California State University, Long Beach

Computer Engineering and Computer Science Department

Fire Alarm Detection Model from Smoke Data

Submitted by

Matthew Fehr, Jeffrey Hua

Abstract

This project aims to analyze different machine learning models such as logistic regression, decision trees, SVMs, naive bayes, mlp and random forest to accurately and reliably predict whether a fire is or is not present. The idea is to build accurate models and fuse them together to build an AI powered fire alarm.

The dataset given works with over 60,000 readings measured at one-second intervals in different simulated environments to determine if a fire alarm will be triggered or not. The sensors measure data such as temperature, humidity, TVOC, eCO2, H2, raw ethanol, pressure, and particulate size and concentration amounts. By determining which features hold high importance, training a model may be incredibly valuable as an early prediction tool. The resulting model represents the potential that machine learning may have in early-prediction tools and risk mitigation

Using various techniques such as class balancing, feature selection, data transformation to fix feature skew and hyper parameter tuning, accurate models were able to be created. Logistic regression had an accuracy of 84.33%. Decision tree has an accuracy 99.98%. SVM has an accuracy of 99.98%. And naive bayes have an accuracy of 91.03%.
By fusing these models, a powerful fire detector model was about to be created with an accuracy of 99.98%, which is comparable to deeper and more expensive models such as MLP with an accuracy of 99.96% and random forest with 99.98%.

Introduction

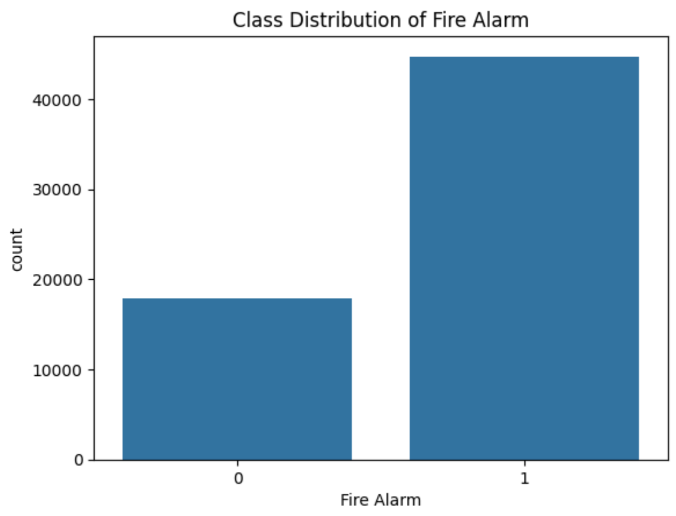
With the threat of California fires being ever so present, the need to develop stronger and better detection systems is vital to the well-being of those in fire-hazard severity zones. The dataset used in the training of this model utilizes Arduino Nicla Sense ME sensors alongside other external sensors in an attempt to accurately demonstrate different conditions a fire may or may not occur[1]. These sensors recorded under various environmental conditions - such as indoor wood, outdoor coal, and outdoor high humidity. The dataset includes a variety of features such as temperature, humidity, TVOC, eCO2, H2, raw ethanol, pressure, and particulate size and concentration amounts[2]. Overall, the dataset contains roughly ~62,000 rows with 14 features recorded over a time period of 4 days at each second interval, which provided an adequate amount to train our baseline model.

To start off, the model utilized the raw, unedited data, with the removal of the counter and tag as they were irrelevant to the overall training and negatively impacted future models. To improve upon this model, multiple techniques were used in the following iterations including class balancing, feature selection to remove low importance and highly correlated features, data transformation to fix skewness, hyperparameter tuning, and model fusion to ultimately achieve the final model presented in this paper below.

