programmed using different tools such as Python, C, or MATLAB. The platform has a wide range of applications, including signal generation, signal processing, and analysis. Red Pitaya is very versatile, and is designed to be a cost-effective alternative to other SDR platforms. It offers high-performance capabilities, and its affordability makes it accessible to many hobbyists and researchers. The flexibility and affordability of Red Pitaya make it an attractive option for those looking to experiment with various signal processing techniques and for exploring new applications [6].

Ultrasonic sensors combined with Red Pitaya are commonly used for distance measurement, object detection, and level sensing in various applications such as robotics, industrial automation, and home automation. Red Pitaya can process the data from the sensor and display the distance measurements in real-time. Additionally, the combination can enable object detection, allowing for the triggering of alarms or events when objects are detected. This is beneficial in parking assistance systems, automated doors, and security systems, among others.

III. SYSTEM ARCHITECTURE

Here, the description of the FIUS Sonar Sensor Project which considers parameter setting and reliability test of a sensor system for person detection in a car wearing winter clothes is provided.

A. *UltraSonic Sensor and RedPitaya*

The hardware system provisioned for the project utilized a small Ultrasonic sensor integrated with RedPitaya for detecting people and objects. The SRF02 sensor has a minimum measurement range of about 20cm (7.9 inches), and it can be powered by a 5V grounded power supply. The SRF02 uses a single transducer for both transmission and reception [4]. The Red Pitaya wirelessly transmits the collected data for further analysis through an end user application connected to a PC already paired to the RedPitaya with Wi-Fi.

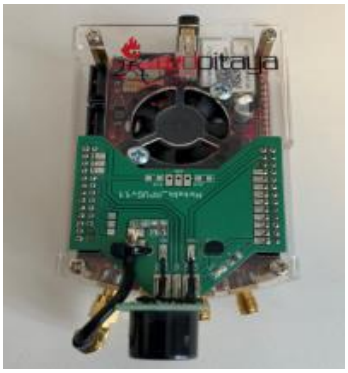


Fig. 2. Ultrasonic Sensor integrated with redpitaya

To be able distinguish between an empty seat and one occupied by a human subject wearing winter clothes, the system examines the frequency profile of ultrasonic waves. When the waves reflect off hard surfaces, they lose less energy, whereas softer surfaces absorb some of the waves, leading to greater energy loss. The system uses the distinct frequency profile generated by these differences to identify whether a seat is occupied.

Fig. 3. Car for the Experiment

The sensor was placed on the car dashboard, facing the passenger seat and to the right of the steering wheel. During the experiment, the distance and angle between the sensor and other objects were adjusted to consider various conditions. The car was kept stationary, and the engine was not running while data was collected.

B. *Object Profiles*

The study examined different human profiles, see Table I, and dummies, all wearing different winter clothing. Each profile was subjected to a variety of conditions as described in Table II. An empty seat profile was also used as a key parameter for training the models to detect when there is no person in the car. It was essential to ensure that the car remained closed and that the sensor's line of sight was not disturbed by any random movements during the measurement for the empty seat profile.

TABLE I. Human Subject Profiles

	Sex	Skin Color	Hair Color	Build	Height (cm)
#1	M	Brown	Black	Ectomorph	180
#2	M	Brown	Black	Mesomorph	175
#3	M	Brown	Black	Mesomorph	170
#4	M	Brown	Black	Ectomorph	185

TABLE II. Observation Scenarios for Dataset

	Object Profiles						
	Gestures		Winter gears			Seat Position	
Events	M	W/O M	HW	GL	SB	Seat angle	Seat Dist. (cm)
#1	X	-	X	-	-	90º	42
#2	X	-	-	X		45º	42
#3	X	-	-	-	X	90º	32
#4	-	X	X	-	-	90º	42
#5	-	X	-	X	-	45º	42
#6	-	X	-	-	X	90º	32
Table Legend	X: active, - : not active M: With Movement, W/O M: Without Movement HW: Head warmer, GL: Gloves, SB: Seat Belt Seat Dist.: Seat distance from the Ultrasonic sensor						

C. *Winter Outfits and Measurement plot*

The study included the following winter clothing and gears for the measurement: sweaters, hoodies, jackets, scarves, and head warmers. These items were taken into consideration to ensure that the different profiles were properly outfitted for the winter conditions.

In addition to varying winter outfits, each measurement was modified to simulate real-time scenarios such as a person sitting still in the car, a person with consistent gestures and body movements, and different dummy scenarios such as a winter jacket draped over the chair. These modifications allowed the study to accurately reflect certain real-world situations.

D. *Measurement Software and Data Format*

To retrieve data from the sensor, it was pertinent to make use of the software program already developed for this purpose by the department. The application allowed for different numbers of observations to be recorded for each scenario. For this project, the client was set up to collect 500 observations per scenario. The data from each observation was saved as ADC and FFT data, which are the two main data formats used for recording the readings from the sensor. All the data was labelled and saved according to the experiments for further analysis. The software was essential for collecting the data and for performing real-time analysis of the data patterns under different conditions.

