Comparison of KNN and Decision Tree Algorithms for Classification in Fire Detection Systems using IoT Sensor Data

Maria Diaz Alba
Machine Learning and IoT
Florida International University, Miami, Florida, US
mdiaz683@fiu.edu

***Abstract*—The objective of this study is to compare the performance of the K-Nearest Neighbors (KNN) and Decision Tree algorithms in predicting discrete categories, specifically within the context of a fire detection system using IoT sensor data.**

***Index Terms*—Machine Learning, K-Nearest Neighbors, Decision Tree, Classification, IoT, Fire Detection**

I. Introduction

Supervised learning is one of the most widely used approaches in Machine Learning (ML) where the model is trained on a labeled dataset to learn the relationships between input features and a target variable. Both are provided during the training process, enabling the model to learn and make predictions on new unseen data. ML tools address two main statistical tasks – classification (pattern recognition) and regression (function approximation). [1]

Classification tasks involve predicting a discrete label or category. The model assigns an input to one of several predefined classes based on learned patterns from the training data. Classification is essential in problems where the goal is to categorize data into distinct groups, such as spam detection, disease diagnosis, or image recognition.

Some machine learning algorithms, such as K-Nearest Neighbors (KNN) and Decision Trees, are versatile and can be applied to both classification and regression problems. This study focuses on classification tasks and compares the performance of KNN and Decision Trees, specifically in the context of a fire detection system.

The objective of the study is to compare the performance of KNN and Decision Tree algorithms in predicting discrete categories in the context of a fire detection system. Both algorithms are applied to classify whether a fire alarm will be triggered based on sensor readings. By analyzing their effectiveness in classification, we aim to determine which algorithm offers better performance for this specific prediction task.

II. Dataset Overview

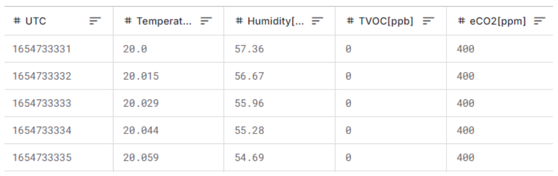
In recent years, the rise of the Internet of Things (IoT) has significantly enhanced applications like smart home systems, industrial monitoring, and fire detection. IoT devices are equipped with multiple sensors that can gather a variety of environmental data in real time, such as temperature, humidity, and gas concentrations, allowing for more efficient monitoring and predictive analytics. In fire detection systems, sensor data from IoT devices can be used to identify early signs of smoke or fire hazards, enabling automated alarm systems that improve response times and safety measures. [2]

In this project, we use IoT sensor data to classify whether a fire alarm should be triggered, based on a series of sensor readings. The dataset used in this experiment was obtained from Kaggle.com [3] and is titled ”Smoke Detection IoT Dataset.” The dataset includes various sensor readings related to air quality, gas concentration, and environmental factors. These features are typically measured in real-time by IoT sensors in environments where smoke detection is critical, such as residential buildings, factories, or public spaces.

The dataset contains the following features (Sensor readings): UTC, Temperature[C], Humidity[%], TVOC[ppb], eCO2[ppm], Raw H2, Raw Ethanol, Pressure[hPa], PM1.0, PM2.5, NC0.5, NC1.0, NC2.5, CNT(sample counter)

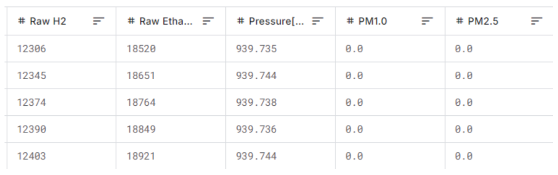
The target variable is **Fire Alarm**, where 1 indicates the alarm is triggered and 0 indicates no alarm.

Figures 1, 2 and 3 show a short overview of all of them.



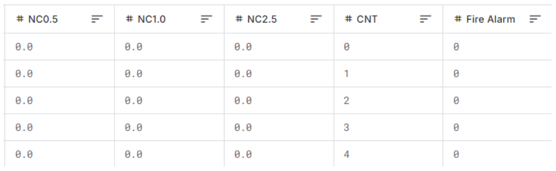
# UTC	# Temperat...	# Humidity[...	# TVOC[ppb]	# eCO2[ppm]
1654733331	20.0	57.36	0	400
1654733332	20.015	56.67	0	400
1654733333	20.029	55.96	0	400
1654733334	20.044	55.28	0	400
1654733335	20.059	54.69	0	400

