Machine Learning Based Ensemble Model for Smart Fire Detection System using Classification Algorithms

Abstract:

Fire detection is crucial for reducing loss of life, property destruction, and economic disruptions. Despite advancements in AI and machine learning, fire detection technology still faces challenges in reducing false alarms, enhancing sensitivity, and safeguarding complex installations. This review analyzes current cutting-edge practices in fire detection and monitoring strategies, focusing on continuous monitoring of variables like temperature, flame, gas content, and smoke. The study aims to create a hybrid ensemble classifier that combines Decision tree classifier and Random Forest classifier, enhancing prediction accuracy and resilience. The performance of the proposed machine learning algorithm is evaluated using various metrics such as confusion matrix, model accuracy, precision, and error calculation. Results show that the suggested ensemble technique performs more accurately in terms of classification than previously reported literature. The created model aids in real-time data analytics with high accuracy and low root mean square error (RMSE).

Keywords: Fire detection, AI, machine learning, false alarms, ensemble classifier, Decision tree classifier, Random Forest classifier, model accuracy, precision, RMSE

1. Introduction:

Fire, a formidable force, has long been responsible for a multitude of tragic outcomes, causing loss of life, devastating property, and severely disrupting economies across the globe. The annual occurrence of numerous fire incidents leaves immeasurable devastation in its wake. To minimize these harrowing consequences, it is of utmost importance to implement cutting-edge and efficient early warning technologies for fire detection. Despite some strides made in research publications addressing this issue, fire detection technology continues to grapple with challenges, such as reducing false alarms, improving sensitivity and dynamic response, and safeguarding complex and costly installations.

In this comprehensive review, we aim to delve into the current state of the art practices in fire detection and monitoring strategies. Specifically, our focus lies on innovative methods that involve continuous monitoring of crucial variables like temperature, flame, gas content, and smoke. By examining the advantages, disadvantages, measurement standards, and parameter ranges of these techniques, we seek to shed light on the various aspects of fire detection technology.

Moreover, our investigation extends towards exploring ongoing research directions, identifying persistent challenges in fire detection technology, and envisioning future prospects for developing advanced fire sensors. Through this overview, we aspire to

inspire further research in fire sensor technology, thus paving the way for revolutionary advancements in fire detection techniques.
The review seeks to address challenges faced by existing fire detection methods, such as false alarms, sensitivity, and protection of valuable installations. There has been previous research that explores the use of technology to reduce fire outbreaks, such as computer vision techniques and deep learning advancements. The emphasis is on the potential of artificial intelligence (AI) and machine learning (ML) as effective solutions for managing fire disasters. These technologies have shown promise in detecting and analyzing fire signals, addressing false alarms, and identifying fire-related issues.
Furthermore, the research's specific aim is to develop a hybrid ensemble classifier by combining the Decision tree and Random Forest classifiers. This classifier will be employed to predict fire scenarios using a preprocessed dataset. The implementation of an average voting ensemble technique aims to enhance prediction accuracy and the learning algorithm's resilience. The performance evaluation will employ various metrics, including confusion matrix, model accuracy, F1 Score, and error calculation like RMSE. Lastly, the proposed methodology will be validated through experiments using a smart IoT sensor node prototype, emphasizing practical application and real-world testing.

2. Literature Review:

Study [1] examined various classification algorithms, including logistic regression, SVM, decision trees, and random forest, for binary classification tasks. Logistic regression outperformed decision trees and SVM according to the sliding control method. However, the Random Forest algorithm was deemed more suitable due to its resistance to changes in the training sample and its ability to handle complex boundaries.
Paper [2] Focused on developing an IoT-based fire alarm system using multiple sensors and classifiers. Classification algorithms like K-Nearest Neighbor (KNN), Decision Tree, Naïve Bayes, and Support Vector Machine (SVM) were compared, and Naïve Bayes achieved the highest accuracy of 100%. The study suggests incorporating online learning for real-time classifier selection.
In study [3] a hybrid ensemble model was introduced for real-time fire detection using multi-sensor data. The model combined logistic regression, SVM, decision tree, and Naive Bayes classifiers and outperformed individual classifiers and existing literature in accuracy, AUC, and precision. A smart multi-sensor fire detection device was developed for wireless data transmission to a cloud platform.
Paper [4] explored the use of Machine Learning models for predicting forest fires, evaluating eight algorithms. The Boosted decision tree model performed the best with an AUC value of 0.78. The paper proposed an IoT-based smart Fire prediction system that incorporates meteorological data and images to predict fires at an early stage.
Forest fire prediction in Riau, Indonesia [5] utilized Decision Tree and Bayesian Network algorithms for fire prediction. Decision Tree provided attribute relationships,

while Bayesian Network offered accuracy. The Bayesian Network showed potential but required further improvement and additional experiments with Deep Learning and forecasting algorithms, as well as data from other IoT sensors and environmental expert interpretation.

Research [6] successfully used Naïve Bayes to classify forest fire hotspots in Pelalawan district from 2015 to 2019. Accuracy for 2015 dataset: 99.45%, for 2019 dataset: 99.52%. Enables timely preventive measures. Future work includes a web-based application for displaying results.

Systematic review [7] focused on forest fire detection using IoT and machine/deep learning. Temperature, humidity, CO, and light are primary features. Common protocols include WIFI, ZigBee, GSM and mostly synthetic datasets are used. Machine learning predominates few deep learning models as classification, regression, object detection is popular. Combining IoT and machine/deep learning shows promise for wildfire detection and forecasting.