The FFT data from the observations was saved as text (.txt) files and preprocessed to remove unnecessary device information. Each text file was then converted to the Open XML format (.xlsx) and a new column called 'LABEL' was added to indicate the label of each observation. Readings with people in the car were labelled as '1', while those of dummies and empty seats were labelled '0' as described in Table III. It was important to keep the class categories to two choices only, though it is possible in some cases to train the model on multiple classes. The data fed to the model consists of multidimensional arrays of labels paired with multiple features across 85 columns.

TABLE III. Dataset Description

	Label	Object Profile	Total Data per Object
1	1	Human Subject	41,916
2	0	Dummy	6,986
3	0	Empty Seats	6,986
Total Data			55,888

IV. Evaluation Metrics

Given that the project was solely focused on classification, the task began by training multiple models on a specific subset of the dataset. Our objective was to gain a better understanding of the various patterns that were observed during each iteration. This approach allowed for comprehensive evaluation of the performance of each model on a separate test set using critical evaluation metrics such as accuracy, precision, recall, and F1 score. Based on this analysis, the weaker models were eliminated and the top three performing models shortlisted for further investigation.

To analyze and detect patterns in the training data, three distinct machine learning models, namely Support Vector Classifier, Multilayer Perceptron Classifier, and Decision Tree Classifier, were utilized. These models were used make predictions on a new data, a subset of the dataset which was not seen during the training.

The following evaluation metrics provided basis for comparing the performances of each model.

A. Confusion Matrix

This is a table that helps evaluate the performance of a classification model by comparing the predicted and actual results. It provides the foundation to assess accuracy, precision, recall, and F1 score. The matrix has four quadrants that represent the four possible outcomes, including:

a. True Positive (TP), which means the model predicted the outcome correctly.

b. False Positive (FP), which means the model incorrectly predicted a positive outcome when it should have been negative.

c. True Negative (TN), The model accurately predicted a negative outcome.

d. False Negative (FN), The model incorrectly predicted a negative outcome when it should have been positive.

In the confusion matrix, the rows correspond to the true class labels and the columns correspond to the predicted class labels for a set of samples. The total count of all samples is calculated by adding up all the values present in the matrix.

		Actual	
		CLASS 1	NOT CLASS 1
Predicted	CLASS 1	True Positive	False Negative
	NOT CLASS 1	False Positive	True Negative

Fig. 4. Confusion Matrix

B. Accuracy

This is a metric that measures how well the model predicts the correct class labels.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Total\ correct\ predictions = TP + TN \quad (2)$$

$$Total\ predictions = TP + TN + FP + FN \quad (3)$$

C. Precision

This measures how many of the predicted positive instances are exactly positive.

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

D. Recall

This measures how many of the actual positive instances the model correctly predicts as positive.

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

E. F1-Score

This is a weighted average of precision and recall and is a useful metric for imbalanced datasets. In other words, it is the harmonic mean between precision and recall.

$$F1 - Score = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall}\right) \quad (6)$$

F. Cross-Validation Score

Cross-validation is a technique used to evaluate the performance of a model. It is a measure of how well the model generalizes to new data. Cross-validation involves splitting the data into multiple subsets, or "folds", and training the model on each subset while evaluating its performance on the remaining data. This process is repeated multiple times, with each fold serving as the test set for a different iteration.

The cross-validation score is then calculated as the average of the performance scores across all the folds. This provides a more reliable estimate of the model's performance than simply evaluating it on a single training-test split, as it allows for a more comprehensive evaluation of the model's ability to generalize to new data.

The scikit-learn library [7] was used to provide the cross-validation function implemented for this task.

V. MODEL TRAINING AND HYPERPARAMETER TUNING

Before training the models, the data was further preprocessed by standardizing the data. Standardizing data proves a crucial pre-processing step for machine learning to ensure that different features contribute equally to the analysis, helps to keep the data free from any form of bias, reduce complexity, and affords models to learn more effectively. The data normalization was made possible using the *StandardScaler* function of the scikit-learn python library.

A. Training Support Vector Classifier

The support vector classifier (SVC), also known as support vector machine (SVM) is a type of supervised learning algorithm used for classification analysis. It creates a linear boundary in an enlarged feature space to detect non-linear boundaries. The model identifies the optimal hyperplane that separates the dataset into classes. The data points closest to the decision plane, also referred as support vectors, play a critical role in SVM's accuracy and performance. SVMs are well-known for their reliability and precision [8].

SVMs can tackle linear and non-linear problems thanks to different kernel functions. These kernels transform data into higher-dimensional spaces that allow for data separation via a hyperplane. The hyperplane is used to establish boundaries for different categories in the data. SVMs quickly classify incoming data by locating its position relative to the hyperplane.

In this project, the radial basis function (RBF) kernel, also known as Gaussian kernel, was given special attention. This kernel maps samples into a higher-dimensional space non-linearly, enabling it to detect non-linear relationships between class labels and features, unlike the linear kernel. The RBF

kernel allows SVMs to model intricate decision boundaries and effectively handle high-dimensional data by capturing the underlying structure of the data in a higher-dimensional space.