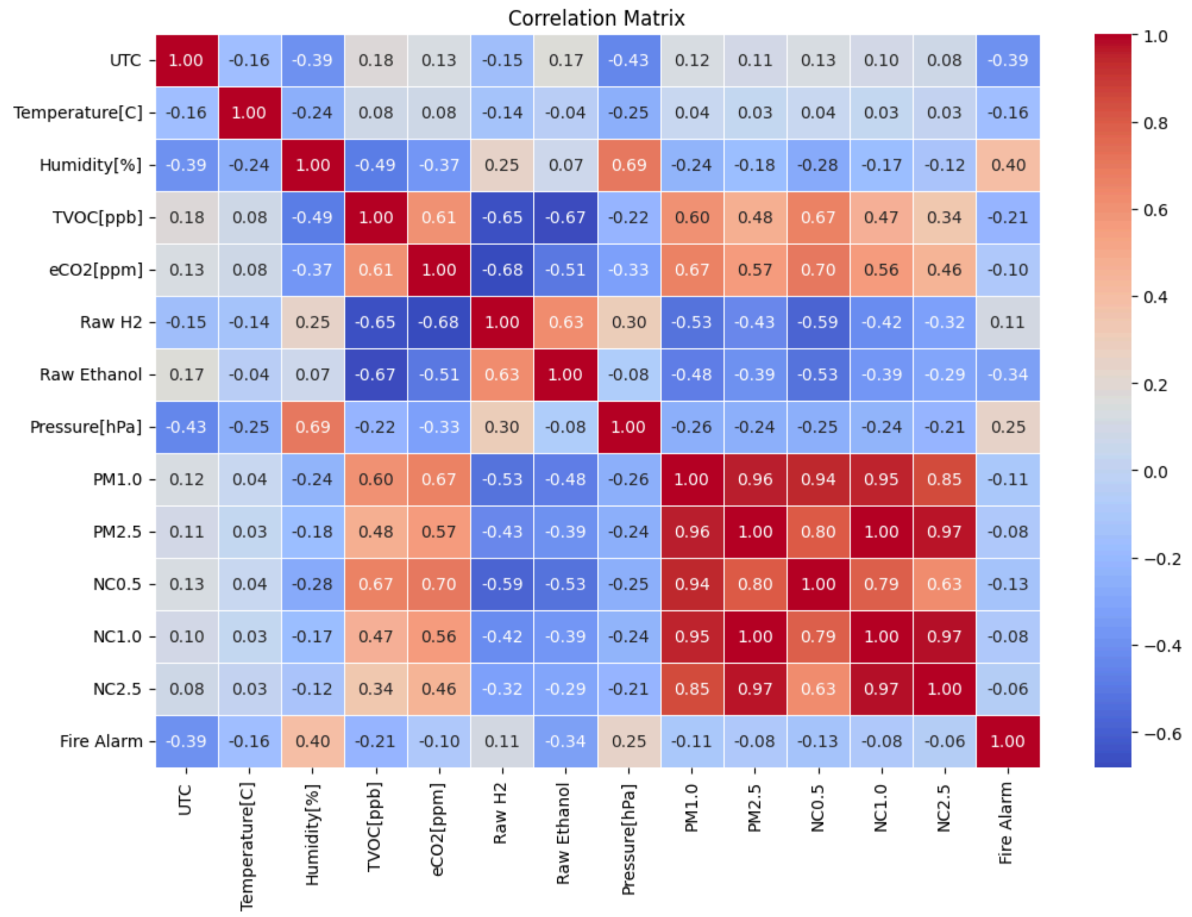
Problem Statement

Develop a model to accurately classify whether a fire is present given many factors like temperature, humidity, pressure, presence of gasses and particulates. Models such as Logistic Regression, Decision Trees, SVM, Naive Bayes, MLP and Random Forest will be used.