Proposed forest fire detection system [8] with wireless sensor networks and machine learning is effective, accurate, and fast. Adapts to different conditions, uses rechargeable batteries and solar power, alerting authorities faster than existing systems.

Study [9] evaluated four fire predictive models using machine learning in Pu Mat National Park, Vietnam and achieved high accuracy (AUC > 0.90). The Bayesian model was slightly better but less interpretable than Multivariate Logistic Regression. Updates and attention to human activities were needed for fire susceptibility management. Future studies can include additional variables for better fire behavior understanding.

Review [10] analyzed 135 papers on machine learning in wildfire management. Categorized applications into pre-fire prevention, active wildfire management, and post-wildfire activities. Main methods included random forest, support vector machines, neural networks and there were various challenges like improving speed, interpretability, and computational time.

The article [11] proposes a Multimodal framework for early detection of forest fires. It utilizes temperature, humidity, and image data from deployed sensors to predict fire-prone areas. The integrated model shows high accuracy, aiding prompt actions to mitigate forest fires.

Researchers [12] used three meta-heuristic classifiers and one data extraction classifier on a dataset with 25 attributes affecting student performance. SVM achieved the highest accuracy (87.5%), and a hybrid ensemble model further improved accuracy to 98.6%, enabling precision education.

The proposed method [13], CFS_FPA, achieved 87% accuracy and reduced FNR to 0.123 by combining correlation feature selection and forest penalized attribute. Experiments with CICIDS2017 dataset showed 99.73% accuracy, outperforming other algorithms, ensuring reliable and robust intrusion detection.

The paper [14] proposes hybrid AdaBoosting and Bagging approaches for IDS. Four classifiers (SVM, RF, KNN, NB) are modified and used with AdaBoosting to detect known and unknown attacks. Improved efficiency with 99.99% training accuracy and 85.49% testing accuracy. It outperformed other techniques in the field. Further enhancements needed for rare network traffic threats.

3. Research Methodology:

3.1 Data Gathering

The initial phase includes the acquisition of Dataset. The fire data obtained from the NIST Website has been used for conducting performance evaluation of the proposed model.
The Dataset outlines a series of experimental tests aimed at understanding fire scenarios. The dataset determines the presence of fire in relation to various factors. The data presented in the report includes information about varying concentrations of CO, CO2, and O2, smoke obscuration, and temperature at various points within the structure.
The dataset contains 5450 data samples with readings signifying Temperature, Smoke, CO conc., CO2 conc., O2 conc. with respective fire labels.

3.2 Data Preparation

Data preprocessing is an essential phase in machine learning that involves preparing unprocessed data for analysis through cleansing and converting it. Sometimes, the raw data comes from multiple sources and may contain incomplete, sparse, or inconsistent information. Therefore, it becomes necessary to filter out similar sparse data during this stage and ensure the data is standardized.
Various data preparation techniques are commonly employed such as:

- Data integration: Information is combined from different sources to create a unified dataset, which can be a laborious task due to handling diverse forms, structures, and meanings. Techniques like record relations and data emulsion are utilized in data integration.
- Data transformation: It is another crucial aspect, wherein data is converted into a suitable format for analysis. Common techniques in data transformation include normalization, standardization, and discretization. Standardization adjusts the data to have a mean of zero and a variance of one, while normalization scales the data to a common range. Continuous data is discretized into discrete categories through this method.
- Data reduction: It involves condensing the dataset while retaining essential information. This process helps to manage large datasets and facilitate more efficient analysis without losing vital data.

3.3 Data Partition

In machine learning, splitting the dataset refers to dividing the available data into distinct subsets for different purposes, typically for training and testing. The main motivation behind dataset splitting is to effectively train and evaluate

machine learning models to ensure their generalizability on unseen data. Data is generally split into:

1. Training set: This is the largest portion of the dataset and is used to train the machine learning model. During training, the model learns from the patterns and relationships present in the training data to make predictions on new, unseen data.
2. Test set: The test set is used to assess the final performance of the trained model. It represents a completely unseen dataset, and the model's predictions on this set can provide an unbiased estimate of how well the model will perform in the real world.

The typical approach for splitting the dataset involves randomly shuffling the data and then partitioning it into the desired proportions (e.g., 70% training, 30% testing). The training set is used to fit the model's parameters and the test set is reserved for final evaluation.

It's crucial to ensure that the data split is representative of the overall dataset to avoid any biases in the model's training and evaluation process. We may build precise predictive models, assess how well they generalise, and prevent overfitting by following these procedures.

3.4 Model Building

The requirement is to create a model using the Python programming language, utilizing essential libraries like NumPy, Pandas, matplotlib etc. and scikit-learn library is used for machine learning. Different machine learning methods, such as Logistic Regression, Naive Bayes, Decision Tree, Support Vector Machine, Random Forests, and K-Nearest Neighbours (KNN), have been used for classifying the data. The classification algorithms with the highest accuracy were selected after testing several different algorithms.

Individual Classifiers -

Logistic Regression: Logistic regression is a statistical method used for binary classification, where the goal is to predict one of two possible outcomes based on input features. It is a supervised learning algorithm commonly used in machine learning.

In logistic regression, the output is transformed using the logistic function (also known as the sigmoid function) to ensure that the predicted values are in the range [0, 1]. This transformation allows us to interpret the output as a probability of belonging to a particular class.

The algorithm works by finding the best-fitting line (or hyperplane for multi-dimensional data) that separates the two classes in the feature space. It does this by minimizing a cost function, such as the binary cross-entropy, which measures the difference between the predicted probabilities and the actual class labels in the training data.