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \quad \gamma > 0 \qquad (7)$$

Where $K(x_i, x_j)$ defines the kernel function. Gamma (γ), controls the influence of the new features on the decision boundary. x_i, x_j are projected vectors.

Gamma, γ controls the shape of the decision boundary. The choice of gamma is critical to the performance of the RBF kernel. When the value of gamma is high, the decision boundary becomes more complex and can fit the training data more closely. When the value of gamma is low, the decision boundary becomes smoother and less complex, and the model is more likely to generalize well to new data. Selecting the optimal value of gamma requires balancing the trade-off between underfitting and overfitting. Different values of gamma were experimented prior to achieving the optimal result during the training.

It is worth knowing that one other hyperparameter, the regularization Parameter, C controls the trade-off between maximizing the distance between the decision boundary and the closest data points, otherwise known as margin and minimizing the classification error on the training data. When the value of C is high, the SVM algorithm is penalized heavily for misclassifications, and as a result, the model will try to classify all data points correctly. In contrast, when the value of C is low, the SVM algorithm is less penalized for misclassifications, and as a result, the model will try to maximize the margin even if it means misclassifying some of the training data points.This can lead to a model with a small margin, which may not generalize well to unseen data. A low value of C can often lead to underfitting, as the model may not capture the underlying structure of the data .

Fig. 5 shows snippet of the active hyperparameters that could be tuned when training support vector classifier model using the Scikit-learn Python library [7].

```
{
  'C': 1.0,
  'break_ties': false,
  'cache_size': 200,
  'class_weight': None,
  'coef0': 0.0,
  'decision_function_shape': 'ovr',
  'degree': 3,
  'gamma': 'scale',
  'kernel': 'rbf',
  'max_iter': -1,
  'probability': false,
  'random_state': None,
  'shrinking': true,
  'tol': 0.001,
  'verbose': false
}
```

Fig. 5. sklearn.svm.SVC Parameters

The learning curve in Fig. 6 and Fig. 9 shows some of the best performances of the model when different parameter values for the kernel, regularization and gamma are applied.

By setting gamma to *'scale'*, the gamma parameter can be adapted in a way that is appropriate for the dataset [7]. Both learning curves indicates that the model gets better as more data is used for training. The fact that the model converges quickly suggests that it is a good fit for the dataset.

