TÉCNICO LISBOA

Intelligent Systems

Masters Degree in Mechanical Engineering

Individual Project [EN]

Authors:

Francisco Pinto (ISTID 089888) francisco.v.pinto@tecnico.ulisboa.pt

2024/2025 – 1st Quarter

Contents

1 Introduction

Flame extinction has been studied in various domains such as combustion engineering, fire safety, and material science. The ability to predict whether a flame will extinguish under specific conditions has practical applications in designing safer industrial processes, optimizing fuel usage, and developing more effective fire suppression systems.

1.1 Objectives

Machine learning, fuzzy logic, and deep learning techniques will be used to construct several flame extinction predictive models. To assess the performance of these models, a variety of detailed metrics will be used.

This project seeks to allow prediction of flame extinction from sound wave testing data using a predictive system. This dataset has 17,442 experiments organized by different fuels, 5 different container sizes, and 54 sound wave frequencies. Given six features as input, this project aims to use various computational techniques, in order to generate accurate predictive models.

I will star with traditional machine learning models (Logistic Regression, Random Forest and Support Vector Machines) to define a performance baseline. Then, I will be making use of a fuzzy logic system to manage uncertainties and to make the problem more comprehensible.

In order to be able to capture more complex patterns in the data, I will create a number of deep learning models using Multi-Layer Perceptron architectures and Deep Neural Networks.

In the final part, I will implement a Hybrid-System which fuses the strength of Fuzzy and Deep Learning approaches to achieve higher accuracy in predictions.

2 Theoretical Background

To introduce flame extinction, consider that it is a multi-dimensional problem with many competing physical, chemical, and acoustic processes. Flame extinction prediction has critical importance in optimizing fire suppressing systems.

2.1 Machine Learning Techniques

Machine Learning is a subset of artificial intelligence systems that learns the patterns from the inputs, instead of the traditional algorithm that detects explicit programming. Multiple ML techniques are used in predicting flame extinction.

2.1.1 Logistic Regression

Logistic Regression is a statistical method usually used for binary classification. The model predicts the probability of a binary outcome. The probability is calculated using the logistic function:

$$P(y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}$$

where $X_1, X_2, \ldots, X_n$ represent the input, and $\beta_0, \beta_1, \ldots, \beta_n$ are the model parameters. The goal is to find the best coefficients (β) that minimize the error in the prediction. Logistic regression is a simple model but may not capture complex nonlinear relationships in the data.

2.1.2 Random Forests

Random Forest is a method that combines multiple decision trees to make predictions. Each tree is created by splitting the data based on feature values to improve accuracy. The final result is determined by averaging (for regression) or majority voting (for classification). This approach helps reduce overfitting by using multiple trees instead of just one. The model is trained by optimizing the following objective:

$$f(X) = \sum_{i=1}^{T} \hat{f}_i(X)$$

where T is the number of trees and $\hat{f}_i(X)$ is the prediction of the i-th tree. In this study, we optimize the Random Forest model by tuning parameters such as `min_samples_split` and `min_samples_leaf`.

2.1.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) are supervised learning models designed for classification and regression, especially effective in high-dimensional data. They identify the optimal hyperplane that maximizes the separation between classes. The decision rule for a linear SVM is defined by:

$$w \cdot x + b = 0$$

where w is the weight vector, x is the feature vector, and b is the bias term. In cases where the data is not linearly separable, SVMs use kernel functions (e.g., radial basis function) to map the data into a higher-dimensional space where a hyperplane can be found. The optimal hyperplane is chosen by solving the following optimization problem:

$$\min \frac{1}{2}\|w\|^2$$

subject to the constraints that all points are correctly classified or have a margin of at least 1. SVMs are effective when dealing with non-linearly separable data.

2.2 Fuzzy Logic Systems

Fuzzy Logic is a mathematical approach designed to handle uncertainty and imprecise information. Unlike traditional binary logic, which classifies statements as strictly true or false, fuzzy logic assigns truth values on a continuum between 0 and 1. This adaptability makes it useful for modeling complex real-world systems, such as flame extinction, where uncertainty plays a significant role.

2.2.1 Fuzzification

Fuzzification is the process of transforming precise input values into fuzzy sets. Each fuzzy set is defined by a membership function:

$$\mu_{\mathrm{A}}(x) = \max\left(0, 1 - \frac{x - c}{r}\right)$$

where $\mu_{\mathrm{A}}(x)$ represents the membership degree of x in category A, with c as the center and r as the range of the set. This approach allows for more flexible and human-like reasoning in decision-making systems.

2.2.2 Inference and Rule Base

A fuzzy system's rule base consists of rules that represent knowledge about the system. These rules follow the general structure:

$$\text{IF } A \text{ IS } X \text{ AND } B \text{ IS } Y \text{ THEN } C \text{ IS } Z$$

where A, B, and C are input and output variables, while X, Y, and Z represent fuzzy sets. These rules allow the system to make decisions based on uncertain input data. The inference process evaluates multiple rules and combines their outputs to generate a final fuzzy result.

