`

MSE413:Application of Machine Learning in Mechatronics

Course Project

Group 18

Zafeer Abbasi 301394323

Hesham Khan 301401563

Alexander Maljaars 301478980

Table of Contents

1. Introduction

A Smoke detector is a device that can detect smoke, and smoke is typically used to indicate an active fire. Often, smoke detectors are cased in plastic enclosures which are usually the shape of a disk, 150 millimeters in diameter and 25 millimeters in thickness, however, these measurements are just estimates and the actual size may vary. These devices have saved many lives. For example, in France from 1982 to 2012, the number of fire victims fell by more than 48%, and in the UK from 1982 to 2013, the number of fire victims fell by 56% (“Fire Safety Statistics”). These reductions can largely be linked to increased national fire safety measures such as the installation of smoke alarms. Furthermore, in the U.S. 96% of all homes have smoke alarms and approximately 20% of those homes have non-operational smoke alarms. It is estimated that if all homes were equipped with operational smoke alarms, the number of deaths from residential fires in the U.S. could decrease by 36%, which equates to saving 1100 lives annually (“Smoke Alarm Research | NIST”).

In this project, we aim to use multiple sensor readings to predict the behavior of a smoke detector with different classification methods and determine which algorithms with which pre-processing techniques and hyper-parameter configurations perform the best. The source of our data can be found here:

GitHub - Blatts01/ai-smokedetector: use ai sensor fusion for smoke detection

2. Data Discovery

Upon initial inspection, we removed unnecessary columns such as ‘Index’, ‘UTC’, and ‘CNT’. The output of ‘data.head( )’ is shown below.

	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
0	20.000	57.36	0	400	12306	18520	939.735	0.0	0.0	0.0	0.0	0.0	0
1	20.015	56.67	0	400	12345	18651	939.744	0.0	0.0	0.0	0.0	0.0	0
2	20.029	55.96	0	400	12374	18764	939.738	0.0	0.0	0.0	0.0	0.0	0
3	20.044	55.28	0	400	12390	18849	939.736	0.0	0.0	0.0	0.0	0.0	0
4	20.059	54.69	0	400	12403	18921	939.744	0.0	0.0	0.0	0.0	0.0	0

A brief table explaining each column is presented below.

Temperature [°C]	Air Temperature in Degrees Celsius.
Humidity [%]	Percentage of Humidity in the Air.
TVOC [ppb]	Total Volatile Organic Compounds, measured in parts per billion. These are emitted as gasses from certain solids or liquids and include a variety of chemicals.
$eC0_2$ [ppm]	Equivalent Carbon Dioxide, measured in parts per million. It is an estimation of the C02 level in the air and is calculated from different values, such as TVOC.
Raw H2	Raw Molecular Hydrogen from sensor readings; no compensations.
Raw Ethanol	Raw Ethanol Gas, similar to Raw H2.
Pressure [hPa]	Air Pressure, measured in hectopascals.
PM1.0	Particle Matter, under 1.0um
PM2.5	Particle Matter, between 1.0um and 2.5um
NC0.5	Number Concentration of particles in the air under 0.5um
NC1.0	Number Concentration of particles in the air between 0.5um and 1.0 um
NC2.5	Number of particles in the air between 1.0um and 2.5um

From the data source, we know that the following sensors were used to capture this data. Note that there are some redundant sensors to improve data quality:

Bosch BMP390	Pressure	$6.33[1]

Bosch BMP388		$6.33[1]
Bosch BME688	Humidity, Temperature, TVOC, Raw H2, Raw Ethanol	$18.19[1]
Sensirion SPS30	Particle Sensor	$63.30[1]
Sensirion SHT31	Humidity, Temperature	$8.33[1]
Sensirion SPG30	Gas Sensor: TVOC, eCO_2	$10.16[2]

(1) Prices from Digikey.ca in CAD

Ideally, some of these sensors could be removed to reduce the cost of the device.

Below we've presented the analysis of the data exploration.

2.1 Shape & Size

```
Shape: (62630, 13)
Size:  814190
```

2.2 .info( )