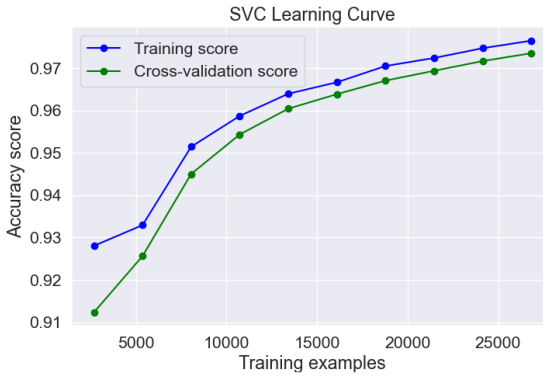


Fig 6. SVC Learning Curve. 'kernel'= 'rbf' 'gamma' = 'scale', 'C'=1.0

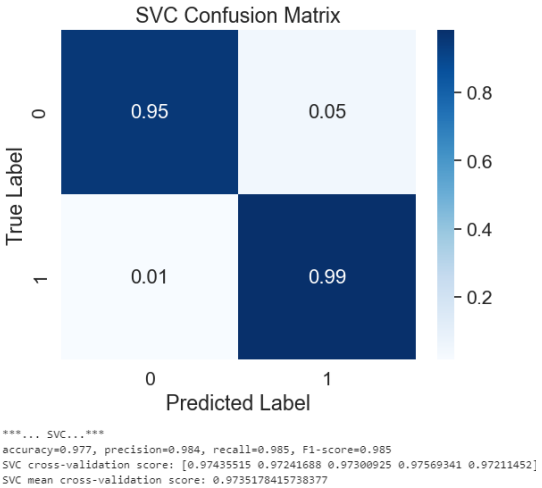


Fig. 7. SVC Confusion Matrix. 'kernel'= 'rbf', 'gamma' = 0.1, 'C'=0.1

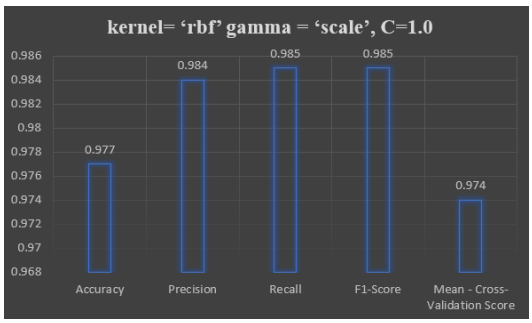


Fig. 8. Evaluation metrics from Fig. 6-7

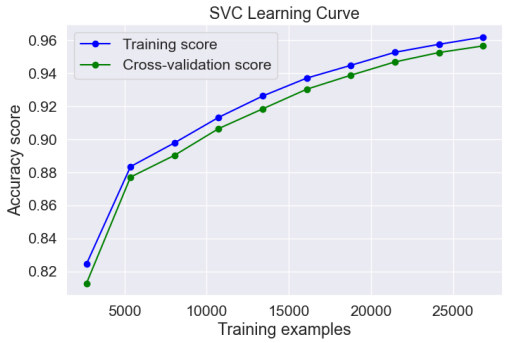
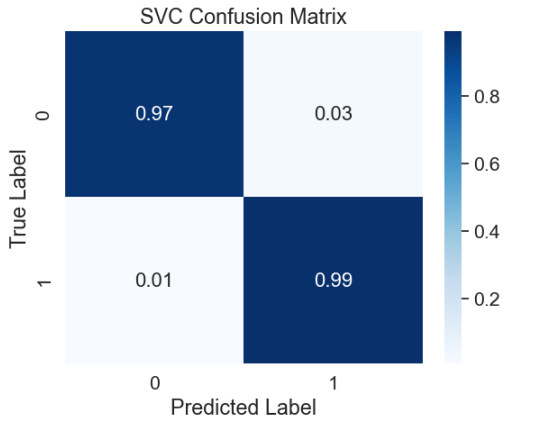


Fig. 9. DT Learning Curve. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =15, 'max_features' = None, splitter='best'




```
***... SVC...***
accuracy=0.989, precision=0.991, recall=0.993, F1-score=0.992
SVC cross-validation score: [0.98673028 0.9856866  0.98464062 0.98687742 0.98434238]
SVC mean cross-validation score: 0.985655460278023
```

Fig. 10. DT Confusion Matrix. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =15, 'max_features' ='sqrt', splitter='best'

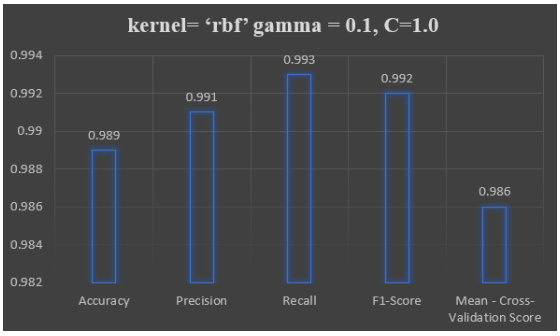


Figure 11. Evaluation Metrics from Fig.9-10

The mean cross-validation score from both demonstrations was slightly lower than the accuracy on the holdout test set but still indicates good performance. Overall, these metrics suggest that the SVM model is performing well with a high degree of accuracy and effectiveness in predicting outcomes on the testing and validation datasets.

B. Training Decision Tree (DT) Classifier

Decision Tree Classifier is another supervised learning algorithm used for classification tasks. It uses a tree-like structure to make decisions based on a set of features starting with a single node, representing the entire dataset, and splitting it into smaller subsets based on the values of selected input features. Each internal node of the tree represents a decision based on a feature, while each leaf node represents a class label [9]. Decision Tree Classifier sets out to create a model that can accurately classify new data points by learning decision rules from the training data. The algorithm does this by recursively splitting the data into subsets based on the feature that provides the most information gain, which is a measure of how much a feature reduces the uncertainty about the class labels. The process continues until all the data is classified or a stopping criterion is met.

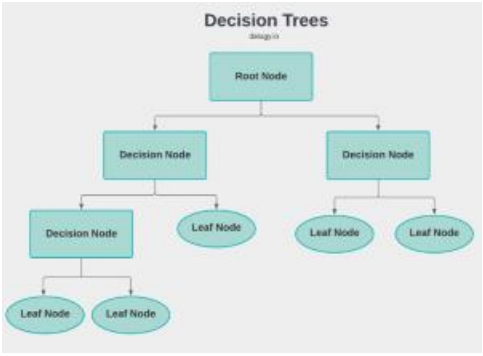


Fig. 12. Decision Tree Flowchart

To train a Decision Tree Classifier model it was important to pay attention to several crucial parameters. These include but are not limited to maximum depth of the tree, the minimum number of samples required to form a leaf node, the maximum number of features used to split the data, the criterion used to determine how to split the data, and the type of splitting algorithm employed.

The maximum depth of the tree is an important hyperparameter to set as it controls how many levels the decision tree can have. This helps prevent overfitting, where the model is too closely tailored to the training data and does not generalize well to new data.

The minimum number of samples per leaf node is also important, as it ensures that each leaf node contains enough data points to be significant. This helps prevent overfitting and ensures that the model is able to accurately classify new data.

The maximum number of features used to split the data is another hyperparameter that can help prevent overfitting and improve generalization. This limits the number of features that are considered for each split, which can help prevent the model from becoming too complex and improve its ability to generalize to new data. The criterion on the other hand determines which feature is selected for each split.

Fig. 13 shows a snippet of the parameters that could be tuned when training a DT classifier model using the Scikit-learn Python machine learning library.