Fig. 1. First five features of the dataset



# Raw H2	# Raw Etha...	# Pressure[...	# PM1.0	# PM2.5
12306	18520	939.735	0.0	0.0
12345	18651	939.744	0.0	0.0
12374	18764	939.738	0.0	0.0
12390	18849	939.736	0.0	0.0
12403	18921	939.744	0.0	0.0

Fig. 2. Second five features of the dataset



# NC0.5	# NC1.0	# NC2.5	# CNT	# Fire Alarm
0.0	0.0	0.0	0	0
0.0	0.0	0.0	1	0
0.0	0.0	0.0	2	0
0.0	0.0	0.0	3	0
0.0	0.0	0.0	4	0

Fig. 3. Last four features of the dataset and the target

III. Class Distribution

Initially, the dataset had a class imbalance with more instances of the fire alarm being triggered than not. Since machine learning models tend to favor the majority class in such situations, we balanced the dataset by under-sampling the majority class (Fire Alarm = 1) and selecting an equal number of samples from both classes. This allowed us to use methods like KNN and Decision Trees without the models being biased towards the majority class.

In this case, after balancing, the dataset contained:

- 10,000 samples for Fire Alarm = 0
- 10,000 samples for Fire Alarm = 1

IV. Data Preprocessing

A. Data Cleaning

Before applying machine learning algorithms, it is crucial to clean the dataset to ensure that only relevant features are used in the prediction process. In this dataset, one column was identified as irrelevant for the prediction task:

- CNT: A simple identifier column that increments for each record.

This column was dropped to avoid adding unnecessary noise to the models, as it does not provide any predictive value for the fire alarm classification task.

Additionally, the dataset was reviewed for any missing values (NaNs) to ensure data integrity.

B. Feature Scaling

Properly preparing the data ensures that machine learning algorithms can efficiently learn and produce accurate predictions.

Since KNN is a distance-based algorithm, feature scaling was crucial. We used *StandardScaler* to standardize the sensor readings to ensure that each feature contributed equally to the distance calculation.

Decision Trees are not affected by the scale of the features, but since we were using KNN, scaling was performed for both models to maintain consistency.

C. Train-Test Split

We split the dataset into training and test sets using a 70-30 split ratio. This is a common practice in machine learning, as it reserves 30% of the data for testing while using 70% to train the model. This ensures that the model is evaluated on unseen data, giving a better indication of its real-world performance.

We used the *random state=42* parameter to ensure the split is reproducible, allowing for repeatable results and consistent evaluations when tuning or comparing models. Without setting this parameter, each run would produce a slightly different split, which could lead to variations in model performance.

V. Modeling and Hyperparameter Tuning

To systematically search for the best combination of hyperparameters, we use **GridSearchCV**. This method tests every possible combination of the hyperparameter values provided in a grid and selects the one that performs best according to a chosen evaluation metric. [4]

GridSearchCV also employs **cross-validation**, which splits the training data into multiple subsets (folds), trains the model on different combinations of those subsets, and then evaluates the model's performance. So, GridSearchCV helped us optimize both the KNN and Decision Tree models by automatically finding the best values for hyperparameters.

A. KNN Classifier

The parameters we explored for KNN were:

- **n_neighbors**: The number of neighbors KNN uses to classify each data point. Fewer neighbors might cause overfitting, while too many neighbors could lead to underfitting.
- **weights**: We tested both uniform (all neighbors have equal weight) and distance (closer neighbors have more influence).
- **metric**: The distance metric used to compute the distances between points. We explored both Euclidean and Manhattan distance metrics.

By applying GridSearchCV, we tested all possible combinations of these hyperparameters. The best ones found by the grid search to provide the optimal performance during 5-fold cross-validation in this case were determined to be:

- **n_neighbors**: 3
- **weights**: Uniform
- **metric**: Euclidean

B. Decision Tree Classifier

Similarly, we used *GridSearchCV* for the Decision Tree, exploring the following hyperparameters:

- **max_depth**: The maximum depth of the tree. Limiting the depth prevents overfitting, while allowing deeper trees increases model complexity.
- **max_features**: The number of features considered for each split.

By applying GridSearchCV and tested various combinations of them it was found that the optimal parameters were:

- **max_depth**: 3
- **max_features**: All features

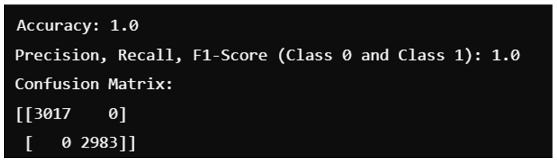
VI. Error Metrics

By using multiple metrics, we can gain a comprehensive understanding of the strengths and weaknesses of each algorithm in different scenarios. The following metrics were selected to measure the performance of the models:

- **Accuracy**: Proportion of correctly predicted instances out of the total number of predictions.
- **Precision**: Proportion of positive predictions (Fire Alarm = 1) that were actually correct.
- **Recall**: Also known as sensitivity or true positive rate; measures how many of the actual positives (Fire Alarm = 1) were correctly identified by the model
- **F1-score**: Is the harmonic mean of precision and recall.
- **Confusion Matrix**: Table that summarizes the performance of a classification model by comparing the predicted classes against the true classes.