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62630 entries, 0 to 62629
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Temperature[C]  62630 non-null  float64
 1   Humidity[%]     62630 non-null  float64
 2   TVOC[ppb]       62630 non-null  int64  
 3   eCO2[ppm]       62630 non-null  int64  
 4   Raw H2          62630 non-null  int64  
 5   Raw Ethanol     62630 non-null  int64  
 6   Pressure[hPa]   62630 non-null  float64
 7   PM1.0           62630 non-null  float64
 8   PM2.5           62630 non-null  float64
 9   NC0.5           62630 non-null  float64
 10  NC1.0           62630 non-null  float64
 11  NC2.5           62630 non-null  float64
 12  Fire Alarm      62630 non-null  int64  
dtypes: float64(8), int64(5)
memory usage: 6.2 MB
```

As we can see from the above output, we do not have any null or missing values in our dataset.

2.3 Skewness

	Feature	Mean	Median
0	Temperature[C]	15.970424	20.130
1	Humidity[%]	48.539499	50.150
2	TVOC[ppb]	1942.057528	981.000
3	eCO2[ppm]	670.021044	400.000
4	Raw H2	12942.453936	12924.000
5	Raw Ethanol	19754.257912	19501.000
6	Pressure[hPa]	938.627649	938.816
7	PM1.0	100.594309	1.810
8	PM2.5	184.467770	1.880
9	NC0.5	491.463608	12.450
10	NC1.0	203.586487	1.943
11	NC2.5	80.049042	0.044
12	Fire Alarm	0.714626	1.000

From the above results we can see that:

TVOC, eC02, PM1, PM2, NC0, NC1, and NC2 are Right-Skewed.

This indicates that for the above variables, there are a few extremely high values and the bulk of the data is concentrated on the left.

2.4 Correlation Matrix

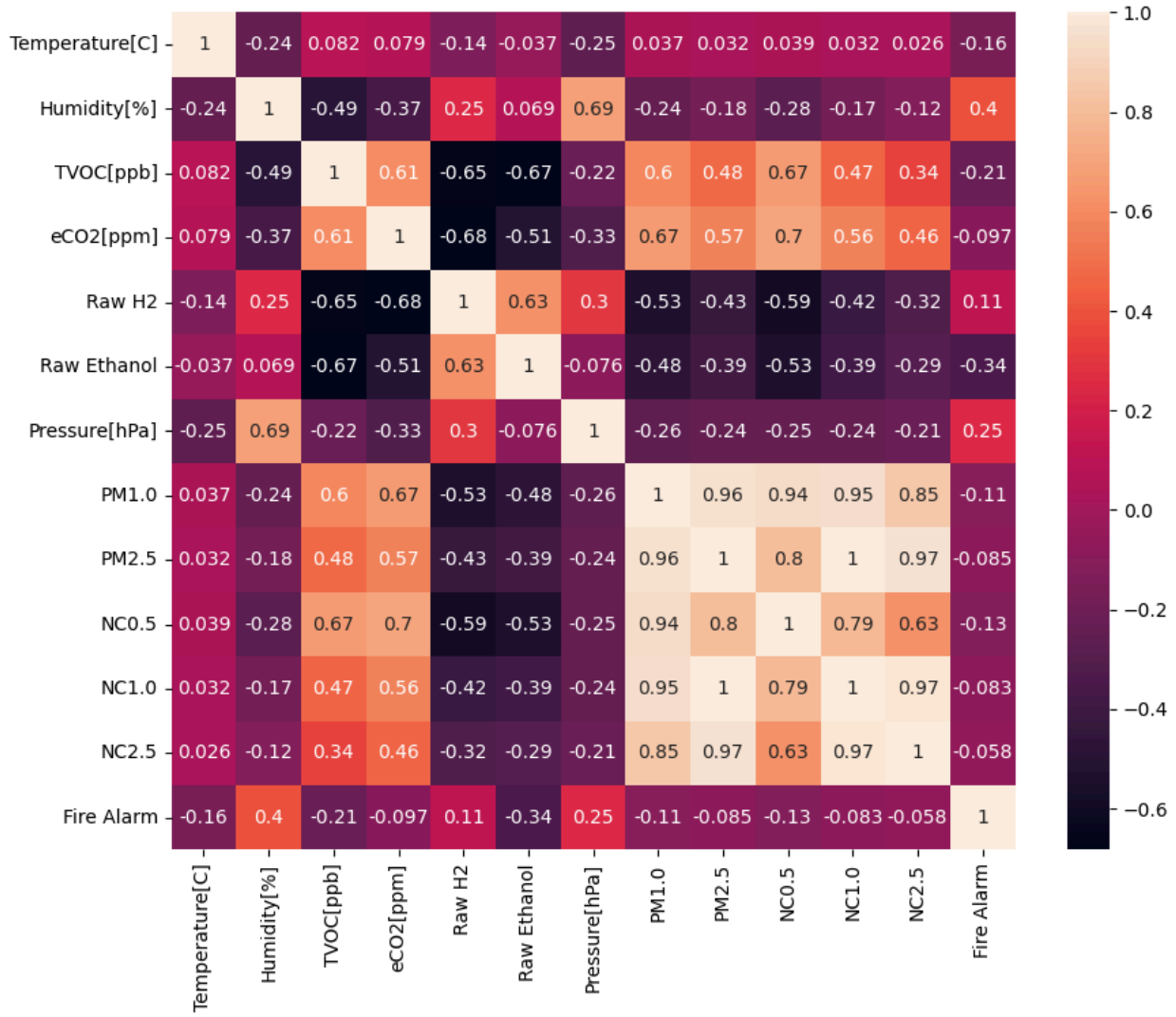
	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
Temperature[C]	1.000000	-0.243986	0.082442	0.079265	-0.135540	-0.037343	-0.251203	0.037330	0.032084	0.039291	0.031608	0.025526	-0.163902
Humidity[%]	-0.243986	1.000000	-0.488878	-0.369095	0.247199	0.068782	0.694614	-0.236538	-0.178882	-0.277389	-0.174224	-0.118502	0.399846
TVOC[ppb]	0.082442	-0.488878	1.000000	0.606118	-0.653053	-0.673715	-0.220004	0.597366	0.477424	0.670657	0.467386	0.344721	-0.214743
eCO2[ppm]	0.079265	-0.369095	0.606118	1.000000	-0.682785	-0.506695	-0.326043	0.665482	0.572691	0.699604	0.564252	0.456323	-0.097006
Raw H2	-0.135540	0.247199	-0.653053	-0.682785	1.000000	0.631495	0.303090	-0.530714	-0.431079	-0.587769	-0.422626	-0.318518	0.107007
Raw Ethanol	-0.037343	0.068782	-0.673715	-0.506695	0.631495	1.000000	-0.075926	-0.480698	-0.393192	-0.529186	-0.385720	-0.293351	-0.340652
Pressure[hPa]	-0.251203	0.694614	-0.220004	-0.326043	0.303090	-0.075926	1.000000	-0.258938	-0.243071	-0.248657	-0.241148	-0.213390	0.249797
PM1.0	0.037330	-0.236538	0.597366	0.665482	-0.530714	-0.480698	-0.258938	1.000000	0.956118	0.940045	0.949860	0.854901	-0.110552
PM2.5	0.032084	-0.178882	0.477424	0.572691	-0.431079	-0.393192	-0.243071	0.956118	1.000000	0.798873	0.999787	0.969382	-0.084916
NC0.5	0.039291	-0.277389	0.670657	0.699604	-0.587769	-0.529186	-0.248657	0.940045	0.798873	1.000000	0.786274	0.626711	-0.128118
NC1.0	0.031608	-0.174224	0.467386	0.564252	-0.422626	-0.385720	-0.241148	0.949860	0.999787	0.786274	1.000000	0.974249	-0.082828
NC2.5	0.025526	-0.118502	0.344721	0.456323	-0.318518	-0.293351	-0.213390	0.854901	0.969382	0.626711	0.974249	1.000000	-0.057707
Fire Alarm	-0.163902	0.399846	-0.214743	-0.097006	0.107007	-0.340652	0.249797	-0.110552	-0.084916	-0.128118	-0.082828	-0.057707	1.000000

From the results presented above, we can observe a few things:

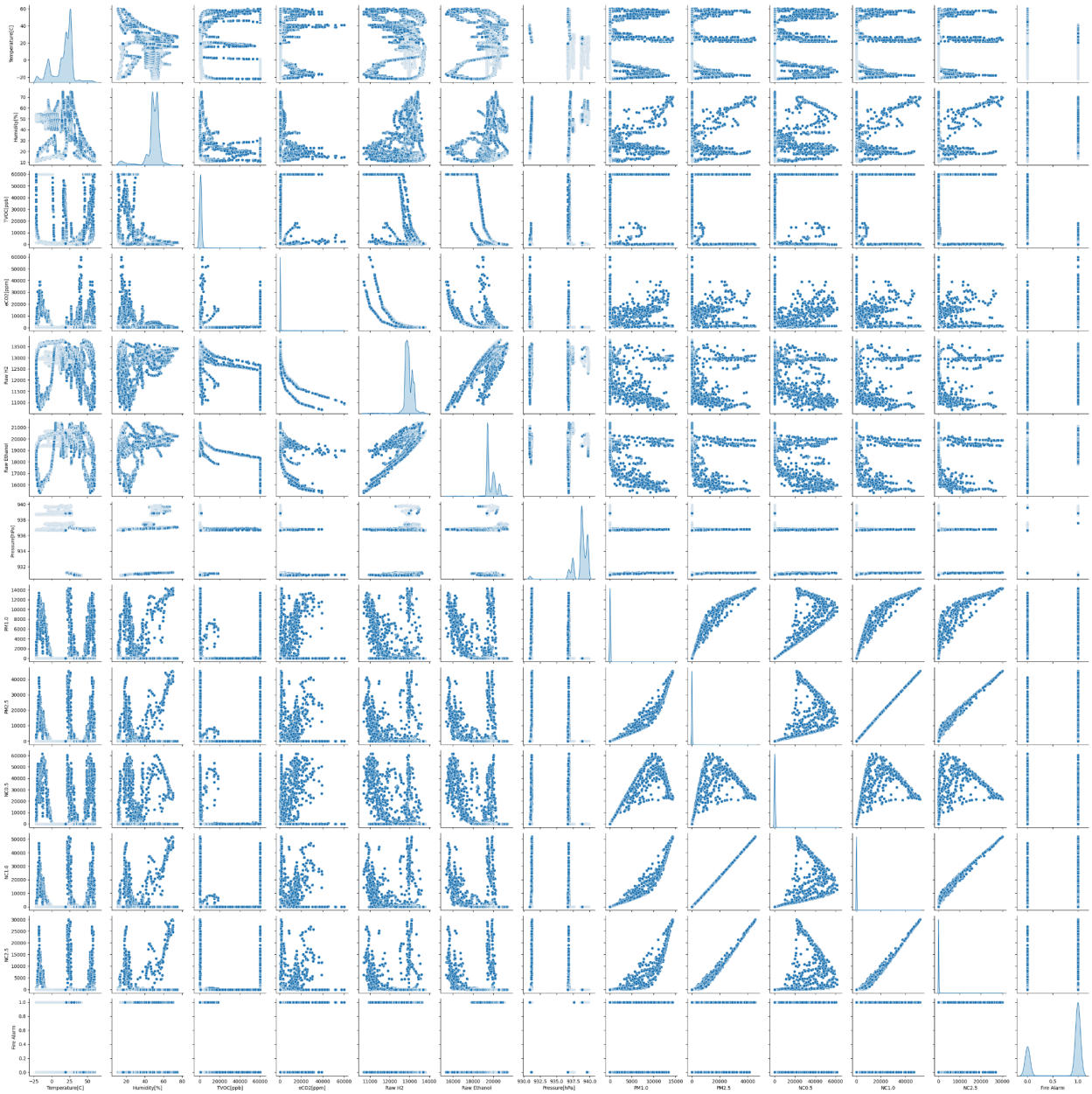
1. TVOC and Raw Ethanol have a high negative correlation.
2. eC02 and Raw H2 have a high negative correlation.

3. PM1.0 and PM2.5 have a very high positive correlation. This is expected since PM2.5 in itself includes particles of PM1.0 size.
4. NC0.5, NC1.0, and NC2.5 all show very high positive correlations with each other. This indicates that they are likely measuring related or similar aspects of the air quality.

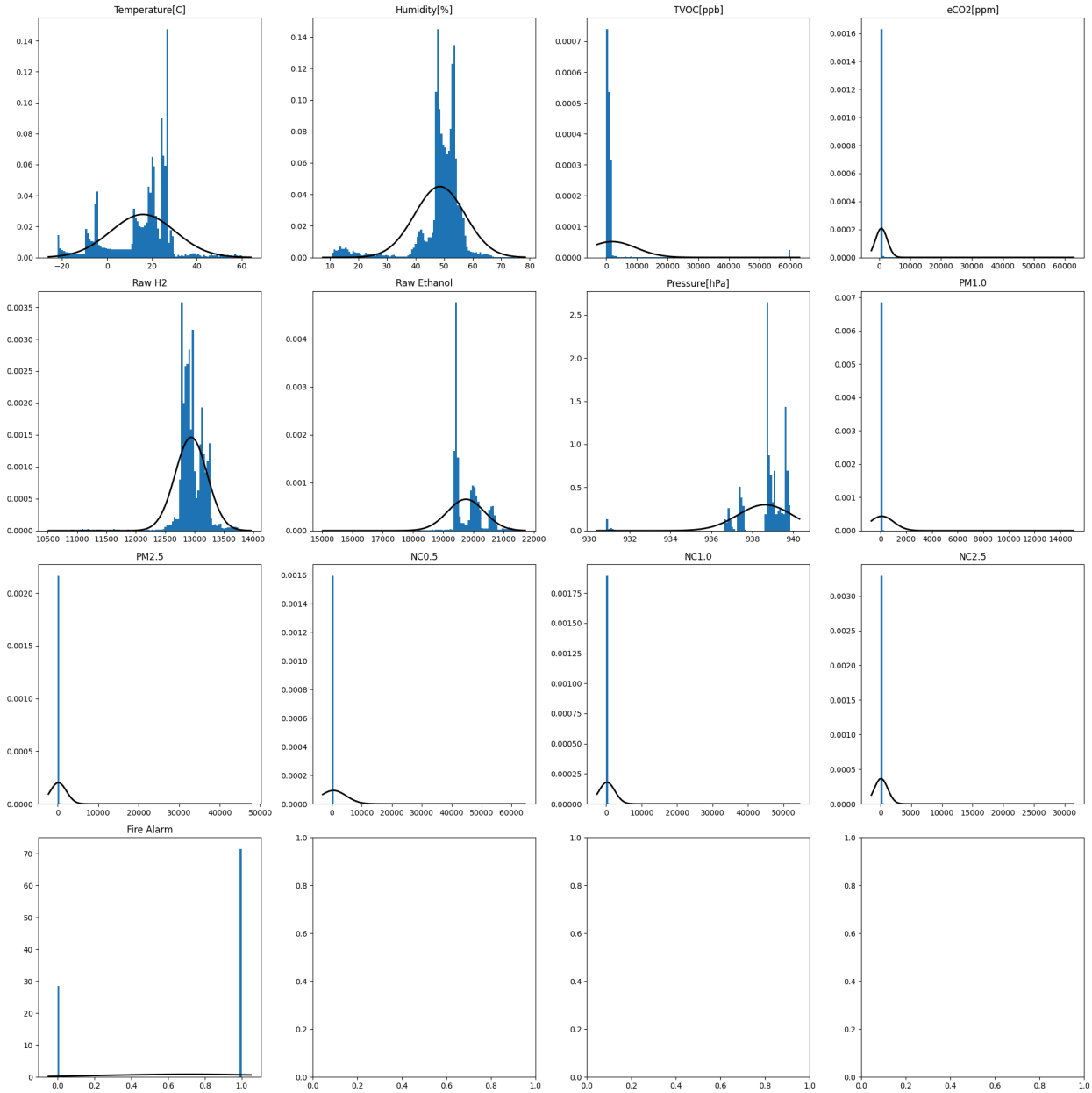
2.5 HeatMap



2.6 PairPlot


Temperature[C]
Humidity[%]
TVOC[ppb]
eCO2[ppm]
Raw H2
Raw Ethanol
Pressure[hPa]
PM1.0
PM2.5
NC0.5
NC1.0
NC2.5
Fire Alarm

2.8 Data Distribution Histograms



3. Modeling Results and Discussion

In this section, we will first create 8 different models and assess their performance. Then we will apply multiple different techniques to improve each model's performance and observe the results.

3.1 Logistic Regression

Logistic Regression predicts the probability that a given input belongs to a specific class using the logistic, or the sigmoid, function. This ensures the output ranges between 0 and 1. In this section we will use logistic regression to predict the fire alarm behavior, analyze how various key components influence the performance of the model, and which set of key components result in the best performance. The key components that will be modified, calibrated and explored are split into two categories:

1. **Data Preprocessing Techniques**
    a. *Normalization*

        This technique aims to scale the data such that the data has a zero mean and unit variance, which helps to speed up convergence during training.

    b. *PCA (Principal Component Analysis)*

        PCA reduces the dimensionality of the data by creating new, uncorrelated variables, often called principal components. This process of eliminating dimensions and creating new variables simplifies the model and reduces multicollinearity.

2. **Model Regularization**

    Regularization is a technique used to prevent overfitting by discouraging overly complex models. It implements this by adding a penalty to the loss function used to train the model. Common types of regularization are:

    a. *Regularization Type*

        i. L1 (Lasso)

This option adds a penalty based on the absolute value of the magnitude of coefficients. This can lead to some coefficients being zero, which means it inherently conducts feature selection.