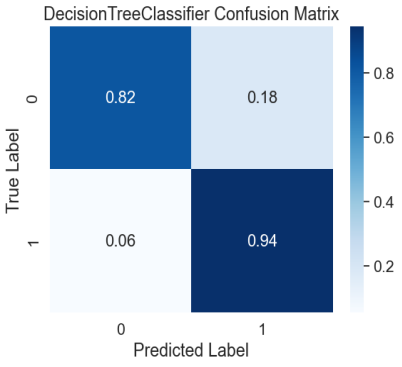
```
{
'ccp_alpha': 0.0,
'class_weight': None,
'criterion': 'gini',
'max_depth': None,
'max_features': None,
'max_leaf_nodes': None,
'min_impurity_decrease': 0.0,
'min_samples_leaf': 1,
'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0,
'random_state': None,
'splitter': 'best'
}
```

Fig. 13. sklearn.tree.DecisionTreeClassifier Parameters

The scikit-learn library employs a tree algorithm called CART (Classification and Regression Trees). This algorithm creates binary trees by selecting the feature and threshold that gives the highest information gain at each node. During the training process, the maximum depth of the tree was set as 'None', which means that the nodes can continue to expand until all the leaves become pure or the minimum number of samples required to split an internal node is not met.

Fig. 14-19, shows some of the best results realized using the DT model.

Fig. 14. DT Learning Curve. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =15, 'max_features' = None, splitter='best'




```
***... DecisionTreeClassifier...***
accuracy=0.912, precision=0.939, recall=0.944, F1-score=0.942
DecisionTreeClassifier cross-validation score: [0.91382138 0.90741017 0.91246645 0.91246645 0.90650164]
DecisionTreeClassifier mean cross-validation score: 0.9105332170727968
```

Fig. 15. DT Confusion Matrix. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =15, 'max_features' ='sqrt', splitter='best'

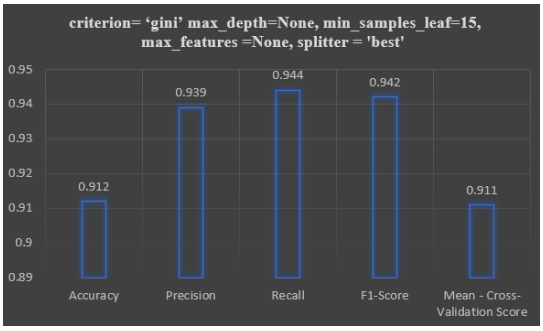


Fig. 16. Evaluation Metrics from Fig. 14-15

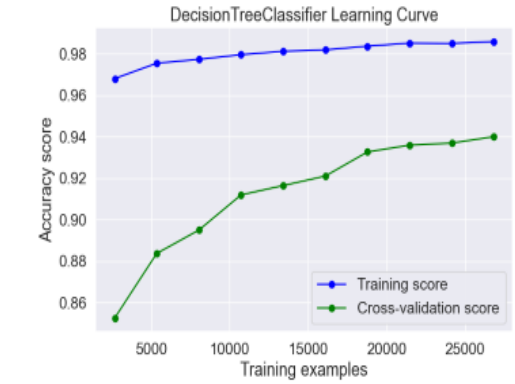
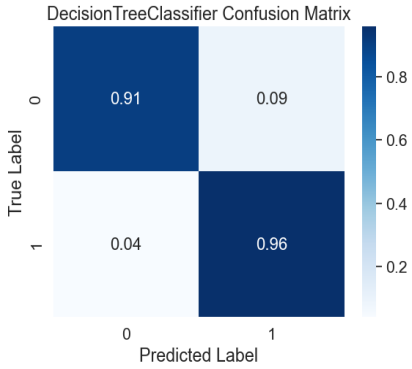


Fig. 17. DT Learning Curve. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =5, 'max_features' =None, splitter='best'




```
***... DecisionTreeClassifier...***
accuracy=0.946, precision=0.968, recall=0.959, F1-score=0.964
DecisionTreeClassifier cross-validation score: [0.93857164 0.93931713 0.94333433 0.94258873 0.93781688]
DecisionTreeClassifier mean cross-validation score: 0.9403257414621853
```

Fig. 18. DT Confusion Matrix. 'criterion'='gini', max_depth=None, 'min_samples_leaf' =5, 'max_features' =None, splitter='best'

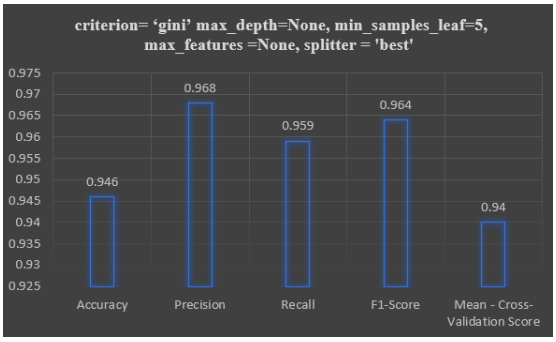


Fig. 19. Evaluation Metrics from Fig. 17-18

Though the metrics from both sample demonstrations indicates that the decision tree classifier model is performing well in predicting the target variable, it is worth knowing the impact of the '*min_samples_leaf'* parameter value as can be seen from Fig. 16 and 19. On adjusting the value from *15 to 5*, the metrics improved slightly across the board for all the evaluation metrics implemented. The cross-validation scores of *'[0.91382138, 0.90741017, 0.91246645 0.91246645 0.90650164]'* and *'[0.93857164, 0.93931713, 0.94333433, 0.94258873 0.93781688]'* respectively from Fig. 15 and Fig. 18 indicate that the model performed consistently well across different folds of the test set.