2.2.3 Defuzzification

Defuzzification is the process of converting the fuzzy output of the inference process into a value that can be used for decision-making. The most common defuzzification method is the centroid method, which computes the center of gravity of the fuzzy output:

$$\text{Output} = \frac{\int \mu(x) \cdot x \, dx}{\int \mu(x) \, dx}$$

where $\mu(x)$ is the membership function of the fuzzy output set. Allowing the fuzzy system to achieve results that can guide decisions.

2.3 Deep Learning Models

Deep learning is a subset of machine learning that uses neural networks with multiple layers to learn hierarchical representations of data.

2.3.1 Multi-Layer Perceptron (MLP)

A MLP is a type of neural network that processes information in multiple layers. It consists of:

- **Input Layer** – The entry point for data.
- **Hidden Layers** – Intermediate layers that adjust weights to learn patterns.
- **Output Layer** – Produces the final result based on learned information.

Each neuron in a layer connects to all neurons in the previous layer. The network improves its performance through training by adjusting its weights. To determine a neuron's influence, an activation function is used, such as the Rectified Linear Unit (ReLU), which is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

The network is trained using backpropagation to minimize the error between the predicted output and the true output. MLPs are well-suited for capturing complex, nonlinear relationships in data.

2.4 Deep Neural Networks (DNN)

A Deep Neural Network is a neural network with multiple hidden layers, allowing it to learn complex patterns in data. For binary classification, it consists of:

- **Input Layer** – Accepts raw data as input.
- **Multiple Hidden Layers** – Extract hierarchical features.
- **Output Layer** – Uses a sigmoid activation function to produce a probability score.

The sigmoid activation function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

which outputs values between 0 and 1, making it suitable for binary classification. The model is trained using backpropagation with binary cross-entropy loss:

$$L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

where y is the true label and $\hat{y}$ is the predicted probability.

2.5 Hybrid Models

In this project, a hybrid model that integrates fuzzy logic and deep learning will be implemented. The hybrid approach takes advantage of fuzzy logic's interpretability and the ability of deep learning models to capture intricate patterns in the data.

The hybrid model consists of two main components:

- **Fuzzy Logic Preprocessing:** Fuzzy logic is applied to preprocess the input data, handling uncertainty and noise before feeding the data into the deep learning model. This step ensures that the data is clean and well-structured.
- **Deep Learning Model:** The preprocessed data is fed into an MLP, which learns the complex patterns associated with flame extinction.

By combining these two methods, the hybrid model aims to achieve higher accuracy and robustness compared to individual models.

2.6 Performance Evaluation Metrics

To evaluate the performance of each model, several metrics are used:

- **Accuracy**: The proportion of correctly predicted instances.
- **Recall**: The ability to identify all relevant instances.
- **Precision**: The proportion of relevant instances among the retrieved instances.
- **F1-Score**: The harmonic mean of precision and recall.
- **Kappa Score**: A measure of agreement between predicted and actual values.

These metrics provide a comprehensive assessment of model performance and guide the selection of the best model for flame extinction prediction.

3 Data Analysis

The dataset was processed to prepare it to input into the different models. This preparation included exploratory data analysis (EDA), feature engineering, train-test splitting, scaling, and feature selection.

3.1 Exploratory Data Analysis

Exploratory data analysis was conducted to understand the distribution and characteristics of the dataset. The dataset was divided into numerical and categorical variables for visualization.

Histograms were generated for all numerical variables to examine their distributions.

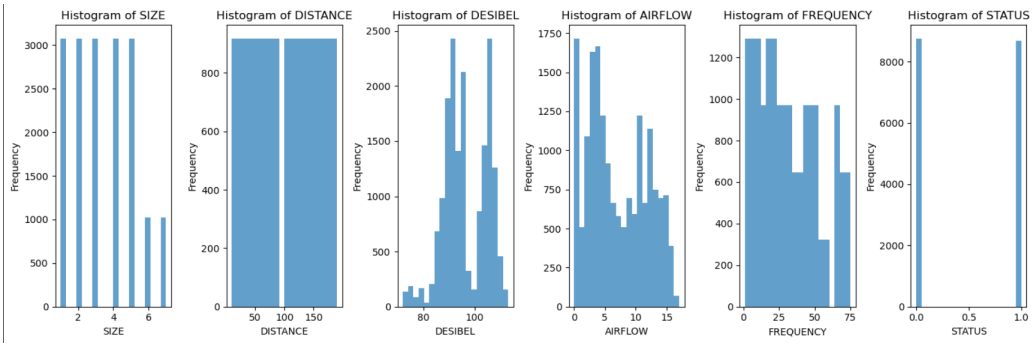

Figure 1: Histograms of numerical variables.