ii. L2 (Ridge)

This option adds a penalty equal to the square of the magnitude of coefficients. This does not reduce the coefficients to zero, but rather, distributes the error among them. This is better in cases where many features are correlated.

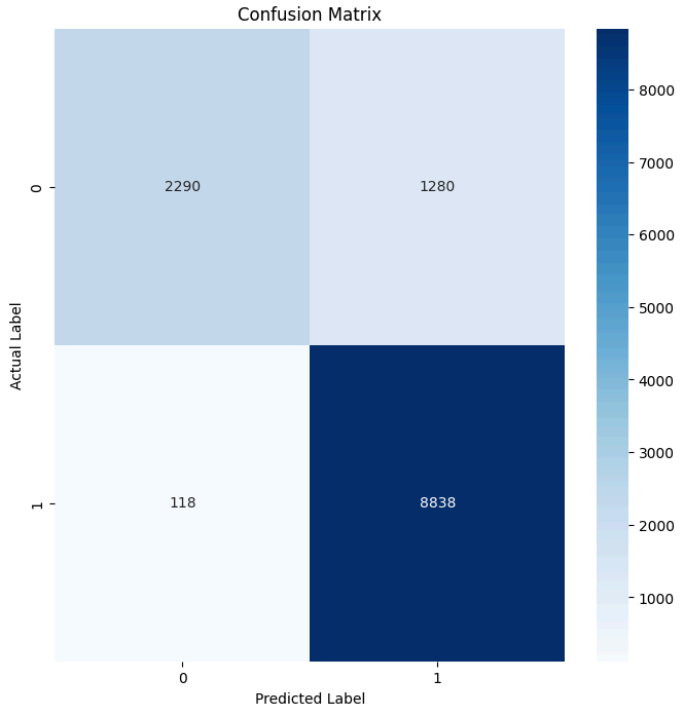
b. *Regularization Strength*

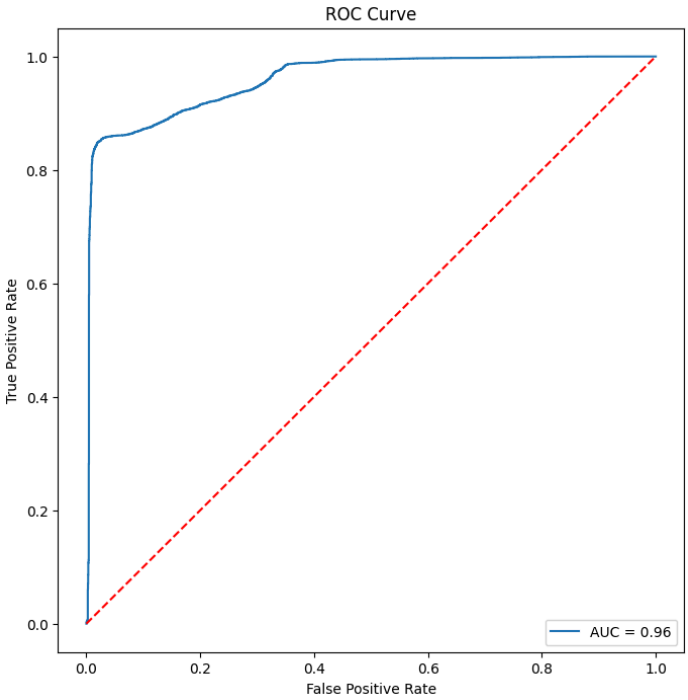
This parameter, often denoted by C or λ in logistic regression, inversely controls the amount of regularization applied: A smaller λ value leads to more regularization. Regularization strength helps to balance the bias-variance tradeoff. High regularization strength (low λ) increases bias but reduces variance, preventing the model from fitting too closely to the training data.

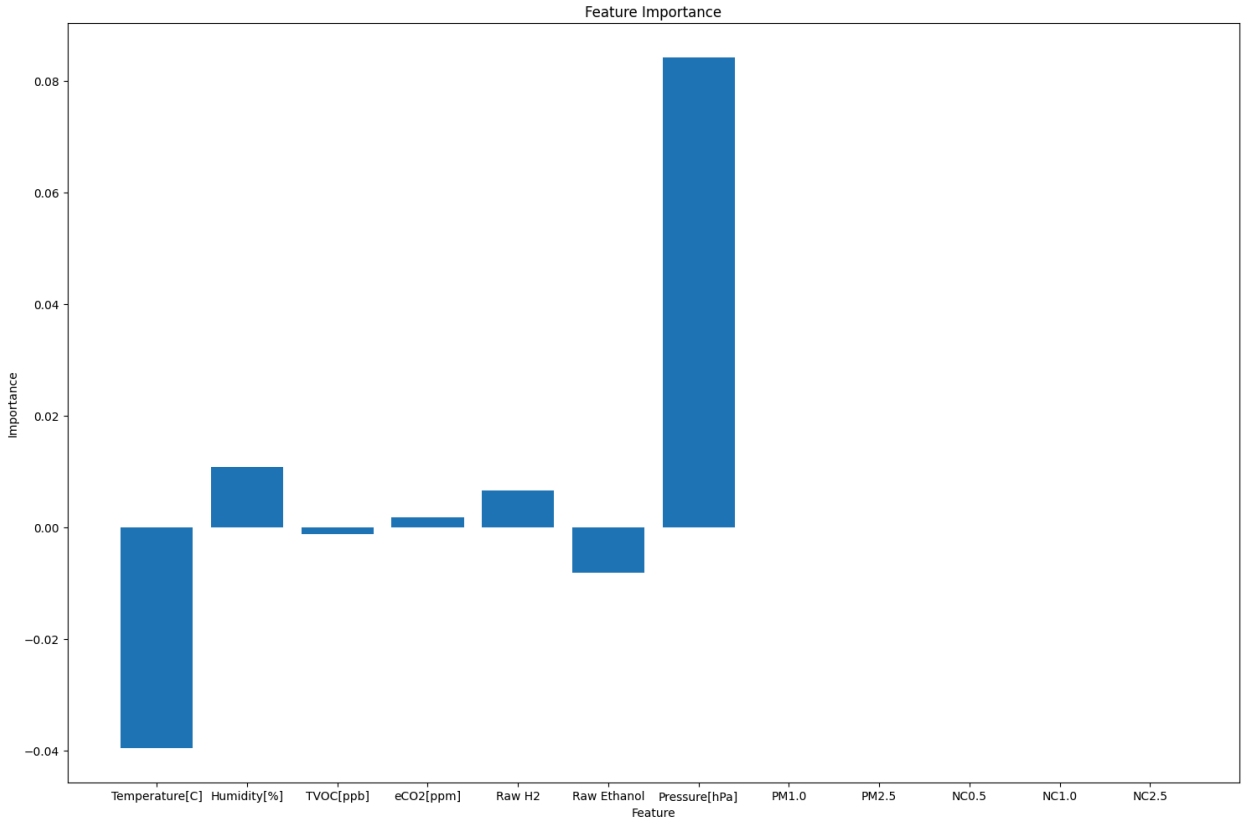
We first start by training a simple Logistic Regression Model with default parameters;

Regularization Type	L2
Regularization Strength (λ)	1.0
Normalization	False
PCA	False

Results


Confusion Matrix
2290
1280
118
8838
Actual Label
Predicted Label
0
1
8000
7000
6000
5000
4000
3000
2000
1000


ROC Curve
True Positive Rate
False Positive Rate
AUC = 0.96
0.0
0.2
0.4
0.6
0.8
1.0


Feature Importance
Importance
Feature
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
Temperature[C]
Humidity[%]
TVOC[ppb]
eCO2[ppm]
Raw H2
Raw Ethanol
Pressure[hPa]
PM1.0
PM2.5
NC0.5
NC1.0
NC2.5

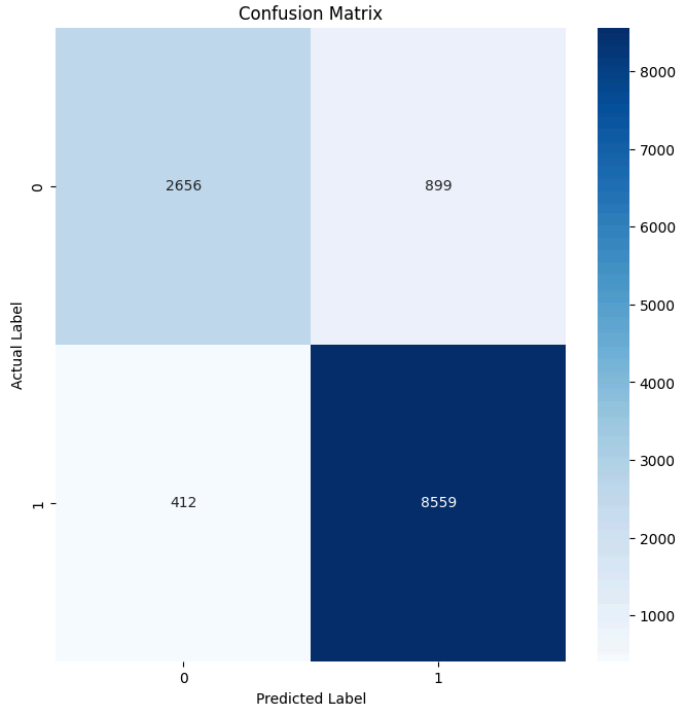
Accuracy Across Entire Dataset	**0.85**	
Metric	**Class 0**	**Class 1**
precision	0.95	0.87
recall	0.64	0.99
f1-score	0.77	0.93
support	3555	8971

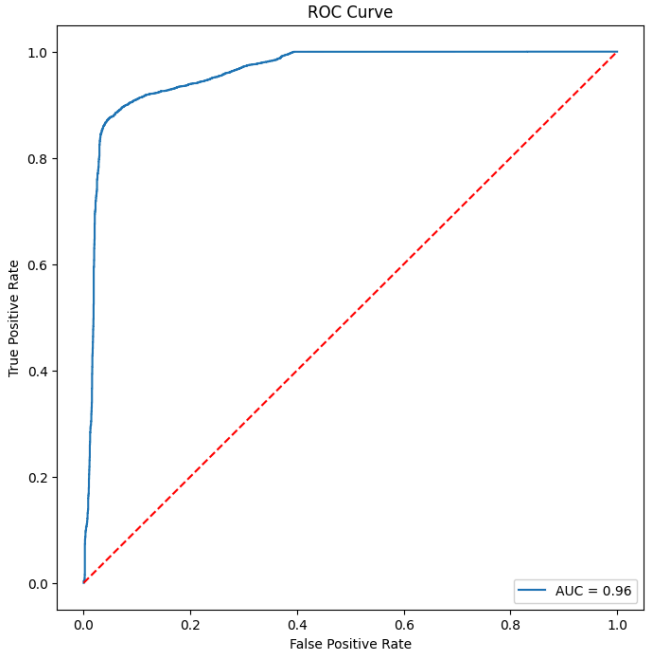
We can observe from the Feature Importance graph that Pressure has the greatest importance and we can also observe that some features have absolutely no importance. This can be due to several reasons, but since the default settings were applied; no PCA or Regularization, the main reason left is that the logistic Regression Model could have determined that some features have little or no predictive power. We can also observe from the Metrics Table, that the logistic regression model with default settings, performs rather well, which is indeed surprising considering no regularization and no PCA. The next step is to find the optimal set of parameters which result in the best performance. The technique employed to find the optimal set of parameters utilized was hyperparameter tuning, and the specific tool used is the Scikit-Learn 'GridSearchCV'. The results are presented below:

Best Parameters:

Regularization Type	L2
Regularization Strength (λ)	1.0
Normalization	True
PCA	False

Here are the results:





Accuracy Across Entire Dataset	**0.90**	
Metric	**Class 0**	**Class 1**
precision	0.87	0.90
recall	0.75	0.95
f1-score	0.80	0.93
support	3555	8971

One strange observation can be made from looking at the best set of parameters: Even though PCA is set to False, this set performs better than the set with PCA set to True. This at first seemed quite counter-intuitive as the core principle of PCA is to reduce dimensionality, reduce collinearity, and introduce new, non-correlated features. Then the question that bogs the mind is, how can it be possible that not including PCA results in the most optimal performance?

The answer lies in the fact that PCA works by creating new features that are linear combinations of the original features. These principal components are selected to maximize variance, which often means they capture the most significant patterns in the data. However, this transformation can sometimes discard subtle but important details in the data which could be critical for making accurate predictions.

Other than the PCA being false, most other parameter values can be intuitively understood as optimal for the model's performance.

3.2 Decision Tree Classifier

Decision Tree Classifiers work by breaking down a dataset into smaller subsets while at the same time, an associated tree is incrementally developed. The key characteristics in Decision Trees are Decision Nodes, which represent essentially asking a question about the data based on which branch is taken, and Leaf Nodes, which represent the outcome or the decision taken after computing all attributes. In this section, we will use Decision Tree Classifiers to predict the fire alarm behavior, analyze how various key components influence the performance of the model, and which set of key components results in optimal performance. The key components that will be modified, calibrated and explored are split into two categories:

1. **Data Preprocessing Techniques**
    a. *Normalization*

        This technique aims to scale the data such that the data has a zero mean and unit variance, which helps to speed up convergence during training.

    b. *Recursive Feature Elimination (RFE)*

RFE works by recursively considering smaller and smaller sets of features. It starts by training a model, pruning the least important features, and then retraining. This process continues until a specified number of features remain.

2. **Hyperparameter Tuning**
    a. *Criterion*

    This determines how the tree decides to split at each node. Choosing between 'gini' and 'entropy' affects how each split is evaluated in terms of the impurity reduction.

    b. *Max Depth*

    This limits the depth of the tree. A deeper tree can capture more detailed data patterns which is good for accuracy but may lead to overfitting. Conversely, a shallow tree might underfit but will generalize better.

3.3 Random Forest Classifier

1. **Data Preprocessing Techniques**
    a. *Normalization*

    Normalization is used to scale the features so that the mean is zero and the standard deviation is one. This ensures that all features contribute equally to the model while also preventing bigger-scale features from dominating the learning process.

    b. *Train-Test Split:*