C. *Training Multilayer Perceptron Classifier*

The Multilayer Perceptron (MLP) is a type of deep neural network that consists of multiple layers, including an input layer, output layer, and one or more hidden layers. These layers are composed of interconnected neurons, where each neuron in one layer is connected to all neurons in the next layer. The input layer receives the features of the input data, and the output layer produces the predicted output. Each neuron in the MLP uses an activation function to produce an output, which is then passed to the next layer [10].

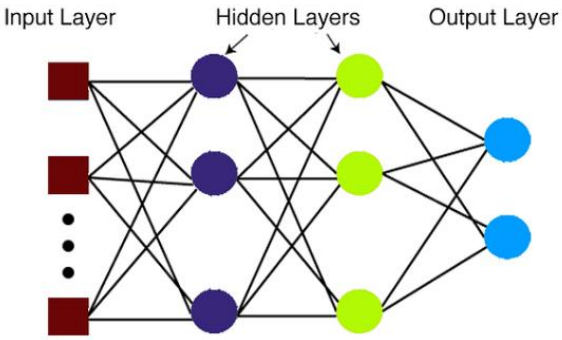


Fig. 20. Multilayer Perceptron Architecture

The perceptron works by taking in multiple real-valued inputs and calculating a weighted sum, which is then transformed by a nonlinear activation function to produce a single output value. Equation 8 provides a mathematical representation and interpretation for the given concept.

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi(w^T x + b) \tag{8}$$

Here, w denotes the vector of weights, x is the vector of inputs, b is the bias and phi, φ is the non-linear activation function.

During training of an MLP model, the weights of the connections between the neurons are adjusted to minimize the difference between the predicted output and the actual output. This is typically done using the Stochastic Gradient Descent (SGD) optimization algorithm, which finds the set of weights in the MLP that minimize the error between the predicted output and the actual output. In comparison to SGD, Adaptive Moment Estimation (Adam), an adaptive optimization algorithm, shows to be less sensitive to hyperparameter tuning, and computationally efficient [11,12].

Apart from the weight optimization parameter, the other parameters that proves critical for tuning the model includes but are not limited to:

i. Activation: The activation function determines the output of each neuron in the MLP. It takes the inputs from the previous layer, processes them, and produces an output through a non-linear transformation.

ii. Learning rate: The learning rate determines the size of the steps taken for the optimization process. It determines the speed at which the weights of the MLP are adjusted during training. A too high learning rate may cause the learning process to diverge, and a too low learning rate may cause the learning process to take too long to converge.

iii. Maximum Iterations: Maximum iterations determine the maximum number of iterations the optimizer should run before terminating. It is used to prevent the optimizer from running indefinitely.

iv. Hidden Layer: Increasing the number of hidden layers can increase the complexity of the model, allowing it to learn more complex patterns. Too many hidden layers can lead to overfitting. It could be inferred that the model has passed the optimal number of layers when the performance of the model no longer improves even as more layers are added.

Fig. 21 shows a snippet of the parameters that could be tuned when training a decision tree classifier model using the scikit-learn python library [7].

```
{
'activation': 'relu',
'alpha': 1e-05,
'batch_size': 'auto',
'beta_1': 0.9,
'beta_2': 0.999,
'early_stopping': false,
'epsilon': 1e-08,
'hidden_layer_sizes': [500],
'learning_rate': 'constant',
'learning_rate_init': 0.001,
'max_iter': 500,
'momentum': 0.9,
'n_iter_no_change': 10,
'nesterovs_momentum': true,
'power_t': 0.5,
'min_impurity_decrease': 0.0,
'max_fun': 15000,
'min_samples_split': 2,
'shuffle': true,
'solver': 'adam',
'tol': 0.0001,
'validation_fraction': 0.1,
'verbose': false,
'warm_start': false
}
```

Fig. 21. sklearn.neural_network.MLPClassifier Parameters

Fig. 22-27 shows some of the performances of the MLP Classifier model. The learning curve graphs implies a strong fit, with the training score and cross-validation score converging as the size of the training sample increased. The cross-validation scores suggest that the model's performance is consistent across different folds of the holdout data. This further suggests that the model is likely to generalize well to new, unseen data. Overall, the results indicates that the MLPClassifier model is a good candidate for the given classification task and may perform well in practice.

There are cases where the specified maximum number of training iterations, *max_iter*, could cause the optimization process not to converge to an optimal solution. Fig. 28 shows a sample of the convergence warning during the training using the Stochastic Gradient Descent (SGD) optimizer. The warning message indicates that the maximum number of training iterations has been reached, but the optimization

process has not yet converged. This can happen if the model has not reached an optimal set of weights or parameters, which can result in suboptimal performance on test data.