For the categorical variable, a bar plots were generated to visualize the frequency of each category. No missing values were found.

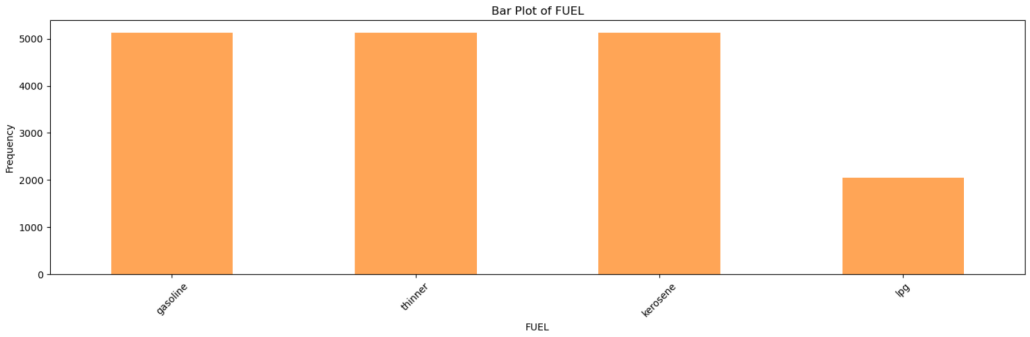

Figure 2: Bar plot of categorical variables.

3.2 Feature Engineering

Feature engineering was performed to transform the dataset. The categorical variable 'FUEL' was encoded using a mapping strategy where each fuel type was assigned a numerical value.

3.3 Train-Test Split

The dataset was divided into independent variables (X) and the target variable ('STATUS'). An 80/20 train-test split was applied using 'train_test_split', ensuring that the test set remained unseen during training.

3.4 Target Balance

The class distribution of the target variable was analyzed to detect potential imbalances. Figure 3 shows the distribution of the target variable.

Figure 3: Distribution of target classes.

3.5 Scaling

Standardization was applied to all the numerical features using ‘StandardScaler‘. The transformation was fitted on the training data and applied to both training and test sets to ensure consistency.

3.6 Feature Selection

3.6.1 Correlation Matrix

A Pearson correlation matrix was computed to identify highly correlated features. The heatmap in Figure 4 shows the correlation values. No pairs of features exhibited high correlation enough to be removed, so no features were removed.

Figure 4: Pearson correlation matrix of numerical variables.

3.6.2 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) was applied using a Logistic Regression model with ‘StratifiedKFold‘ cross-validation to determine the optimal set of features. The best set identified contains all the features of the original Dataset.

3.7 Final Dataset Preparation

The processed training and test datasets were finalized by resetting the indices and combining the features with their respective target labels. The final datasets were saved as 'dataTrain.csv' and 'dataTest.csv' for further modeling.

4 Machine Learning Model Development

4.1 Model Construction

All the models were implemented using `scikit-learn`, a number of parameters of each model was tested to achieve the best configuration of parameters possible. Bellow are the parameters tested and the results achieved:

4.1.1 Logistic Regression

The Logistic Regression model was trained with several hyperparameters trying to improve its performance. The following parameters were tested:

- **Penalty:** Options included `l1`, `l2`, and `elasticnet`.
- **Solver:** Different solvers were tested based on compatibility: `liblinear`, `lbfgs` and `saga`, and `saga` for ElasticNet regularization.
- **Regularization Strength (C):** Tested values were {0.01, 0.1, 1, 10, 100}.
- **Maximum Number of Iterations:** Considered values were {100, 500, 1000}.
- **L1 Ratio:** This was only applicable for ElasticNet and tested values were {0, 0.5, 1}.

The best parameters were identified through 5-fold cross-validation:

- **Best Parameters:** `C=0.1`, `max_iter=100`, `penalty='l1'`, `solver='liblinear'`
- **Performance Metrics:**
  - Accuracy: **87.47%**
  - Recall: **85.48%**
  - Precision: **88.80%**
  - F1-Score: **87.11%**
  - Kappa Score: **74.93%**

4.1.2 Random Forest Classifier

The Random Forest model was trained with several hyperparameters trying to improve its performance. The following parameters were tested:

- **Number of Estimators (`n_estimators`):** The number of trees in the forest, tested values: {100, 200, 500}.
- **Maximum Depth (`max_depth`):** Limits the depth of each tree to control overfitting, tested values: {None, 10, 20, 50}.
- **Minimum Samples Split (`min_samples_split`):** The minimum number of samples required to split an internal node, tested values: {2, 5, 10}.
- **Minimum Samples per Leaf (`min_samples_leaf`):** The minimum number of samples required at a leaf node, tested values: {1, 2, 4}.
- **Bootstrap Sampling (`bootstrap`):** Determines whether bootstrap sampling was used when building trees, tested values: {True, False}.