Exploratory Data Analysis



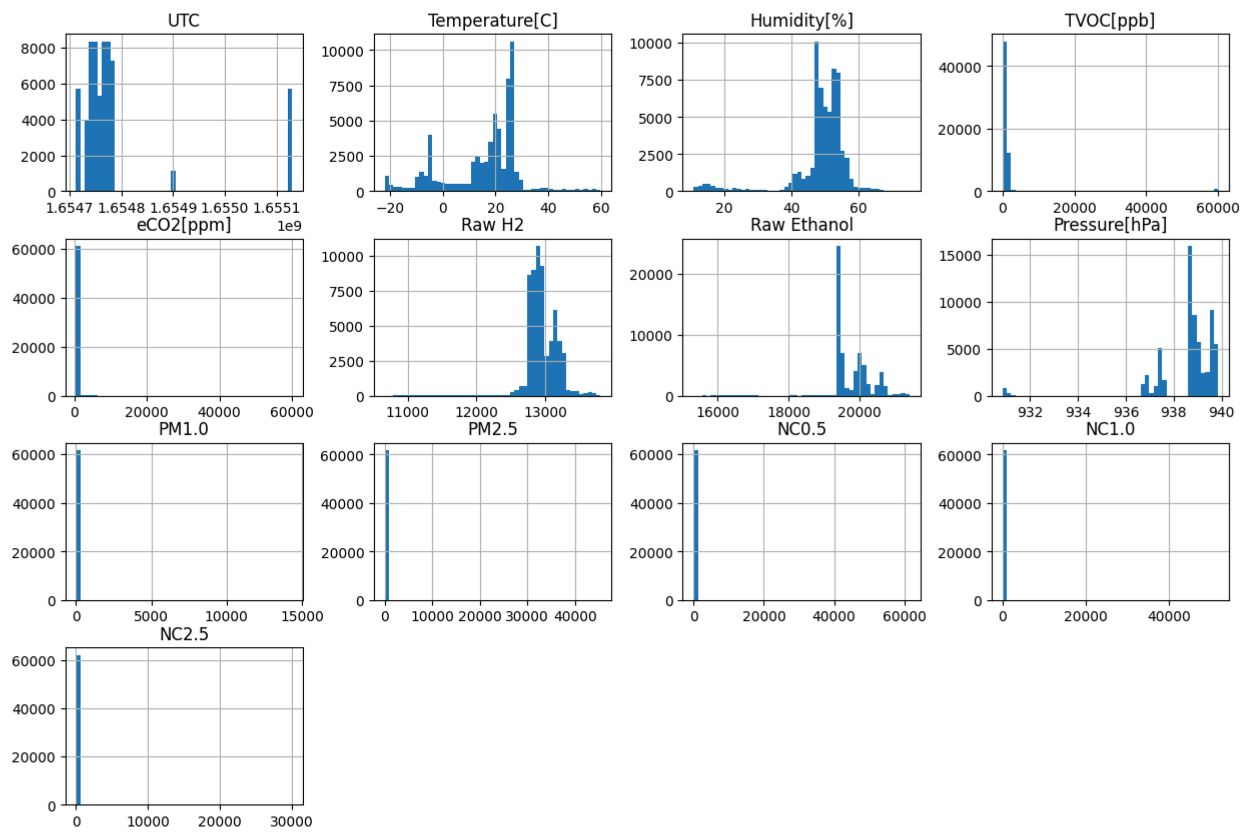
After the initial model was trained, there was a need to balance the dataset as the ratio of fire alarm triggers was imbalanced at roughly ~72% to ~29% with the majority of data indicating that a fire alarm had been triggered.