It is important to monitor the performance of the model during training and adjust the algorithm parameters as needed to achieve the desired level of convergence. To address the '*ConvergenceWarning'*, it is worthwhile to adjust the learning rate or regularization parameters or increasing the maximum number of iterations to allow the optimizer to continue its search for a better solution.

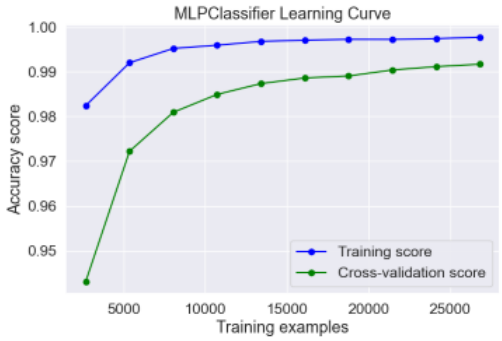
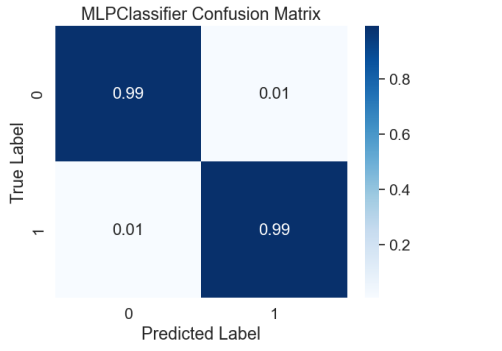


Fig. 22. MLP Learning Curve. 'activation'='relu', 'learning_rate'='constant', 'max_iter'=500, 'hidden_layer_sizes'=(300,)




```
***... MLPClassifier...***
accuracy=0.993, precision=0.996, recall=0.994, F1-score=0.995
MLPClassifier cross-validation score: [0.99314149 0.99090502 0.99269311 0.99209663 0.99224575]
MLPClassifier mean cross-validation score: 0.9922164018336247
```

Fig. 23. Confusion Matrix. 'activation'='relu', 'learning_rate'='constant', 'max_iter'=500, 'hidden_layer_sizes'=(500,)

Fig. 24. Evaluation Metrics from Fig. 22-23

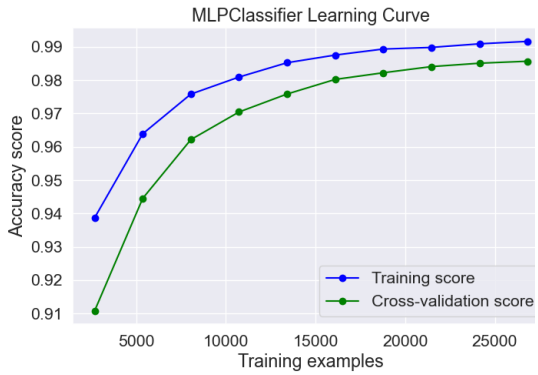
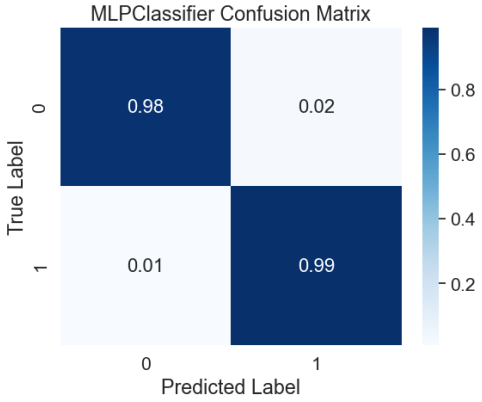


Fig. 25. MLP Learning Curve. 'activation'='relu', 'learning_rate'='constant', 'max_iter'=200, 'hidden_layer_sizes'=(500),




```
***... MLPClassifier...***
accuracy=0.988, precision=0.993, recall=0.991, F1-score=0.992
```

Fig. 26. MLP – Confusion Matrix 'activation'='relu', 'learning_rate'='constant', 'max_iter'=500, 'hidden_layer_sizes'=(200,)

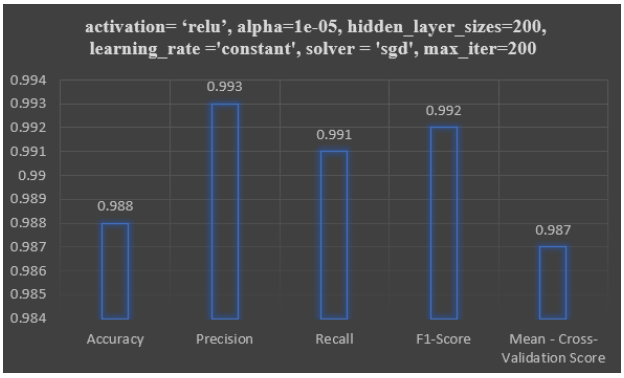


Fig. 27. Evaluation Metrics from Fig. 25-26

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

Fig. 28. Convergence Warning with 'max_iter': 200, 'solver': 'sgd'

VI. Conclusion

MLP Classifier, Decision Tree Classifier, and SVC Classifier are all popular machine learning models used for classification tasks. Each of these models has its own strengths and weaknesses. To make these models more effective, it's important to carefully select and preprocess the data, tune the hyperparameters, and perform cross-validation to evaluate the performance of the model.