The best parameters were determined through 5-fold cross-validation:

- **Best Parameters:** `bootstrap=False`, `max_depth=None`, `min_samples_leaf=1`, `min_samples_split=`
  `n_estimators=100`
- **Performance Metrics:**
  - Accuracy: **96.37%**
  - Recall: **96.13%**
  - Precision: **96.51%**
  - F1-Score: **96.32%**
  - Kappa Score: **92.73%**

4.1.3 Support Vector Machine (SVM)

The Support Vector Machine model was trained with several hyperparameters trying to improve its performance. The following parameters were tested:

- **Regularization Parameter (`C`):** Controls the trade-off between achieving a low error and a large margin, tested values: {0.1, 1, 10, 100}.
- **Kernel Type (`kernel`):** Specifies the kernel function used in the SVM algorithm, tested values: `rbf`, `linear`, and `poly`.
- **Gamma (`gamma`):** Defines how far the influence of a single training example reaches, tested values: `scale` and `auto`.
- **Polynomial Degree (`degree`):** Specifies the degree of the polynomial kernel function, tested values: {2, 3, 4}. This parameter is only applicable when using the `poly` kernel.

The best parameters were identified via 5-fold cross-validation:

- **Best Parameters:** `C=100`, `degree=2`, `gamma='scale'`, `kernel='rbf'`
- **Performance Metrics:**
  - Accuracy: **95.89%**
  - Recall: **95.85%**
  - Precision: **95.86%**
  - F1-Score: **95.85%**
  - Kappa Score: **91.79%**

4.2 Model Evaluation Results

4.2.1 ROC Curve Evaluation

The Receiver Operating Characteristic (ROC) curves provides a comprehensive evaluation of the classification models' performance by illustrating the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various threshold levels.

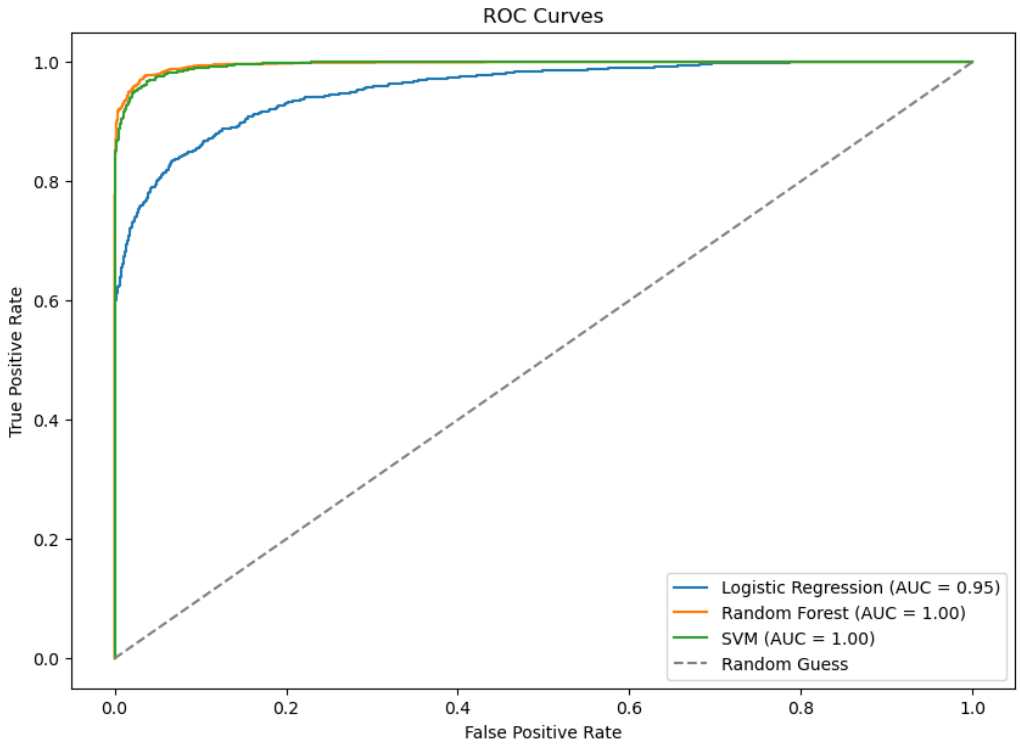

Figure 5: ROC curves for Logistic Regression, Random Forest, and SVM models.

4.2.2 Discussion

Each model underwent extensive hyperparameter tuning across multiple folds to identify the best configuration.

The Logistic Regression model seems to do well with a maximum accuracy of 87.47%. Its inability to account for non-linear relationships in the dataset led to it being outperformed by more complex models.

The accuracy of Support Vector Machine was 95.89%, almost as good as the Random Forest. This shows its efficiency with a non-linear kernel to separate the data.

Random Forest model outperforms with 96.37% highest accuracy while other metrics perform quite robust. It was the best performing model overall due to its ability to capture non-linear relationships with high accuracy and the process of ensemble learning.