Logistic regression is a simple and efficient algorithm, and with the help of Python libraries like scikit-learn, implementing logistic regression becomes straightforward.

Implementation:

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier1 = LogisticRegression()
classifier1.fit(x_train,y_train)
```

Naive Bayes: Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem, which is used for classification tasks. It assumes feature independence, simplifying calculations and making it computationally efficient. During training, it learns class and feature probabilities from labeled data. For prediction, it calculates the posterior probability of each class based on input features using Bayes' theorem.
The formula for Bayes' theorem is as follows:

P(class|features) = (P(features|class) * P(class)) / P(features)
Where:
- P(class|features) is the posterior probability of the class given the input features.
- P(features|class) is the likelihood of the input features given the class.
- P(class) is the prior probability of the class.
- P(features) is the probability of the input features.

Naive Bayes often performs surprisingly well in practice and can be an efficient and quick option for classification tasks, especially when dealing with high-dimensional data or large datasets.
Python's scikit-learn library provides a convenient implementation of Naive Bayes, making it easy to use and apply to various classification tasks.

Implementation:

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
classifier2 = GaussianNB()
classifier2.fit(x_train,y_train)
```

Decision Tree: Decision Tree is a popular machine learning algorithm used for both classification and regression tasks. It is a non-parametric algorithm that recursively splits the data based on different features to create a tree-like structure of decisions. Each internal node of the tree represents a test on a specific feature, and each leaf node represents a class label (in the case of classification) or a numeric value (in the case of regression). During training, the algorithm selects the best feature to split the data. For prediction, it traverses the tree to reach a leaf node and uses its output as the prediction.

Decision trees have several advantages, including simplicity, interpretability, and the ability to handle both numerical and categorical data. They can also automatically handle feature interactions and perform feature selection implicitly by selecting important features near the top of the tree.
However, decision trees can suffer from overfitting, especially when the tree becomes too deep and complex. To overcome this, techniques like pruning (removing branches that do not improve performance) and using ensemble methods like Random Forest can be employed.
Python's scikit-learn library provides an efficient implementation of decision trees, making it easy to use and apply to various classification and regression tasks.

Implementation:

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
classifier3 = DecisionTreeClassifier()
classifier3.fit(x_train,y_train)
```

Support Vector Machine (SVM): Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for classification and regression tasks. It is particularly effective for binary classification problems but can be extended to handle multi-class classification as well.
The main idea behind SVM is to find the optimal hyperplane that best separates data points of different classes in the feature space. In a binary classification scenario, the hyperplane is a line that maximizes the margin, which is the distance between the hyperplane and the closest data points (known as support vectors) from each class. The goal is to find the hyperplane that maximizes the margin while minimizing the classification error.
The training process in SVM involves solving an optimization problem to find the optimal hyperplane and the support vectors. Once the model is trained, it can be used for making predictions on new data points.
Key advantages of SVM include its ability to handle high-dimensional data and its resistance to overfitting. SVM can also work well with small to medium-sized datasets. However, SVM may become computationally expensive for very large datasets.
Python's scikit-learn library provides an efficient implementation of SVM, making it easy to use and apply to various classification and regression tasks.

Implementation:

```
#Support Vector Machine
from sklearn.svm import SVC
classifier4 = SVC()
classifier4.fit(x_train,y_train)
```

Random Forest: Random Forest is an ensemble machine learning algorithm that is widely used for classification and regression tasks. It is an extension of the decision tree algorithm and combines multiple decision trees to make more accurate predictions.

The main idea behind Random Forest is to create a multitude of decision trees during the training phase. Each decision tree is trained on a random subset of the data (sampling with replacement), and at each node of the tree, only a random subset of features is considered for splitting. This randomness helps to introduce diversity among the individual trees, reducing the risk of overfitting and improving the overall performance of the model.

During prediction, the Random Forest aggregates the predictions of all individual decision trees and outputs the mode (for classification) or the average (for regression) of these predictions. This ensemble approach helps to improve accuracy and generalization, as the errors made by individual trees tend to cancel out or be outweighed by the correct predictions.

Key advantages of Random Forest include its ability to handle high-dimensional data, deal with both numerical and categorical features, and handle missing values without requiring imputation. Additionally, Random Forest is less prone to overfitting compared to a single decision tree, making it a robust and powerful algorithm.

Random Forest is easy to implement in Python using libraries like scikit-learn, which provide efficient and optimized implementations of the algorithm. Overall, Random Forest is a versatile and effective tool for a wide range of machine learning tasks.

Implementation:

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
classifier5 = RandomForestClassifier()
classifier5.fit(x_train,y_train)
```

K-Nearest Neighbors (KNN): K-Nearest Neighbors (KNN) is a simple, yet effective machine learning algorithm used for classification and regression tasks. It is a non-parametric and lazy learning algorithm, meaning it does not make any assumptions about the underlying data distribution and does not perform a traditional training phase.

The choice of the value of k is crucial in the KNN algorithm. A small value of k may lead to noisy predictions and higher variance, while a large value of k may lead to oversmoothing and lower model complexity.

During prediction, it finds the k nearest labeled data points to the new data point based on a distance metric and assigns the most common class (for classification) or the average value (for regression) among those neighbors as the prediction.

KNN is very intuitive and easy to implement. It is particularly useful when dealing with small to medium-sized datasets and can handle both numerical and categorical features. However, as the size of the dataset increases, the computation cost of finding the k nearest neighbors also increases, making KNN less efficient for large datasets.