    The train_test_split method from sklearn.model_selection divides the dataset into training and testing sets. This enables the model to be trained on one set of data and evaluated on another, yielding an estimate of its performance on previously unseen data.

2. **Hyperparameter Tuning**

   a. *Number of Estimators (n_estimators)*

   The number of trees in the forest (n_estimators) affects the model's complexity and generalizability. By adjusting this value, we hoped to achieve the best balance of bias and variance.

   b. *Max Depth*

   The maximum depth of each tree (max_depth) determines the depth of its branches. A deeper tree can capture more complicated relationships in data, but it may result in overfitting. We experimented with several max_depth settings to find the best depth that minimizes overfitting while maximizing model performance.

   c. *Min Samples Split*

   The minimum number of samples needed to separate an internal node (min_samples_split) controls the tree's growth. Setting this parameter higher may prevent the model from splitting nodes prematurely, potentially minimizing overfitting.

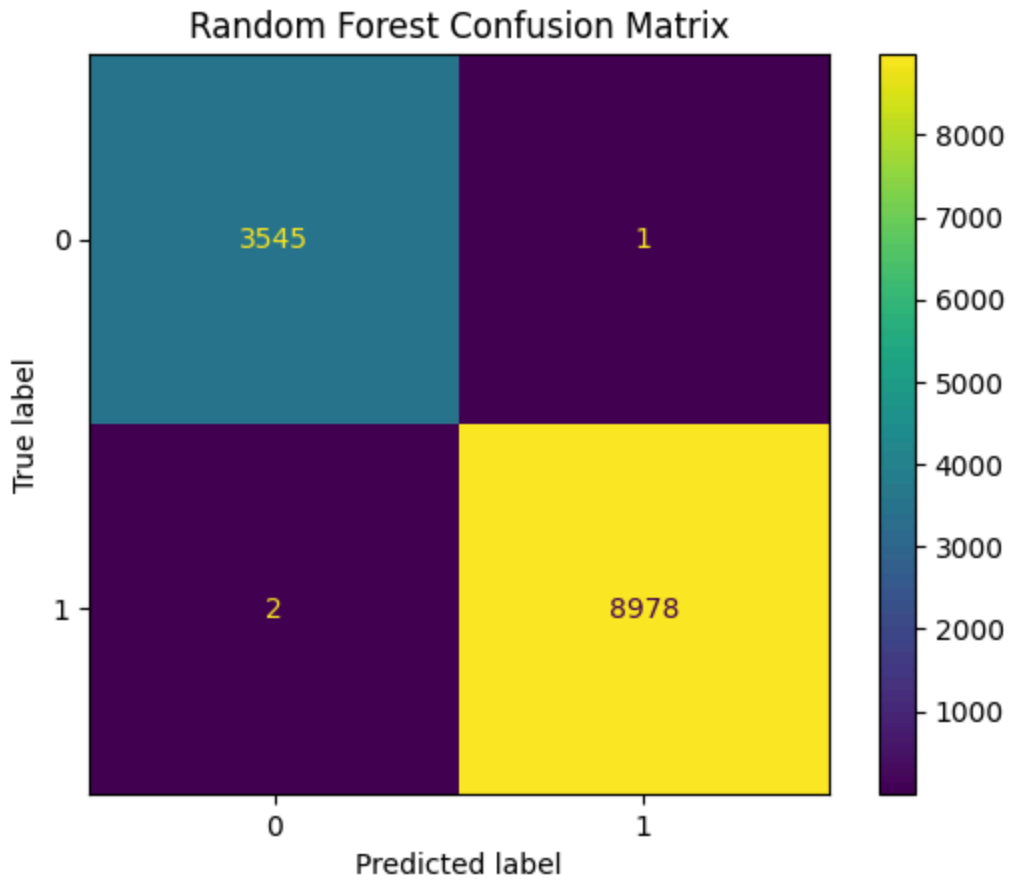
   d. *Min Samples Leaf*

   The minimal number of samples needed at a leaf node (min_samples_leaf) affects the minimum size of leaf nodes. Increasing this parameter prevents the model from forming nodes with too few samples, which reduces overfitting.

Results

We present the results of our Random Forest Classifier, which we developed to predict the behavior of fire alarms based on sensor data. Here's a detailed explanation of our findings:

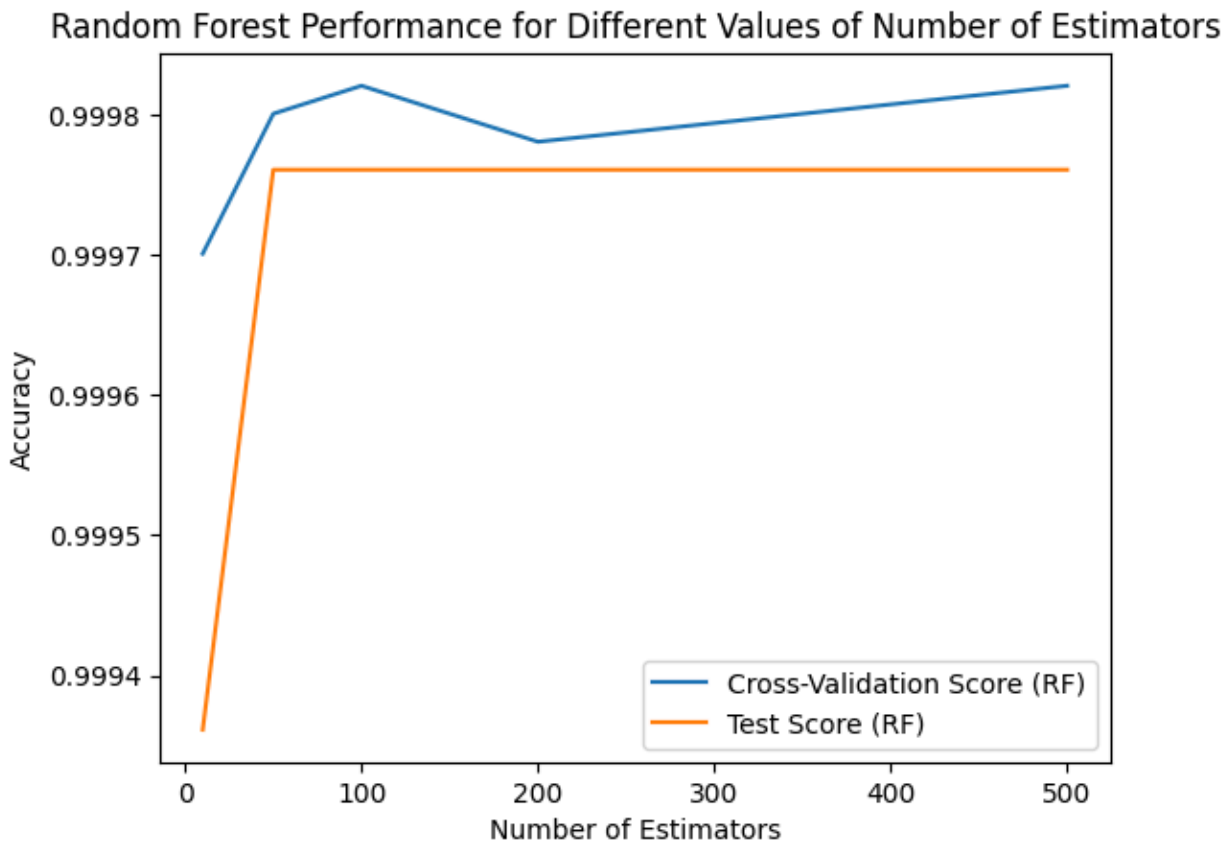
Confusion Matrix (Random Forest):



The confusion matrix provides a comprehensive breakdown of our classifier's predictions compared to the actual labels in the test set. It reveals the following:

- True Negatives (TN): 3545
- False Positives (FP): 1
- False Negatives (FN): 2
- True Positives (TP): 8978

This breakdown allows us to evaluate the classifier's performance in discriminating between fire alarm activations (class 1) and non-alarm activations (class 0). The high number of true positives and true negatives indicates that our classifier is capable of effectively classifying instances into their appropriate categories.



Best Test Score:

Our Random Forest Classifier achieved an outstanding test score of 99.98%. This accuracy score indicates the overall performance of our model on the test dataset. It reflects the classifier's ability to generalize well to unseen data and make accurate predictions.

Best Hyperparameters:

During hyperparameter tuning, the best configuration for our Random Forest Classifier was identified as follows:

- Max Depth: None (unlimited)
- Min Samples Split: 2
- Min Samples Leaf: 1

These hyperparameters were found to considerably improve the resilience and accuracy of our classifier. Allowing trees to grow without depth restrictions while imposing minimum sample

requirements for node splitting and leaf generation improved the performance of our Random Forest model.

Overall, the results show that our Random Forest Classifier can accurately predict fire alarm behavior based on sensor data. The excellent accuracy score and appropriate hyperparameter configuration demonstrate the predictability and efficacy of our model.

3.4 K-Nearest Neighbors Classifier

The K-Nearest Neighbours (KNN) algorithm is a basic yet successful classification approach that assigns new data points to the majority class of their k nearest neighbors in the feature space. In this section, we'll go over how we implemented the KNN classifier and the preprocessing approaches we used to optimize its performance.

1. **Data Preprocessing Techniques**
    a. *Normalization*

        Scaling data to guarantee consistent feature ranges and avoid feature dominance. This ensures that each characteristic contributes proportionately to the distance calculations in KNN, resulting in more balanced and accurate predictions. In our implementation, we used the Normalizer class from scikit-learn to scale the input features to a common scale.

    b. *Feature Selection*

        Choosing the most informative features to reduce dimensionality and concentrate on relevant data. By removing redundant or unnecessary characteristics, the model becomes less prone to overfitting and can more accurately generalize to new data. In our implementation, we used the SelectKBest method along with the f_classif score function to select the top k features that are most strongly related to the target variable. This helps to reduce noise and extraneous information, resulting in improved classification performance.

    c. *Cross Validation*

        Evaluating model performance via k-fold cross-validation ensures robustness and generalizability. By separating the data into subgroups and training the model on

different combinations of these subsets, we can obtain more reliable estimates of the model's performance, which aids in the detection and mitigation of potential overfitting issues. In our implementation, we evaluated the KNN classifier's performance using k-fold cross-validation with k=5. This helps to analyze the model's performance on unknown data and provides information about its resilience and stability.

2. **Hyperparameter Tuning**

   a. *Number of Neighbors (K)*

   This option specifies the number of nearest neighbors to consider when making predictions. We experimented with several values of K to achieve the best balance of bias and variance in our model.

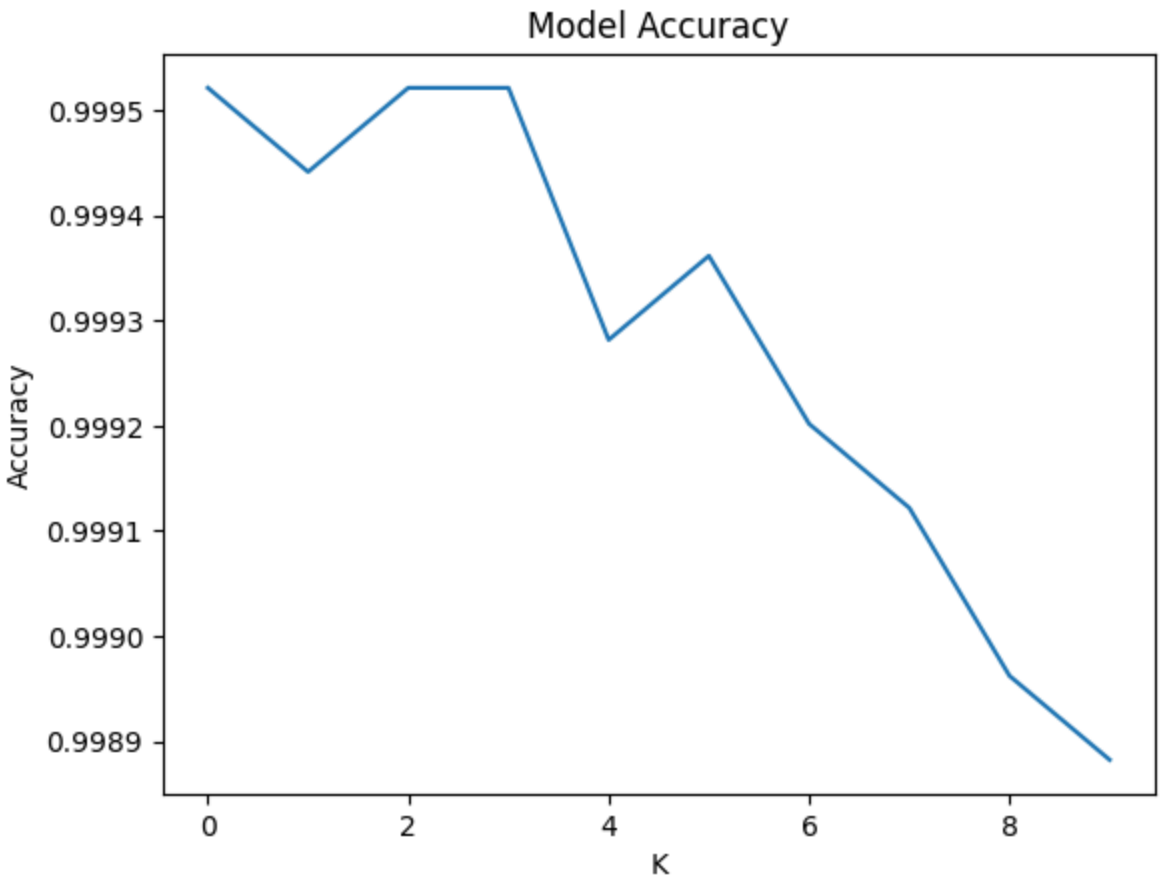
   b. *Distance Metric*

   The distance metric used (for example, Euclidean distance or Manhattan distance) determines how the algorithm analyzes data point similarity. We tested with various distance metrics to find the best one for our dataset.

Results

Analysis of KNN Performance with Varying Values of K:

```
K value:  1 Accuracy:  0.9995209963276385
K value:  2 Accuracy:  0.9994411623822449
K value:  3 Accuracy:  0.9995209963276385
K value:  4 Accuracy:  0.9995209963276385
K value:  5 Accuracy:  0.9992814944914578
K value:  6 Accuracy:  0.9993613284368513
K value:  7 Accuracy:  0.9992016605460642
K value:  8 Accuracy:  0.9991218266006706
K value:  9 Accuracy:  0.9989621587098835
K value:  10 Accuracy:  0.9988823247644899
```



To better understand the influence of K (number of neighbors) on classification accuracy, we tested the performance of our K-Nearest Neighbours (KNN) model with various K values.
We calculated and observed the related accuracies when we changed the value of K from 1 to 10. The results show that accuracy varies modestly with different K values. The best accuracy of around 99.95% was achieved when K was adjusted to 1. As K grows, the accuracy remains reasonably high with slight oscillations, eventually stabilizing at 99.89% when K hits 10.

This analysis emphasizes the significance of choosing an acceptable value for K in KNN modeling. A lower K value may result in more flexible decision boundaries and, therefore, improved training data accuracy, but it may also increase the risk of overfitting. In contrast, a higher K value may result in smoother decision boundaries, perhaps leading to improved

generalization but at the risk of underfitting. As a result, choosing the best value for K requires balancing model complexity with generalization performance.

We then tested the accuracy of our model on the training and testing set. These were the results:

```
Accuracy on training set: 1.0
Accuracy on testing set: 0.9995209963276385
```

Despite reaching a flawless accuracy of 100% on the training set, our K-Nearest Neighbours (KNN) model had a little lower accuracy of around 99.95% on the testing set. This disparity in training and testing accuracy raises the possibility of overfitting, in which the model performs extraordinarily well on training data but struggles to generalize to new data.

To reduce the risk of overfitting and get a more accurate assessment of our model's performance, we used k-fold cross-validation. By splitting the training data into various subsets and iteratively training the model on different combinations of these subsets, we were able to achieve more trustworthy performance estimates. Cross-validation helps to test our model's resilience and generalizability, revealing its true performance on previously unseen data.

In our implementation, we used 5-fold cross-validation, which involves dividing the training data into five equal parts and training the model five times, each with a different half serving as the validation set. By averaging the performance indicators throughout several rounds, we received a more accurate evaluation of the model's ability to categorize unseen data

Classification Report after Cross Validation:

After 5-fold cross-validation, the KNN model achieved high accuracy ratings across all folds, ranging from 99.88% to 99.97%. The mean cross-validation accuracy, computed as 99.92%, reflects the model's overall performance on the training data.

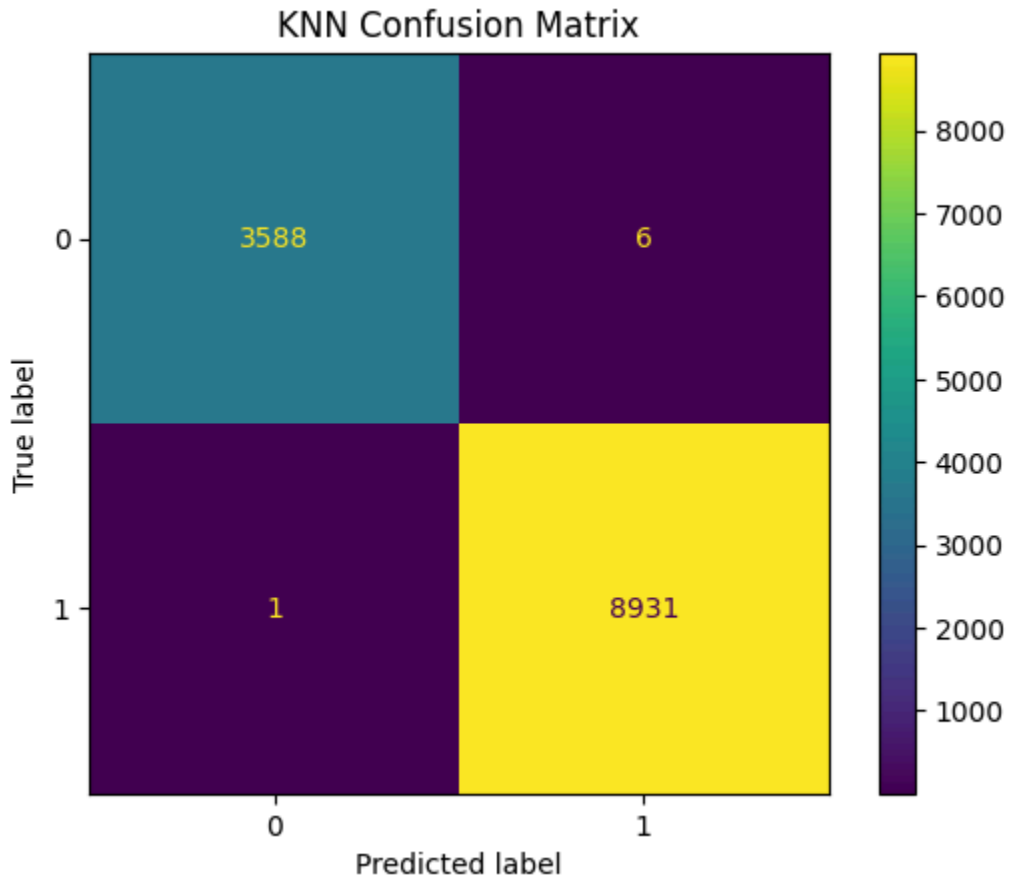
After analyzing the model on the testing set, the KNN classifier attained an accuracy of 99.94%, demonstrating its strength and generality. The classification report also shows the model's exceptional performance, with precision, recall, and F1-score all at 100% for both classes (0 and 1). This shows that the model correctly classified instances of both alarm statuses.