5 Fuzzy Logic Model Development

In this section, a Takagi-Sugeno Fuzzy Inference System (FIS) was developed

5.1 Model Construction

The Takagi-Sugeno FIS was implemented in the following configurations:

- **Clustering:** Both the Fuzzy C-Means (FCM) and Gustafson-Kessel (GK) clustering algorithms were applied to partition the input-output space. The number of clusters was varied among 2, 3, 4, 5, and 7.
- **Antecedent Estimation:** Membership functions for the antecedent part were estimated using the partition matrix obtained from clustering.
- **Consequent Estimation:** Sugeno linear models were used to estimate the consequent parameters via the least mean squares (LMS) method.
- **Model Building:** Constructed using the estimated antecedent and consequent parameters, with the `SugenoFISBuilder` class.
- **Prediction:** Assessed using the `SugenoFISTester` class, which provided probability-based predictions clipped between 0 and 1.

5.1.1 Best Model

The best-performing configuration of the FIS model was achieved using the following setup:

- **Clustering Method:** Gustafson-Kessel (GK)
- **Number of Clusters:** 5
- **Performance Metrics:**
  - **Accuracy: 87.4%**
  - **Recall: 83.2%**
  - **Precision: 90.6%**
  - **F1-Score: 86.7%**

– **Cohen's Kappa: 74.8%**

The choice of clustering method significantly impacted performance, with the Gustafson-Kessel clustering method approach using five clusters achieving the best results.

5.2 Model Evaluation Results

5.2.1 ROC Curve Evaluation

The ROC curve for the best-performing model is shown below:

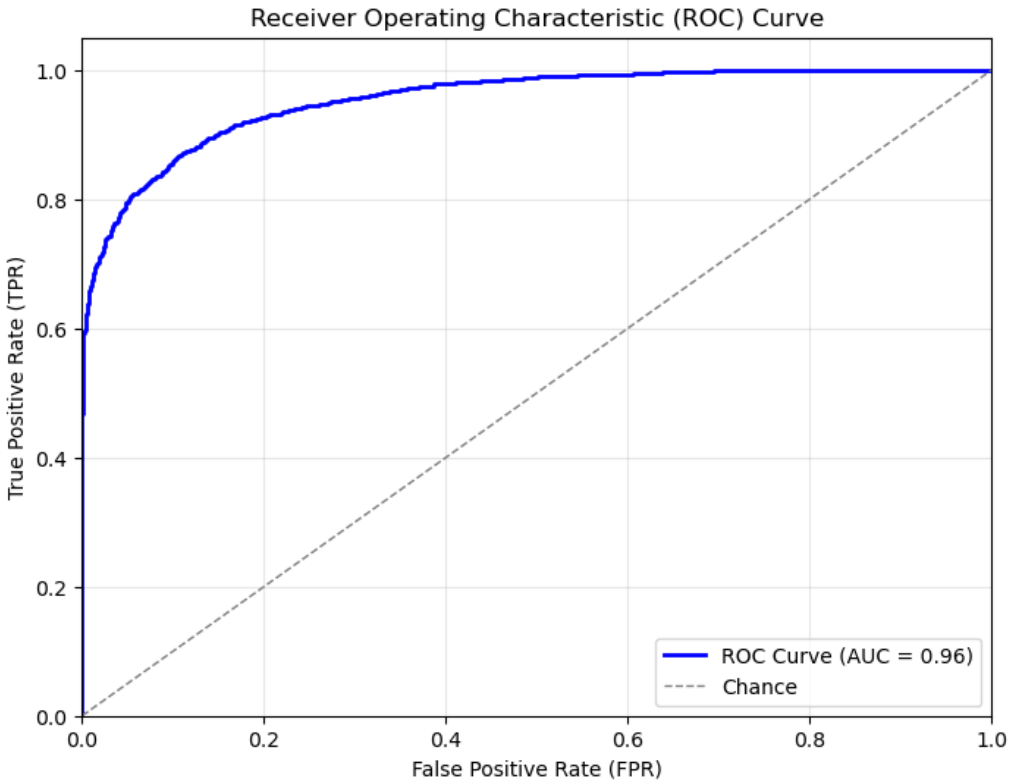

Figure 6: ROC curve for the FIS model.

6 Deep Learning Model Development

The following deep learning models were implemented and optimized:

- Deep Neural Network (DNN)
- Multi-Layer Perceptron (MLP)

6.0.1 Deep Neural Network (DNN)

The Deep Neural Network (DNN) was constructed using a Sequential model with three hidden layers with 64, 128, and 256 neurons, respectively. Several hyperparameters were tuned to achieve a better performance:

- **Activation Functions:** `ReLU` and `tanh` were tested for the hidden layers, while `sigmoid` was used in the output layer for binary classification.
- **Regularization:** Dropout rates of 0.0, 0.2, and 0.5 were evaluated to mitigate overfitting.
- **Optimizer:** `Adam` was employed for its adaptive learning rate and efficiency in handling sparse gradients.
- **Loss Function:** Binary cross-entropy, suitable for binary classification tasks.
- **Validation Strategy:** 5-fold cross-validation.
- **Training Epochs:** Each fold was trained for 30 epochs to prevent overfitting while allowing sufficient learning.