One of the challenges of KNN is selecting an appropriate distance metric, as this can greatly affect the algorithm's performance. Additionally, KNN may struggle with high-dimensional data, as the distance metric becomes less reliable in higher dimensions (this is known as the curse of dimensionality).
Python's scikit-learn library offers a convenient implementation of KNN, making it useful for various machine learning tasks, particularly when data distribution is uncertain, or the decision boundary is non-linear.

Implementation:

```
#K Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
classifier6 = KNeighborsClassifier(n_neighbors=2)
classifier6.fit(x_train,y_train)
```

Ensemble Classifier-

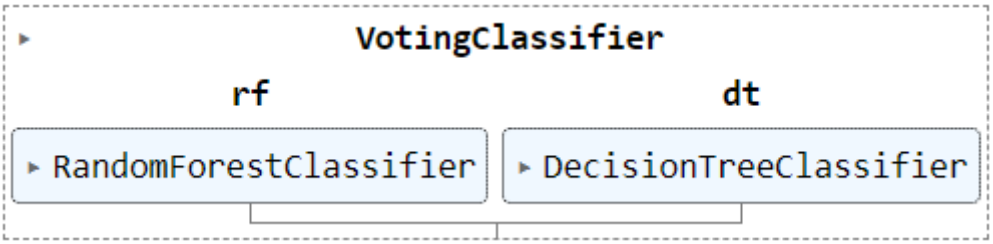
An ensemble classifier is a machine learning model that combines the predictions of multiple individual models (base learners) to make more accurate and robust predictions. Ensemble methods are widely used in machine learning to improve predictive performance, especially in situations where single models might struggle or exhibit limitations.
There are several types of ensemble classifiers, each with its own characteristics and techniques for combining base learners. Here are the main types of ensemble classifiers:
1. Bagging: Bagging (Bootstrap Aggregating) involves training multiple instances of the same base learner on different subsets of the training data. The final prediction is obtained by aggregating the predictions from all base learners. The most popular example of bagging is the Random Forest algorithm.
2. Boosting: Boosting is an iterative technique where the base learner is trained sequentially, with each learner focusing on correcting the errors made by its predecessor. During each iteration, the algorithm gives more weight to misclassified data points, allowing the subsequent learner to focus on those instances. Examples of boosting algorithms include AdaBoost (Adaptive Boosting) and Gradient Boosting Machines (GBM).
3. Voting: Voting classifiers combine the predictions of multiple base learners by taking a majority vote (hard voting) or a weighted average of their probabilities (soft voting). This method is often used for classification tasks with different classifiers.
4. Stacking: Stacking involves training multiple base learners and then using another model (the meta-learner) to combine their predictions. The base learners' predictions serve as features for the meta-learner, which learns to make the final prediction based on these features.

5. Blending: Blending is similar to stacking but involves splitting the training data into two parts: one part is used to train the base learners, and the other part is used to train the meta-learner. Blending is often used when there is a limited amount of labeled data available.

Ensemble classifiers can provide several benefits:

- Improved Performance: Ensemble methods can often outperform individual base learners, especially when the base learners have complementary strengths and weaknesses.
- Reduced Overfitting: Ensemble methods can help reduce overfitting, especially in complex models, by averaging out individual errors and reducing model variance.
- Robustness: Ensembles are more robust to noisy data and outliers, as errors in individual models are often smoothed out by the aggregation process.
- Flexibility: Ensemble methods can be applied to a wide range of base learners, making them versatile and applicable to various machine learning tasks.

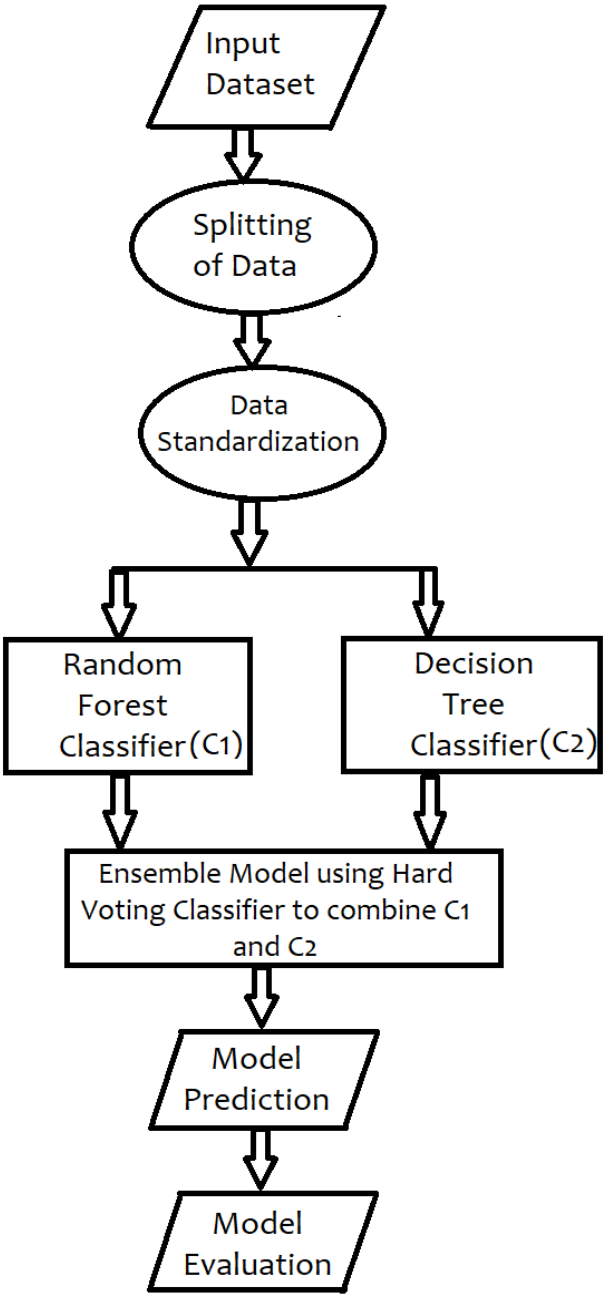
Popular machine learning libraries, such as scikit-learn in Python, provide implementations of ensemble classifiers making it easy to leverage their advantages in practical applications.