Based on the evaluation using the best estimator parameters generated by the GridSearchCV function [7], it was found that the SVM Classifier had the highest accuracy both on the validation and testing set, while the Decision Tree Classifier had the lowest accuracy across the board. Comparing the performance of the MLP model to that of the SVM model is straightforward since the difference between their accuracy scores is negligible. This makes it easy to determine which model is performing better between the two and can help in selecting the optimal model for a given task. Overall, the MLP Classifier and SVM models produced the best performance compared to the Decision Tree model, but in other scenarios, a different model might be more suitable based on the specific requirements and characteristics of the dataset. Therefore, it is important to experiment with different models and evaluate their performance to determine the best model for a given dataset and classification problem. Additionally, one can try using ensemble methods such as bagging, boosting, or stacking to combine multiple models for improved performance.

The GridSearchCV function in the scikit-learn library is highly useful for both hyperparameter tuning and model selection. It conducts a thorough search across a designated hyperparameter grid to discover the best combination of hyperparameters that produce the most optimal model performance. It requires three inputs: the model to be tuned, a dictionary containing hyperparameters and their corresponding possible values, and a scoring metric for evaluating model performance. The function then proceeds to assess model performance for each combination of hyperparameters in the grid through cross-validation procedures.

By tuning the hyperparameters with GridSearchCV function, the model can be optimized for the specific task at hand, leading to improved performance and better generalization to new data.

Acknowledgment

We express our sincere gratitude to our supervisors Prof. Dr. Andreas Pech and Prof. Dr. Peter Nauth, for their unwavering guidance and support throughout the time of this project. Their insight and lectures have been invaluable in building our knowledge from the basic to the advanced understanding and practice in Autonomous Intelligence Systems, Machine learning and Artificial Intelligence.

We are also grateful to Mr. Julian Umansky for his technical expertise and assistance in the data analysis and willingness to impart his vast knowledge related to the subjects of Autonomous Intelligence Systems and granting us continuous access to the Autonomous Lab. Without their help, this project could not have been completed.

References

[1]Y. Zhang, Machine Learning (Intech, Vukovar, Croatia, 2010).

[2]A. H. Pech, P. M. Nauth, and R. Michalik, A New Approach for Pedestrian Detection in Vehicles by Ultrasonic Signal Analysis, IEEE EUROCON 2019 -18th International Conference on Smart Technologies (2019).

[3]X. Jiang, J. Sun, H. Ding, and C. Li, A Silhouette Based Novel Algorithm for Object Detection and Tracking Using Information Fusion of Video Frames, Cluster Computing 22, 391 (2018).

[4]Reichel Versand, SRF02 - Low Cost, High Performance Ultrasonic Ranger - DEV-SRF02 - Sensors, https://en.reichel-versand.de/DEV-SRF02.shtml.

[5]J. Jiang, W. Dang, F. Duan, X. Wang, X. Fu, C. Li, Z. Sun, H. Liu, and L. Bu, An Accurate Ultrasonic Wind Speed and Direction Measuring Method by Combining Time-Difference and Phase-Difference Measurement Using Coded Pulses Combination, Applied Acoustics 159, 107093 (2020).

[6]A. C. Cardenas Olaya, C. E. Calosso, J.-M. Friedt, S. Micalizio, and E. Rubiola, Phase Noise and Frequency Stability of the Red-Pitaya Internal PLL, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control 66, 412 (2019).

[7]F. Pedregosa et al., Scikit-Learn: Machine Learning in Python, Journal of Machine Learning Research 12, 2825 (2011).

[8]J. Alcaraz, M. Labbé, and M. Landete, Support Vector Machine with Feature Selection: A Multiobjective Approach, Expert Systems with Applications 204, 117485 (2022).

[9]Ehsan Ali Kareem, An Optimized Decision Tree Classification Algorithm for a Data Set (2017).

[10]L. M. Silva, J. Marques de Sá, and L. A. Alexandre, Data Classification with Multilayer Perceptrons Using a Generalized Error Function, Neural Networks 21, 1302 (2008).

[11]Z. Zhang, Improved Adam Optimizer for Deep Neural Networks, 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS) 1 (2018).

[12]D. Kingma and J. Lei Ba, Adam: A Method for Stochastic Optimization, 2017.

[13]O. Arinze, E. John, D. Okoyo and A. Bolutife, Dataset Repository, https://github.com/Rinzx/ML_FRAUAS_TG_S2223.

[13]A. Jung, Machine Learning : The Basics (Springer, Singapore, 2022).

[15]V. Ramya, T. Akilan, and N. Vignesh, Designing an Embedded System for Autonomous Building Map Exploration Robot, International Journal of Computer Applications 73, 11 (2013).

[16]Red Pitaya, Documentation - Red Pitaya, https://redpitaya.com/documentation/.

[17]A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, Deep Learning for Object Detection and Scene Perception in Self-Driving Cars: Survey, Challenges, and Open Issues, Array 10, 100057 (2021).