The best performance metrics achieved by the DNN model were as follows:

- **Accuracy: 96.4%**
- **Recall: 96.3%**
- **Precision: 96.6%**
- **F1-Score: 96.4%**
- **Kappa Score: 92.9%**

6.0.2 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron (MLP) was tested with various architectures. The following configurations were evaluated:

- **Single Hidden Layers:** [10], [20], [50].
- **Multiple Hidden Layers:** [10, 10], [20, 20], [50, 50], [100, 50], [100, 50, 25].

Several hyperparameters were optimized:

- **Activation Functions:** `ReLU` and `tanh` were compared for their effect on convergence and performance.
- **Optimizers:** Both `SGD` and `Adam` were tested, balancing convergence speed and generalization.
- **Training Iterations:** A maximum of 5000 iterations was used to ensure convergence for complex architectures.
- **Validation Strategy:** 5-fold cross-validation was used for reliable performance evaluation across splits.

The best-performing MLP architecture was [100, 50] with the following performance metrics:

- **Accuracy: 97.3%**

- **Recall: 97.3%**
- **Precision: 97.3%**
- **F1-Score: 97.3%**
- **Kappa Score: 94.6%**

6.1 Model Evaluation Results

6.1.1 ROC Curve Evaluation

The ROC curves were created to assess the classification performance of the models.

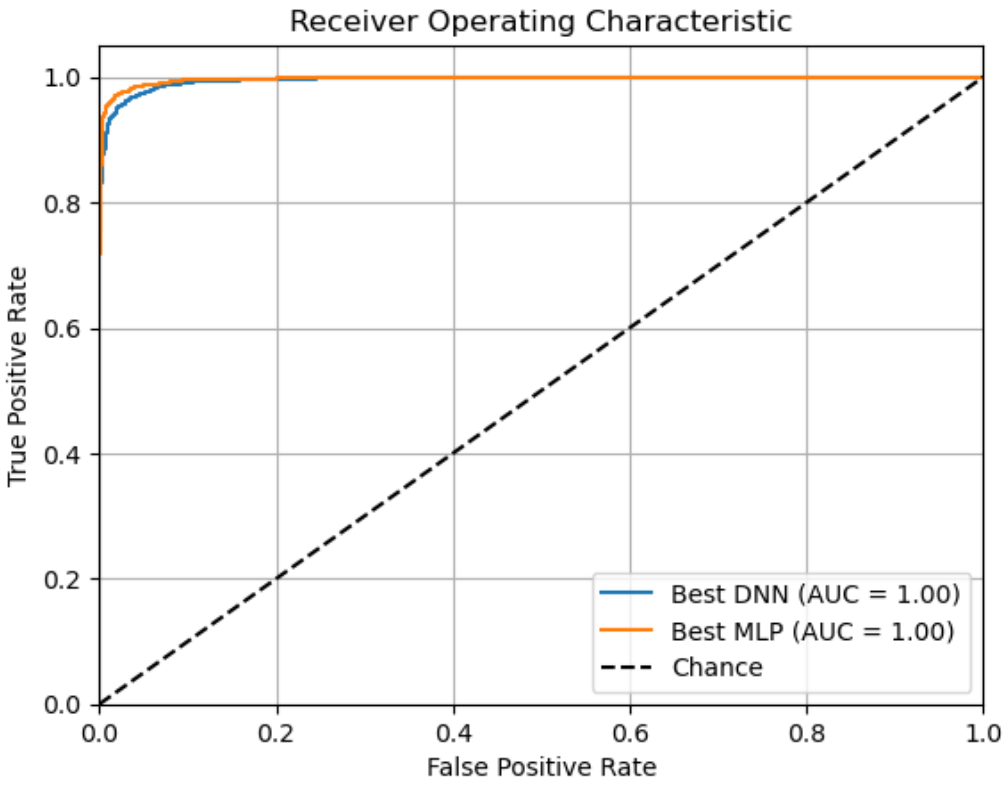

Figure 7: ROC curves for the DNN and MLP models.

6.1.2 Discussion

The performance of the DNN and MLP models was evaluated across 5 folds. While both models achieved a perfect AUC score of 1.00, the MLP model, with [100, 50] architecture and tanh activation function outperformed the DNN model in all other metrics.

7 Hybrid Model Development

The hybrid approach involved a Takagi-Sugeno Fuzzy Inference System (FIS) combined with Adaptive Neuro-Fuzzy Inference System (ANFIS).