Voting Classifier is the most suitable ensemble classifier for our model since it is mainly used for classification and can combine the predictions of multiple base learners.

Proposed Model:

Recent research trends in predictive modeling focus on ensemble learning techniques, combining various classifiers to improve prediction accuracy. The study uses a max voting technique to determine weights for the base models. The work adopts the majority voting method to combine classifiers in a general architecture for the hybrid majority voting classifier. The voting classifier is used to combine the predictions of two classifiers namely Random Forest Classifier and Decision Tree Classifier.



4. Experiments and Results:

4.1 Evaluation of Model-

Confusion Matrix:

A confusion matrix is a performance evaluation metric used in binary and multi-class classification tasks. It provides a comprehensive summary of the predictions made by a machine learning model and how well they align with the actual target labels. The matrix is especially useful for understanding the types of errors made by the classifier.

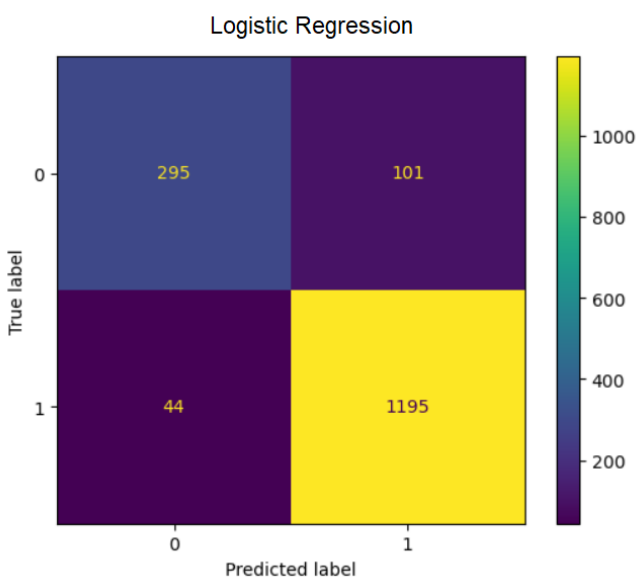
In a binary classification scenario, the confusion matrix has four key elements:

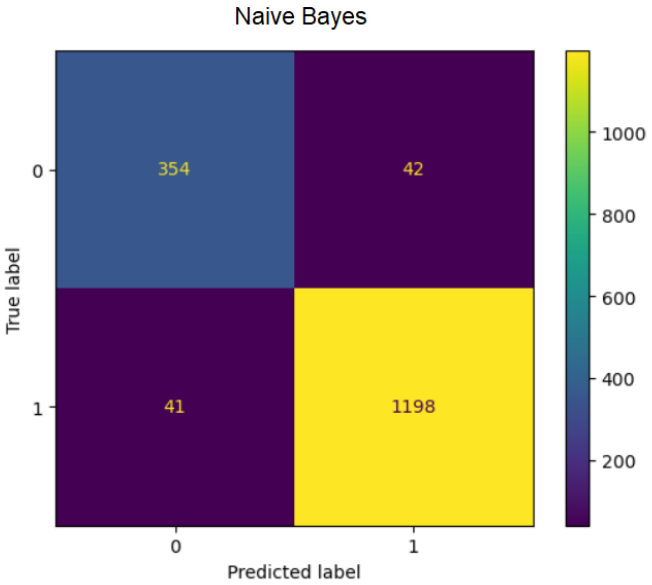
1. True Positive (TP): The number of instances correctly predicted as positive (belonging to the positive class).
2. True Negative (TN): The number of instances correctly predicted as negative (belonging to the negative class).
3. False Positive (FP): The number of instances wrongly predicted as positive when they actually belong to the negative class (Type I error or "false alarm").
4. False Negative (FN): The number of instances wrongly predicted as negative when they actually belong to the positive class (Type II error or "miss").

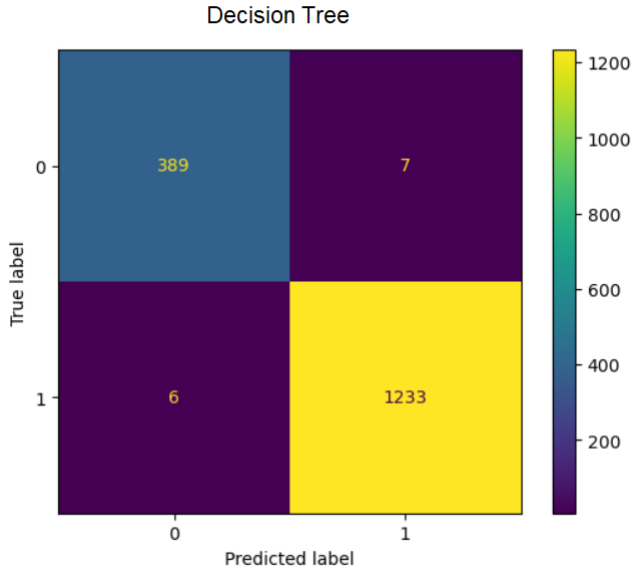
For a multi-class classification problem with `N` classes, the confusion matrix is an `N x N` matrix, where each row represents the instances in an actual class, and each column represents the instances predicted in that class. The diagonal elements of the matrix correspond to the number of correctly predicted instances for each class, while off-diagonal elements represent the misclassifications.