```
Cross-Validation Scores: [0.99920168 0.99880251 0.9990021  0.99970063 0.9993014 ]
Mean CV Accuracy: 0.9992016625366172
Accuracy on Testing Set: 0.9994411623822449
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3594
           1       1.00      1.00      1.00      8932

    accuracy                           1.00     12526
   macro avg       1.00      1.00      1.00     12526
weighted avg       1.00      1.00      1.00     12526
```

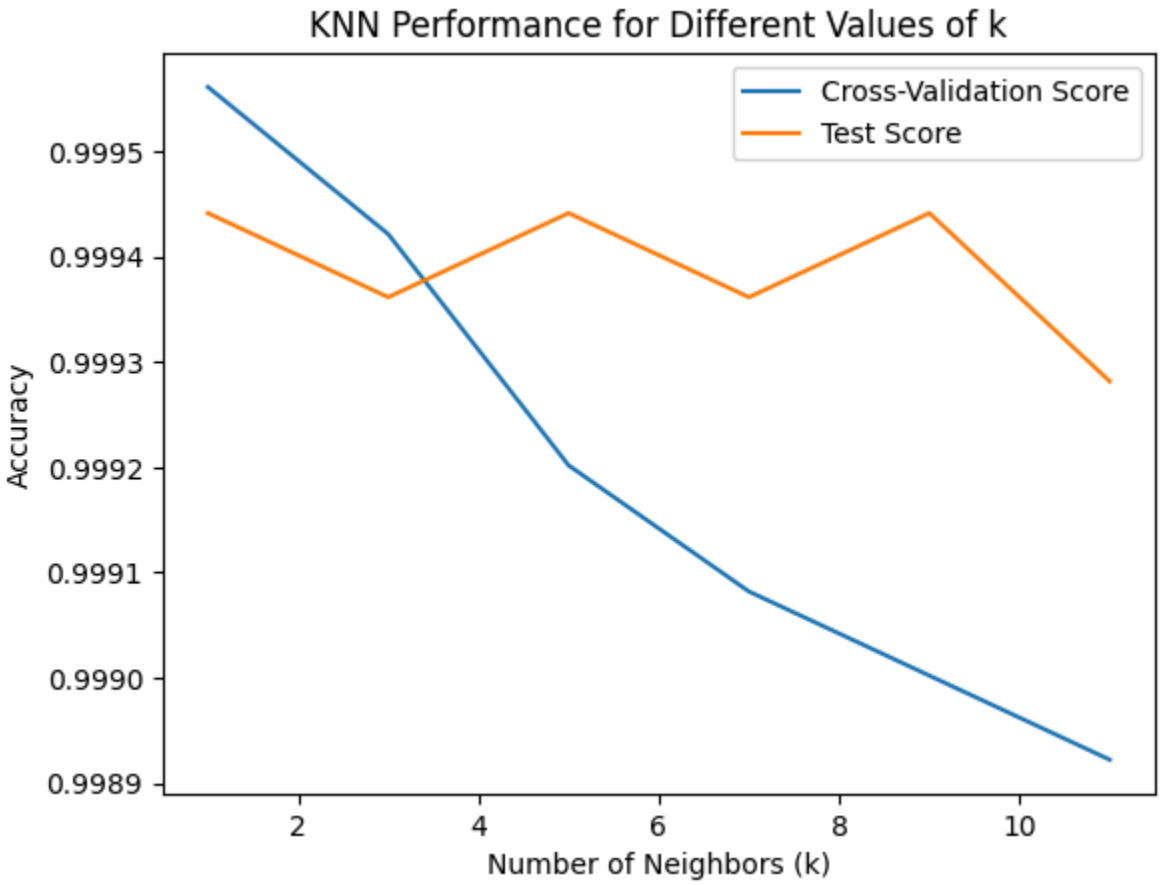
Confusion Matrix after Cross Validation:



The confusion matrix shows that out of 3594 instances of class 0 (no fire alarm), the model accurately predicted 3588 cases, with only 6 false positives. Similarly, out of 8932 cases of class

1 (fire alarm), the model accurately predicted 8931, with only one false negative. These findings show that the KNN model is effective at discriminating between alarm states, with very few misclassifications.

KNN Performance vs. Different Values of K (Number of Neighbors):



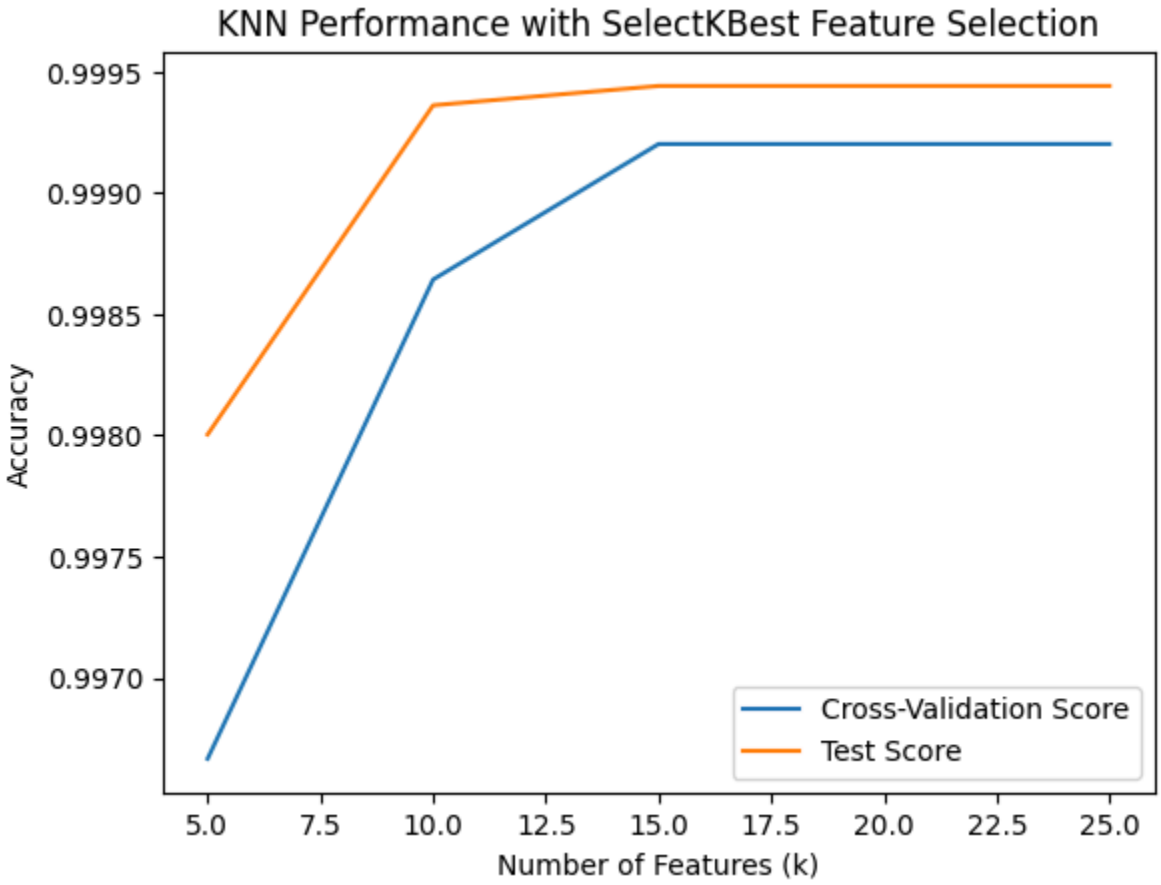
The performance of our K-Nearest Neighbours (KNN) model changes according to the number of neighbors (K) used for classification. Analyzing the link between K values and accuracy shows intriguing patterns.

As the number of neighbors increases, the model's accuracy first declines somewhat, beginning at 0.999 and steadily dropping to around 0.9989. However, beyond this point, accuracy stabilizes and remains good, demonstrating that our KNN classifier performs best with a reasonable number of neighbors.

Surprisingly, while examining test scores, we see swings in accuracy. The test accuracy begins at 0.99947, falls slightly below 0.9994, and then rises back up. This variation indicates that the model's performance on unknown data may vary slightly, but it remains consistently high.

These findings emphasize the importance of choosing a suitable number of neighbors to achieve optimal model performance. They also demonstrate the robustness of our KNN model in reliably identifying fire alarm behavior based on sensor data, even when considering varied numbers of neighbors.

KNN Performance vs. SelectKBest Feature Selection:



Analyzing the performance of our K-Nearest Neighbours (KNN) model in conjunction with SelectKBest feature selection yields useful information about the effect of feature dimensionality on classification accuracy.

When evaluating a small number of features (5), the cross-validation score accuracy initially appears to be worse, sitting just around 0.997. However, as the number of selected features increases, the accuracy gradually improves. This upward trend continues, topping 0.9985 and eventually reaching a stable plateau between 0.9990 and 0.9995, showing that the selected features are effective at gathering meaningful information for categorization.

In contrast, the test score accuracy begins at a higher value of 0.998 and gradually increases with the inclusion of more features. The accuracy gradually increases to slightly around 0.9995 before plateauing as the number of features grows. This trend indicates that, while increasing the amount of features initially improves the model's predictive performance, there comes a point at which additional characteristics may not help much to boost accuracy.

Overall, the results show that feature selection is critical for optimizing the performance of our KNN model, with both cross-validation and test scores emphasizing the advantages of picking an adequate subset of features for classification tasks.

3.5 Support Vector Machine Classifier

Support vector machines create high-dimensional hyperplanes that maximize the distance between the plane and the nearest data points on either side of the plane. In this section, we will cover creating a support vector classifier to predict the alarm's behavior and tuning the model's hyperparameters to get the best results.

1. **Data Preprocessing Techniques**
    a. *Normalization*

        This technique aims to scale the data such that the data has a zero mean and unit variance. This is important for support vector machines as they are sensitive to distances between data points in different directions.
2. **Hyperparameter Tuning**
    a. *C: Margin size*

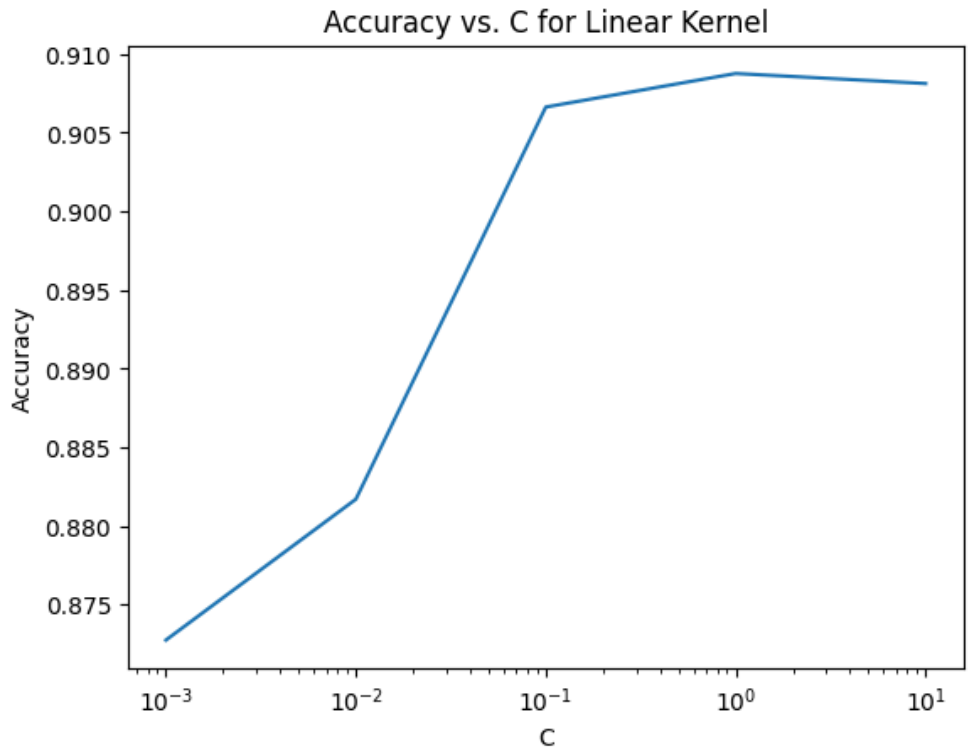
        Larger margin sizes bring the model closer to a hard margin model. This is an important factor in balancing between under and overfitting.
    b. *Kernel*

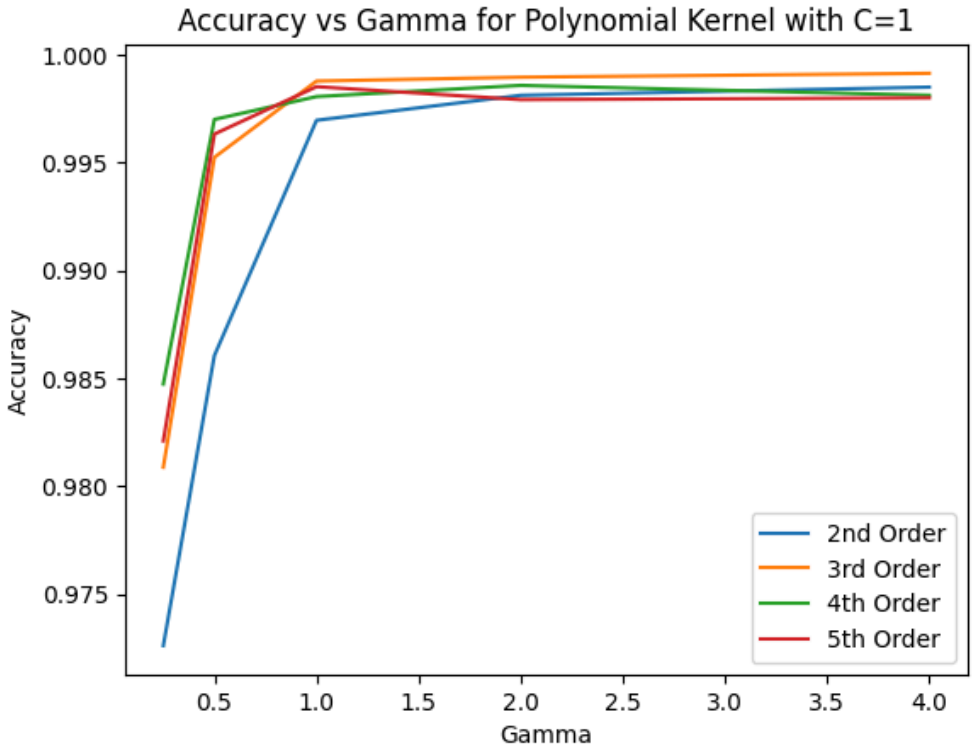
        The kernel of the model applies a pre-transformation of the features to allow for non-linear classification. We will investigate several different kernels including a radial basis function kernel, a linear kernel, several polynomial kernels of different orders, and a sigmoid kernel.