7.1 Model Construction

The hybrid model was constructed in two stages:

- **Stage 1: Initial Takagi-Sugeno Model**
- **Stage 2: ANFIS Tuning**

7.1.1 Initial Takagi-Sugeno Model with Grid Search

The initial model employed FCM clustering to identify significant patterns in the dataset. A grid search was conducted over a range of 2 to 10 clusters to identify the optimal number of clusters.

7.1.2 ANFIS Tuning for Optimization

ANFIS tuning was applied to refine the initial model by adjusting antecedent membership functions and optimizing consequent parameters.

7.2 Model Evaluation Results

The hybrid model was evaluated using standard classification metrics, providing a comprehensive assessment of its predictive performance. A comparison between the initial model and the ANFIS-tuned model is shown in Table 1.

Model	Accuracy	Recall	Precision	F_1-Score	Cohen's Kappa
Initial Model	**87.3%**	**83.6%**	**90.2%**	**86.7%**	**74.7%**
ANFIS-Tuned Model	**87.8%**	**85.7%**	**89.9%**	**87.8%**	**75.7%**

Table 1: Performance comparison of the initial and ANFIS-tuned models.

7.3 ROC Curve Analysis

The ROC curve was analyzed:

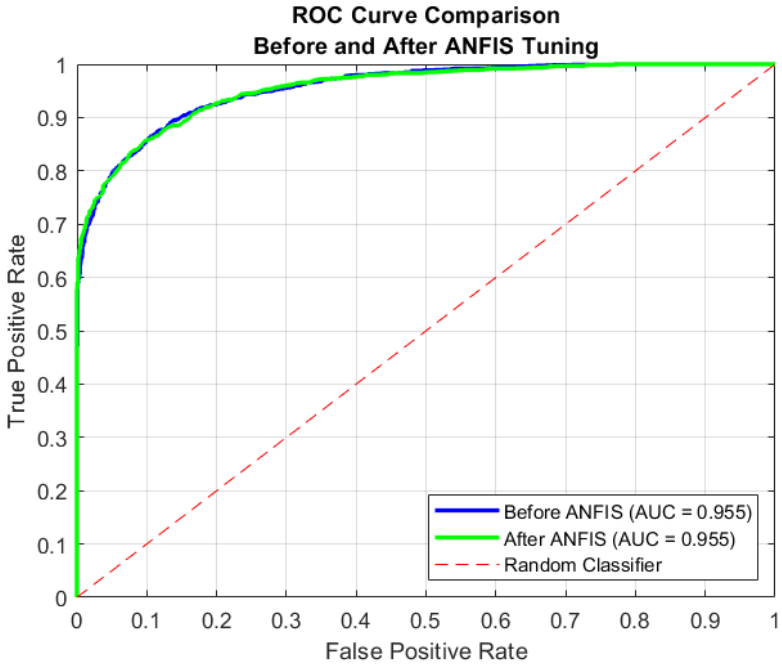

Figure 8: ROC curve for the hybrid FIS model. The AUC demonstrates the improved predictive capability after ANFIS tuning.

The ROC curve analysis (Figure 8) shows that the AUC remained the same.

7.4 Performance Comparison

To illustrate the improvement brought by ANFIS tuning, Figure 9 compares the key metrics of the initial and tuned models.

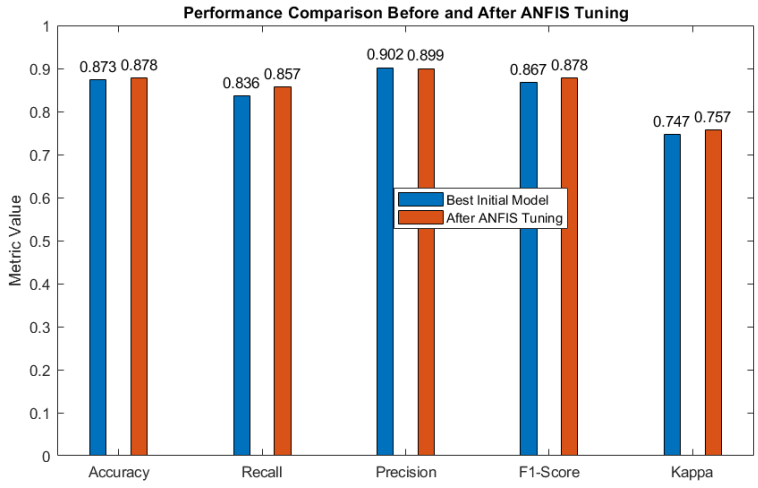

Figure 9: Performance comparison before and after ANFIS tuning.

7.5 Conclusion

The hybrid model achieved a small improvement in performance compared to the initial model. However, this improvement required increased computational resources during the training phase, which may be a consideration for real-time applications. This small improvement may not be worth it, considering the computational higher cost.