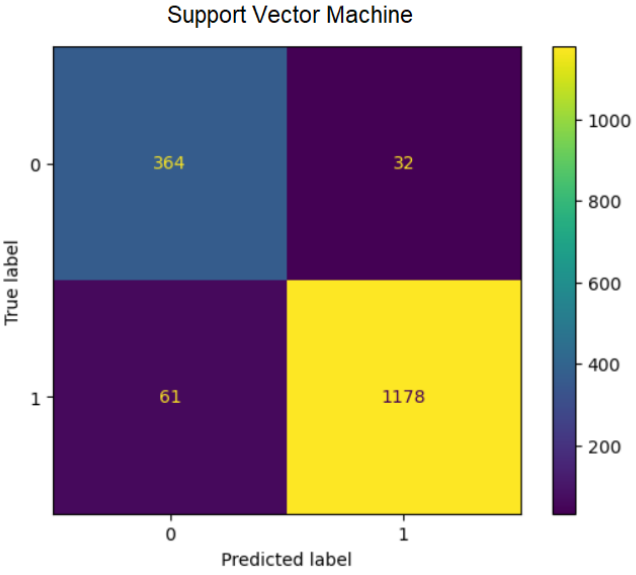
The confusion matrix provides valuable information to calculate various performance metrics like accuracy, precision, recall (sensitivity), specificity, F1-score, and more. It helps to understand the strengths and weaknesses of the model and identify areas of improvement.

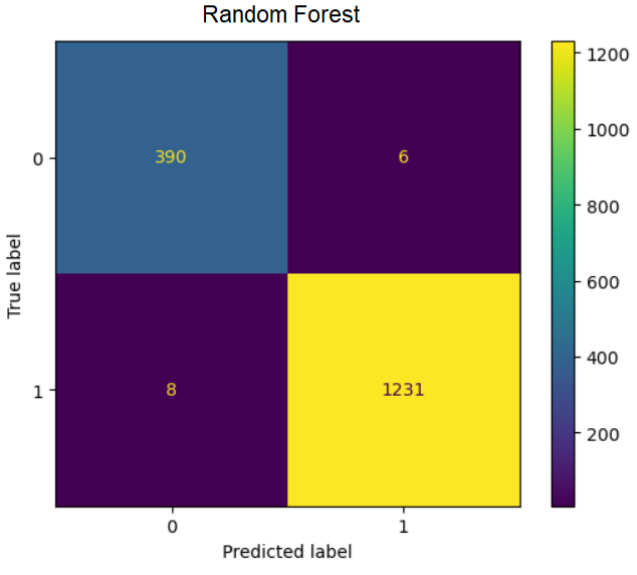
To calculate the confusion matrix in Python, you can use libraries such as scikit-learn.

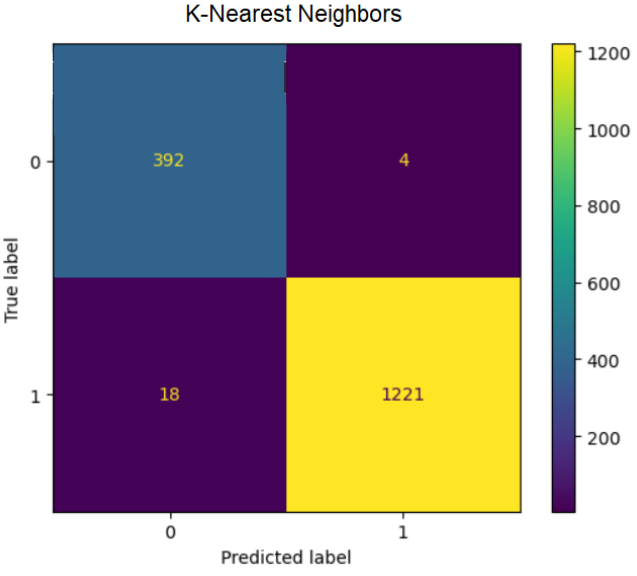

Logistic Regression
295
101
44
1195
True label
Predicted label
0
1
200
400
600
800
1000