    c. *Gamma: Kernel Coefficient*

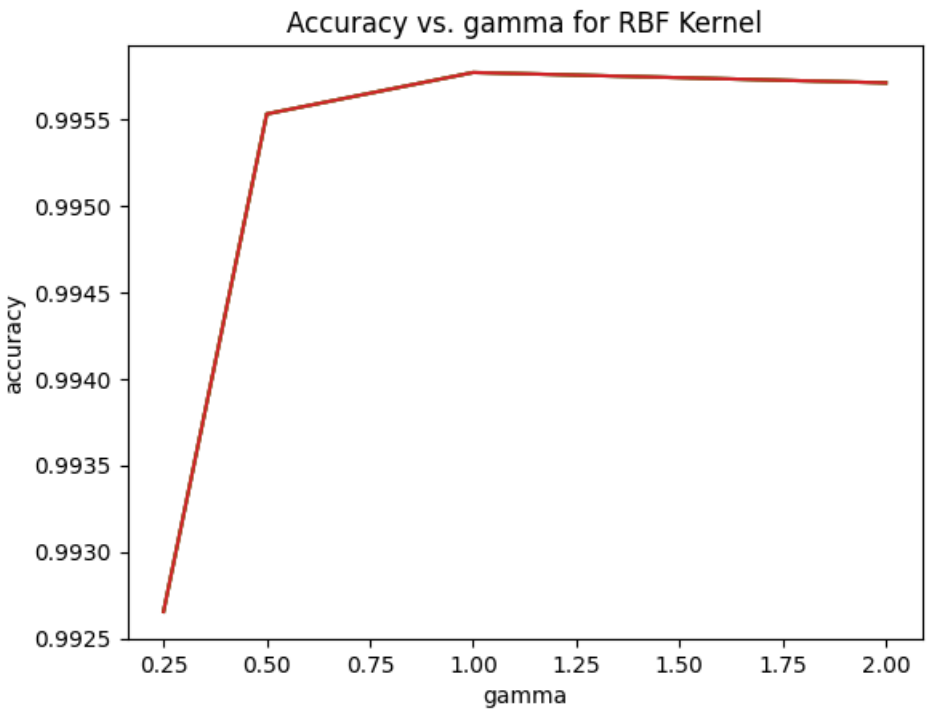
        The radial basis function, polynomial, and sigmoid kernels have coefficients that need to be tuned as they modify how the features are transformed.

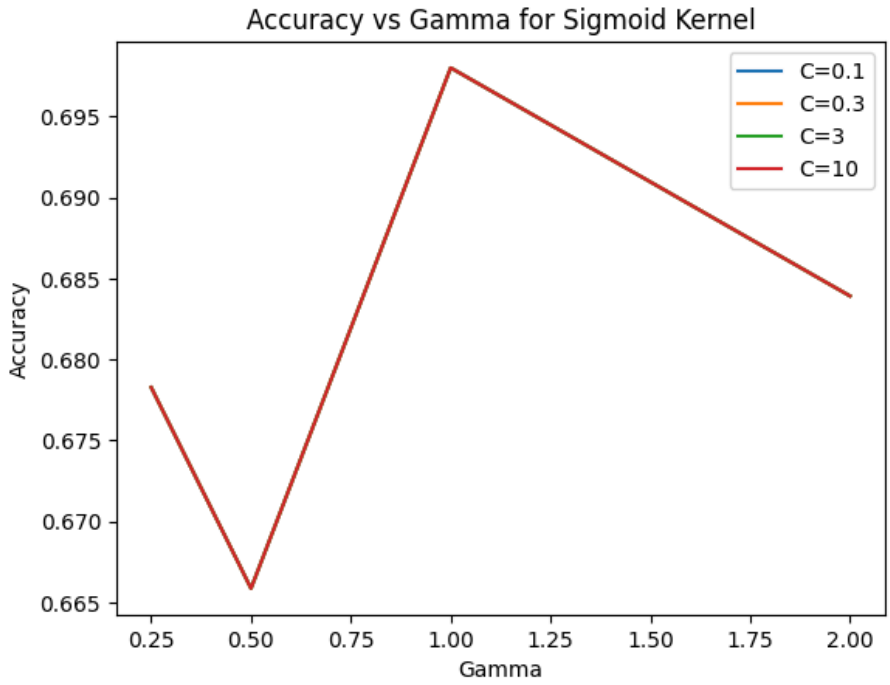
Results

From initial testing, we can tell that C has little effect on any of these models except for if it has a linear kernel. Because of this, we will drop it as a variable parameter.


Accuracy vs. C for Linear Kernel
Accuracy
C
0.910
0.905
0.900
0.895
0.890
0.885
0.880
0.875
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}


Accuracy vs Gamma for Polynomial Kernel with C=1
Accuracy
Gamma
1.000
0.995
0.990
0.985
0.980
0.975
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
2nd Order
3rd Order
4th Order
5th Order





From these graphs, we can see that we should use either a 3rd Order kernel or a RBF kernel. They have a maximum accuracy of

3.6 Multi-Layer Perceptron

A Multi-Layer perceptron uses a dot product of the input vector and a weight vector of equal length plus a bias and then fed into a nonlinear function several times in a row. The weights and bias can be tuned through back-propagation and this can be used to classify through a final sigmoid function.

3. **Data Preprocessing Techniques**
    a. *Normalization*

        This technique aims to scale the data such that the data has a zero mean and unit variance. This is important for support vector machines as they are sensitive to distances between data points in different directions.

4. **Hyperparameter Tuning**
    a. *Number of layers*

        More layers can reduce underfitting as it allows for the approximation of more complex functions. However, we also need to be careful of overfitting.

    b. *Number of nodes per layer*

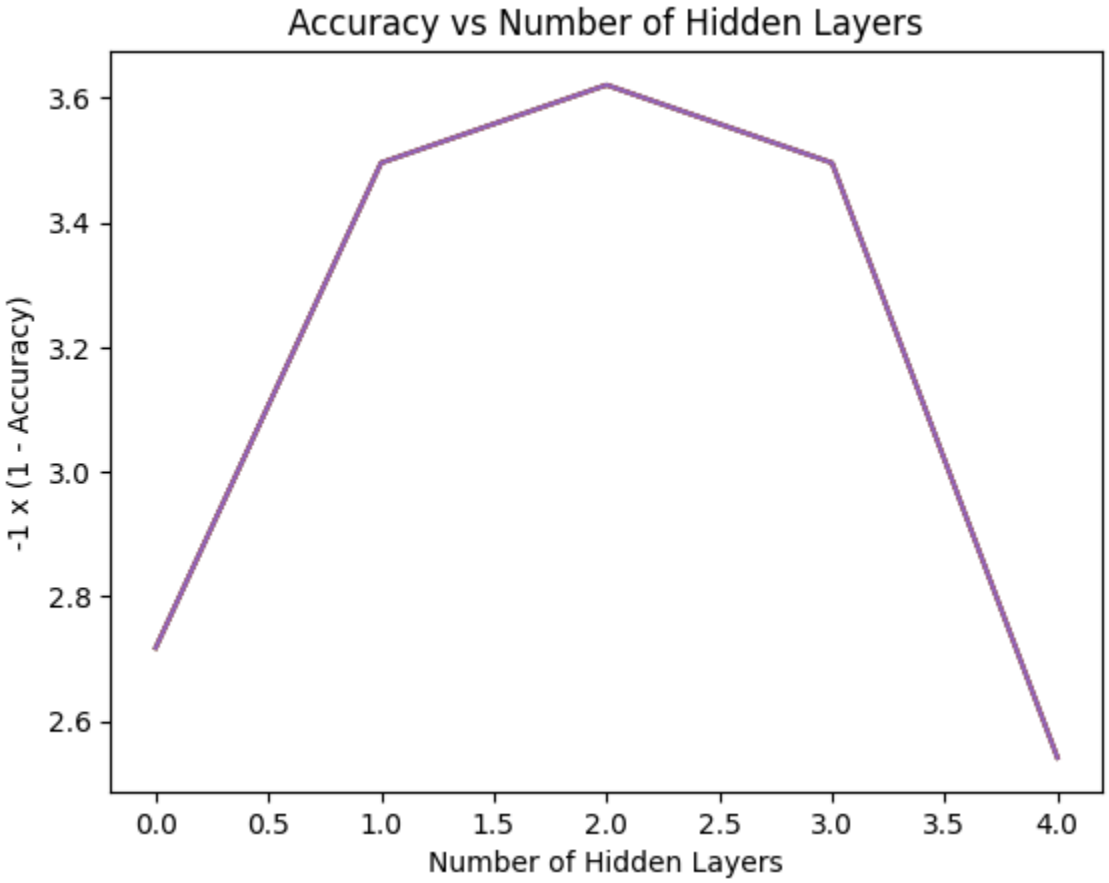
        Each internal layer has a set number of nodes and can be increased or reduced in increasing or decreasing model complexity.

    c. *Activation Function*

        The activation function of the internal layers doesn't have a predictable effect on the model, but different functions may work better than others. Ideally, we would use ReLU for its low complexity.

Results

The MLP model seems very accurate. I wasn't able to see variances when using a regular plot so I used the function y = -1*log(1-x) to plot accuracy.



As you can see the best option had a plot value of just over 3.6 which corresponds to an accuracy of 0.99975. There are several different lines on the plot above which correspond to different numbers of nodes in each layer, with a minimum of 5. Because there is almost no difference between these values we will use a network with 2 hidden layers and 5 nodes per layer.

Conclusion

Following a thorough study and comparison of several machine learning models for predicting smoke detector behavior, each model revealed distinct strengths. However, a few models stood out as top performances.

The Random Forest Classifier maintained its supremacy, with an impressive test score of 99.98%. Its resistance to overfitting and consistent performance across multiple hyperparameter combinations cemented its status as the best-performing model.

Following closely, the Multi-Layer Perceptron (MLP) demonstrated amazing accuracy, scoring 99.976% on the testing set. This performance establishes MLP as a formidable competitor alongside Random Forest.

Furthermore, the Support Vector Machine Classifier (SVC) outperformed other models in terms of testing set correctness, with a score of 99.55%.

While the K-Nearest Neighbours (KNN) method was previously popular, the new results show that it is somewhat less accurate than MLP and SVC on the testing set.

Other models, such as Logistic Regression and Decision Tree Classifier, performed well, but did not match Random Forest, MLP, and SVC in terms of predictive accuracy and generalization.

In conclusion, the Random Forest Classifier is still the best choice for smoke detector prediction jobs due to its remarkable accuracy and durability. However, both MLP and SVC have emerged as formidable challengers, delivering competitive results. The most appropriate model should be chosen after taking into account unique application needs, computational resources, and interpretability factors.

References

1. Baum, A. (n.d.). Binary Classification of Fire Alarm Data. Retrieved from Kaggle: https://www.kaggle.com/code/andrewbaum/binary-classification-of-fire-alarm-data
2. scikit-learn. (n.d.). Retrieved from https://scikit-learn.org/
3. MSE 413: Lab Manual 1
4. MSE 413: Lab Manual 2
5. MSE 413: Lab Manual 3
6. MSE 413: Lab Manual 4