A. KNN Performance

The KNN classifier performed exceptionally well on this dataset, likely due to the simplicity or separability of the data obtained from Kaggle. After training, the model achieved perfect classification results, meaning it correctly predicted both classes for all test data. However, in more complex or noisy datasets, KNN might not perform as perfectly. Figure 4 shows the metrics obtained for this algorithm.

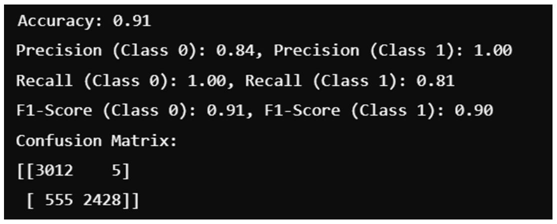


```
Accuracy: 1.0
Precision, Recall, F1-Score (Class 0 and Class 1): 1.0
Confusion Matrix:
[[3017    0]
 [   0 2983]]
```

Fig. 4. KNN Error Metrics

B. Decision Tree Performance

The Decision Tree achieved 91% accuracy on the test set, but had a slightly lower recall (81%) for class 1, indicating some missed fire alarms. Figure 5 shows the metrics obtained for this algorithm.



```
Accuracy: 0.91
Precision (Class 0): 0.84, Precision (Class 1): 1.00
Recall (Class 0): 1.00, Recall (Class 1): 0.81
F1-Score (Class 0): 0.91, F1-Score (Class 1): 0.90
Confusion Matrix:
[[3012    5]
 [ 555 2428]]
```

Fig. 5. Decision Tree Error Metrics

These metrics provide a general idea of the performance of both models in this scenario, as shown in Figure 6, where this is graphically represented.

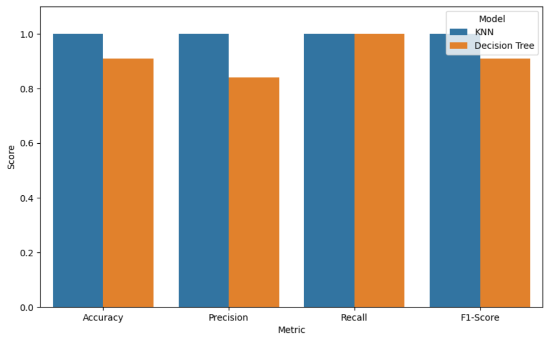


Fig. 6. Comparison of Classification metrics for KNN and Decision Tree

VII. General Comparison of Algorithms

While KNN was able to perfectly classify all instances, likely due to the simplicity or separability of the dataset, it can be computationally expensive for large datasets and may not generalize well if the dataset were more complex or noisy.

Decision Trees are easy to interpret and computationally efficient. The optimized model was able to achieve a reasonable performance after tuning, but without further tuning, the Decision Tree may still suffer from slight overfitting or underfitting issues.

VIII. Conclusion

Based on the experiment, the KNN classifier significantly outperformed the Decision Tree in this specific dataset. The perfect classification results from KNN could be attributed to the dataset's separability. However, such perfect performance may also indicate overfitting, as achieving 100% accuracy, precision, recall, and F1-score across all classes could suggest that KNN is too tightly fitted to this dataset. In real-world scenarios with more noise or variability, KNN may not generalize as well.

For this dataset:

- **KNN** was the better performing model, achieving a perfect classification rate. However, this may raise concerns of overfitting, and further testing on more complex datasets would be recommended.
- **Decision Tree** was a more interpretable but slightly less accurate model, with a tendency to miss some true positives for Class 1.

References

[1] Y. Bengio, A. Courville and I. Goodfellow, *Deep Learning*. MIT Press, 2016.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, ”Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645-1660, Sept. 2013. [Online]. Available: https://doi.org/10.48550/arXiv.1207.0203

[3] Kaggle, ”Smoke Detection IoT Dataset,” 2023. [Online]. Available: https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset [Accessed: Feb. 6, 2025].

[4] Scikit-learn developers, ”sklearn.model_selection.GridSearchCV”, 2025. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV [Accessed: Feb. 6, 2025].