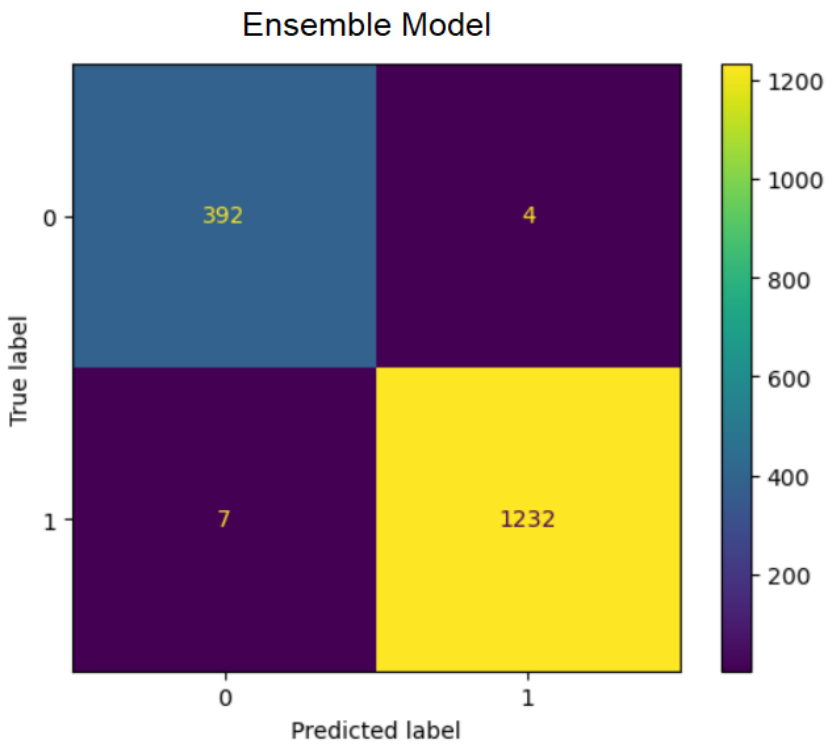

Naive Bayes
354
42
41
1198
True label
Predicted label
0
1
200
400
600
800
1000


Decision Tree
389
7
6
1233
True label
Predicted label
0
1
200
400
600
800
1000
1200


Support Vector Machine
364
32
61
1178
True label
Predicted label
0
1
200
400
600
800
1000


Random Forest
390
6
8
1231
True label
Predicted label
0
1
200
400
600
800
1000
1200


K-Nearest Neighbors
392
4
18
1221
True label
Predicted label
0
1
200
400
600
800
1000
1200



After comparing various classifiers like Logistic Regression, Naive Bayes, Decision Tree, Support Vector Machine(SVM), Random Forest, K- Nearest Neighbors(KNN) on the basis of confusion matrix, Random Forest and Decision Tree have the best result as individual classifier, whereas using Voting classifier for ensembling the predictions of Random Forest and Decision Tree provides the best result for hybrid classifier.

Model Accuracy:

Model accuracy is a performance metric used to evaluate the correctness of predictions made by a machine learning model. It measures the proportion of correct predictions made by the model over the total number of predictions. For binary classification, it is the ratio of the number of true positives and true negatives to the total number of instances. For multi-class classification, it is the ratio of the sum of correct predictions for all classes to the total number of instances.
The formula for calculating accuracy in binary classification is:

Accuracy = (True Positives + True Negatives) / (True Positives + False Positives + True Negatives + False Negatives)
In multi-class classification, accuracy can be calculated similarly, but it considers the sum of correct predictions for all classes.

Classifiers	Model Accuracy	Log Loss
Naive Bayes	0.9492354740061162	1.8297389793863756
Logistic Regression	0.9113149847094801	3.1965319519400532
Random Forest	0.9914373088685016	0.24249552738855598
Decision Tree	0.9920489296636086	0.33067571916621263
K-Nearest Neighbors	0.9865443425076452	0.4849910547771118
Support Vector Machine	0.9431192660550459	2.0501894588305167
Ensemble Model (NB, LR) (Voting)	0.9425076452599388	2.072234506774931
Ensemble Model (NB, RF)	0.9718654434250764	1.0140722054430515
Ensemble Model (NB, DT)	0.9718654434250764	1.0140722054430515
Ensemble Model (NB, KNN)	0.9688073394495413	1.1242974451651224
Ensemble Model (NB, SVM)	0.8978593272171254	3.6815230067171654
Ensemble Model (RF, DT)	0.9938837920489296	0.22045047944414184
Ensemble Model (RF, KNN)	0.9865443425076452	0.4849910547771118
Ensemble Model (RF, SVM)	0.9155963302752294	3.0422166163291546
Ensemble Model (DT, KNN)	0.9865443425076452	0.4849910547771118
Ensemble Model (DT, SVC)	0.9168195718654434	2.998126520440326
Ensemble Model (KNN, SVC)	0.9131498470948012	3.130396808106811
Ensemble Model (NB, LR, RF, DT, KNN)	0.9920489296636086	0.28658562327738435

Based on the Accuracy table given above, the Ensemble Model of Random Forest and Decision Tree appears to be the most accurate and most suitable classifier for building the fire detection model.

F1 Score:

F1 score is another performance metric used in binary classification and multi-class classification tasks. It provides a balance between precision and recall, making it particularly useful when dealing with imbalanced datasets where one class dominates the others.
In binary classification, the F1 score is the harmonic mean of precision and recall, and it is calculated using the following formula:

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
where:
- Precision is the ratio of true positives to the sum of true positives and false positives. It measures the proportion of positive instances that were correctly identified.
- Recall (also known as sensitivity or true positive rate) is the ratio of true positives to the sum of true positives and false negatives. It measures the proportion of positive instances that were correctly identified out of all the actual positive instances.

In multi-class classification, the F1 score can be calculated for each class separately, and then it is typically averaged over all classes to obtain a macro-average F1 score or weighted average F1 score.
A higher F1 score indicates better overall performance, with 1 being the best score and 0 being the worst.
In Python, you can calculate the F1 score using libraries like scikit-learn.

Classifiers	**F1 Score**
Naive Bayes	0.9307941449006037
Logistic Regression	0.8727609386950046
Random Forest	0.988356738543926
Decision Tree	0.9891605287193717
K-Nearest Neighbors	0.9818880716058136
Support Vector Machine	0.9243744121864352
Ensemble Model(Random Forest, Decision Tree)	0.9908595387840671

RMSE:

Root Mean Squared Error (RMSE) is a commonly used metric for evaluating the performance of regression models. It measures the average magnitude of the differences between the predicted values and the actual target values. RMSE is particularly useful because it penalizes large errors more heavily than small errors, making it sensitive to outliers.
The RMSE is calculated as follows:

RMSE = sqrt ((sum of (predicted value - actual value) ^2) / number of samples)
where:
- The "predicted value" represents the predicted output from the regression model.
- The "actual value" represents the true target value.
- The "sum of (predicted value - actual value) ^2" is the sum of the squared differences between the predicted values and the actual target values across all samples.
- The "number of samples" is the total number of data points.