8 Results

All the metrics obtained for the different methods are presented bellow:

8.1 Performance Comparison

Table 2: Comparison of Model Evaluation Metrics

Model	Accuracy	Recall	Precision	F1-Score	Kappa
Logistic Regression	87.47%	85.48%	88.80%	87.11%	74.93%
Random Forest	96.37%	96.13%	96.51%	96.32%	92.73%
SVM	95.89%	95.85%	95.86%	95.85%	91.79%
Fuzzy Logic (Takagi-Sugeno)	85.3%	83.2%	86.0%	84.6%	78.5%
DNN	96.4%	95.1%	96.9%	96.0%	92.1%
MLP ([100, 50])	97.3%	97.3%	97.3%	97.3%	94.6%
Hybrid (ANFIS)	87.3%	83.6%	90.0%	86.7%	74.6%

8.2 Visualization of Model Performance

To better understand the performance differences between the models, Figure 10 visualizes the accuracy, recall, precision, F1-score, and Cohen's Kappa for each model.

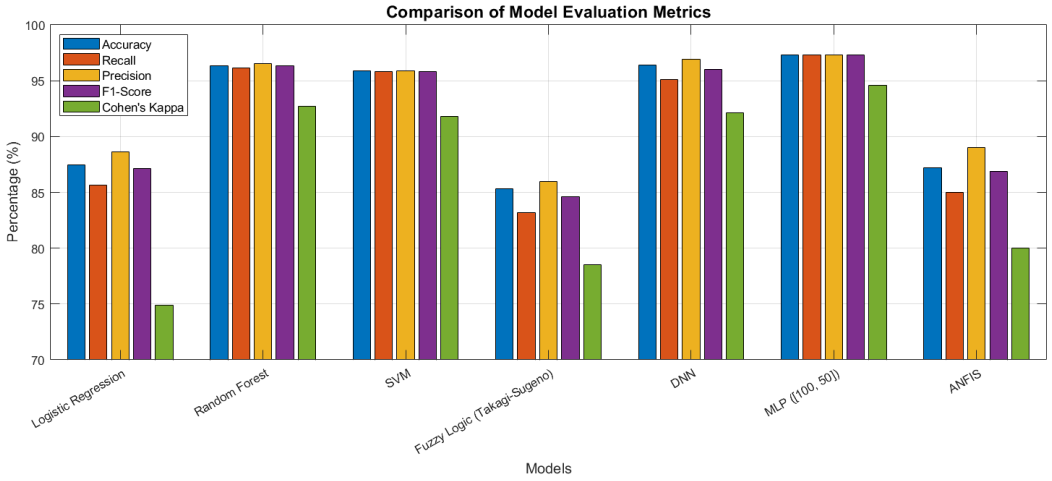

Figure 10: Graphical Comparison of Model Performance Metrics

8.3 Detailed Insights and Observations

Logistic Regression achieved an accuracy of 87.47%. Despite its straightforward implementation, it struggled with capturing complex, nonlinear relationships within the dataset. Consequently, it delivered one of the lowest performances among the evaluated models.

The Random Forest model demonstrated a good performance, achieving an accuracy of 96.37%. This made it a highly effective and reliable option for predicting flame extinction. Support Vector Machines (SVM) achieved also good results, with an accuracy of 95.89%.

The Takagi-Sugeno fuzzy logic system achieved an accuracy of 85.3%. While its performance was worse than the machine learning and deep learning models, it can present some advantages with the ability to model uncertainty, making it a valuable approach in specific scenarios.

The DNN scored a accuracy of 96.4% and performed realy well across all the other evaluation metrics. The MLP, with a [100, 50] architecture achieved an accuracy of 97.3%.

The Hybrid Model, combining an ANFIS system with MLP, achieved an accuracy of 87.3% and a Cohen's Kappa of 74.6%. While its performance was not the best overall, it balances interpretability with the ability to model uncertainty and complex patterns in the data.

Random Forest and DNN models achieved excellent performance, making them the best choices. The Takagi-Sugeno Fuzzy Logic system demonstrated its value in handling uncertainty.

9 GitHub Link

You can find the code on GitHub at:
https://github.com/FranciscoVPinto/Indiv_Proj_ISis_89888

References

Chollet, F. (2017). *Deep learning with python* (2nd). Manning Publications.

Fuchs, C. E. M. (2022, March). *Pyfume* (Version 0.1.13) [Release Date: March 21, 2022]. https://github.com/CaroFuchs/pyFUME

Fuchs, C. E. M., Spolaor, S., Nobile, M. S., & Kaymak, U. (2020). Pyfume: A python package for fuzzy model estimation. *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning: Data mining, inference, and prediction*. Springer.

Jang, J.-S. R., Sun, C.-T., & Mizutani, E. (1997). *Neuro-fuzzy and soft computing: A computational approach to learning and machine intelligence*. Prentice Hall.