A lower RMSE indicates better model performance, where 0 represents a perfect fit (the predicted values match the actual values exactly).
In Python, you can calculate the RMSE using libraries like NumPy.

Classifiers	**RMSE**
Naive Bayes	0.2253098444229275
Logistic Regression	0.297800294308988
Random Forest	0.09253481037695203
Decision Tree	0.08916877444706435
K-Nearest Neighbors	0.11599852366454816
Support Vector Machine	0.23849682166635708
Ensemble Model (Random Forest, Decision Tree)	0.08202334269083021

5. Conclusion:

In conclusion, an ensemble model has been proposed for fire detection by employing the maximum voting technique. To effectively identify fires, two distinct classifiers, the Random Forest Classifier, and the Decision Tree Classifier, have been combined and put to use. The chosen dataset encompasses a wide array of vital information, including precise measurements of CO, CO2, and O2 concentrations, smoke obscuration levels, and temperature readings from various locations within the structure.

The suggested machine learning approach has undergone various performance evaluations on the NIST dataset. The published studies and compelling results have revealed the ensemble classifier's superiority over its individual components in multiple aspects. Furthermore, the confusion matrix, model accuracy, F1 Score, precision as well as reduction of RMSE error and log loss have all been significantly improved, paving the way for a more reliable and efficient fire detection system.

References:

1) **M. D. Molovtsev and I. S. Sineva**
   Classification Algorithms Analysis in the Forest Fire Detection Problem
   2019 International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS)
   Sochi, Russia, 2019, pp. 548-553
   doi: 10.1109/ITQMIS.2019.8928398.
2) **G. Sulistian, M. Abdurohman and A. G. Putrada**
   Comparison of Classification Algorithms to Improve Smart Fire Alarm System Performance
   2019 International Workshop on Big Data and Information Security (IWBIS)
   Bali, Indonesia, 2019, pp. 119-124
   doi: 10.1109/IWBIS.2019.8935885.
3) **S. Jana, S. K. Shome**
   Hybrid Ensemble Based Machine Learning for Smart Building Fire Detection Using Multi Modal Sensor Data
   Fire Technol 59, 473–496 (2023)
   doi.org/10.1007/s10694-022-01347-7
4) **R. Sharma, S. Rani, I. Memon**
   A smart approach for fire prediction under uncertain conditions using machine learning.
   Multimedia Tools and Applications (2020)
   79. 10.1007/s11042-020-09347-x.
5) **B. S. Negara, R. Kurniawan, M. Z. A. Nazri, S. N. H. S. Abdullah, R. W. Saputra, A. Ismanto**
   Riau Forest Fire Prediction using Supervised Machine Learning
   Journal of Physics: Conference Series. 1566. 012002 (2020) 10.1088/1742-6596/1566/1/012002.

6) T. A. Pratiwi, M. Irsyad, R. Kurniawan
Classification of Forest Fires and Land Using the Naïve Bayes Algorithm (Case Study: Riau Province)
Journal of Information Systems and Technology (JustIN)(2021)
9. 101. 10.26418/justin.v9i2.42823.
7) M. Grari, M. Yandouzi, I. Idrissi, M. Boukabous, O. Moussaoui, M. Azizi, M. Moussaoui
USING IOT AND ML FOR FOREST FIRE DETECTION, MONITORING, AND PREDICTION: A LITERATURE REVIEW
Journal of Theoretical and Applied Information Technology. 100. 5445-5461 (2022)
8) U. Dampage, L. Bandaranayake, R. Wanasinghe, K. Kottahachchi, B. Jayasanka
Forest fire detection system using wireless sensor networks and machine learning
Scientific Reports (2022) 12. 10.1038/s41598-021-03882-9.
9) B. T. Pham, A. Jaafari, M. Avand, N. Al-Ansari, T. D. Du, H. P. H. Yen, T. V. Phong, D. H. Nguyen, H. V. Lê, D. Mafi-Gholami, I. Prakash, H. T. Thuy, T. T. Tuyen
Performance Evaluation of Machine Learning Methods for Forest Fire Modeling and Prediction
Symmetry (2020)
10) K. Bot, J. G. Borges
A Systematic Review of Applications of Machine Learning Techniques for Wildfire Management Decision Support
(2022) Inventions. 7. 15. 10.3390/inventions7010015.
11) R. Vikram, D.Sinha
A multimodal framework for Forest fire detection and monitoring.
Multimedia Tools and Applications (2022)
82. 1-24. 10.1007/s11042-022-13043-3.
12) R. Asad, S. Altaf, S. Ahmad, H. Mahmoud, S. Huda, S. Iqbal
Machine Learning-Based Hybrid Ensemble Model Achieving Precision Education for Online Education Amid the Lockdown Period of COVID-19 Pandemic in Pakistan
Sustainability 15, no. 6: 5431 (2023)
doi.org/10.3390/su15065431
13) D. N. Mhawi, A. Aldallal, S. Hassan
Advanced Feature-Selection-Based Hybrid Ensemble Learning Algorithms for Network Intrusion Detection Systems
Symmetry 14, no. 7: 1461 (2022)
doi.org/10.3390/sym14071461
14) N. Mhawi, Doaa & Hashim, Sokeana
Proposed Hybrid Ensemble Learning Algorithms for an Efficient Intrusion Detection System
(2022) 22. 73-84. 10.33103/uot.ijccce.22.2.7.