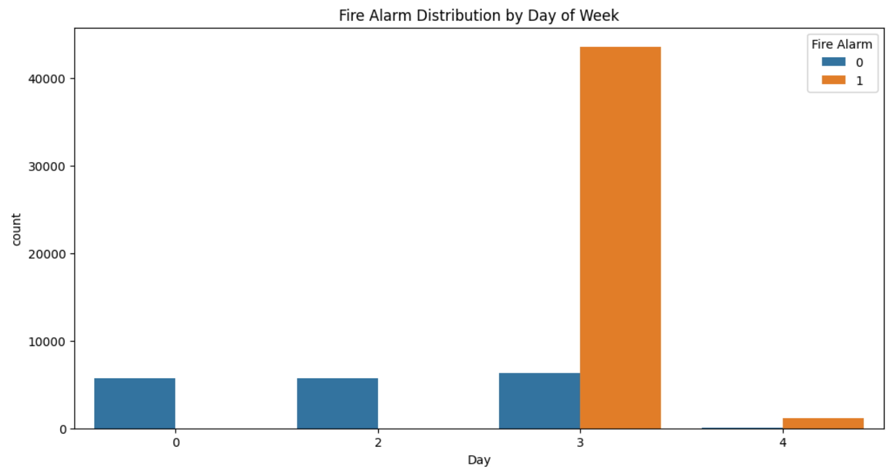
	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
UTC	1.00	-0.16	-0.39	0.18	0.13	-0.15	0.17	-0.43	0.12	0.11	0.13	0.10	0.08	-0.39
Temperature[C]	-0.16	1.00	-0.24	0.08	0.08	-0.14	-0.04	-0.25	0.04	0.03	0.04	0.03	0.03	-0.16
Humidity[%]	-0.39	-0.24	1.00	-0.49	-0.37	0.25	0.07	0.69	-0.24	-0.18	-0.28	-0.17	-0.12	0.40
TVOC[ppb]	0.18	0.08	-0.49	1.00	0.61	-0.65	-0.67	-0.22	0.60	0.48	0.67	0.47	0.34	-0.21
eCO2[ppm]	0.13	0.08	-0.37	0.61	1.00	-0.68	-0.51	-0.33	0.67	0.57	0.70	0.56	0.46	-0.10
Raw H2	-0.15	-0.14	0.25	-0.65	-0.68	1.00	0.63	0.30	-0.53	-0.43	-0.59	-0.42	-0.32	0.11
Raw Ethanol	0.17	-0.04	0.07	-0.67	-0.51	0.63	1.00	-0.08	-0.48	-0.39	-0.53	-0.39	-0.29	-0.34
Pressure[hPa]	-0.43	-0.25	0.69	-0.22	-0.33	0.30	-0.08	1.00	-0.26	-0.24	-0.25	-0.24	-0.21	0.25
PM1.0	0.12	0.04	-0.24	0.60	0.67	-0.53	-0.48	-0.26	1.00	0.96	0.94	0.95	0.85	-0.11
PM2.5	0.11	0.03	-0.18	0.48	0.57	-0.43	-0.39	-0.24	0.96	1.00	0.80	1.00	0.97	-0.08
NC0.5	0.13	0.04	-0.28	0.67	0.70	-0.59	-0.53	-0.25	0.94	0.80	1.00	0.79	0.63	-0.13
NC1.0	0.10	0.03	-0.17	0.47	0.56	-0.42	-0.39	-0.24	0.95	1.00	0.79	1.00	0.97	-0.08
NC2.5	0.08	0.03	-0.12	0.34	0.46	-0.32	-0.29	-0.21	0.85	0.97	0.63	0.97	1.00	-0.06
Fire Alarm	-0.39	-0.16	0.40	-0.21	-0.10	0.11	-0.34	0.25	-0.11	-0.08	-0.13	-0.08	-0.06	1.00

Afterwards, a correlation matrix was created to help identify any highly correlated features that were present. Notice how many of the particulate substances such as PM 1.0, 2.5, and NC 0.5, 1.0, and 2.5 are highly correlated with one another (Greater than absolute value ~ .85). This high correlation makes sense as all are dealing with particulates in the air: either through mass concentration (PMs) or actual count (NCs). The other features like temperature, air pressure, humidity, hydrogen gas, ethanol and less obviously related. Total volatile organic compounds and equivalent CO2 are more related to the highly correlated features than those previously mentioned, but still not above the 0.85 threshold.

Original Feature Distributions



Nearly all features present contain a large skew greater than ±1 as well, with temperature being a notable outlier as the only feature with a relatively symmetric distribution of data points. However, there were no recorded outliers present in this dataset.

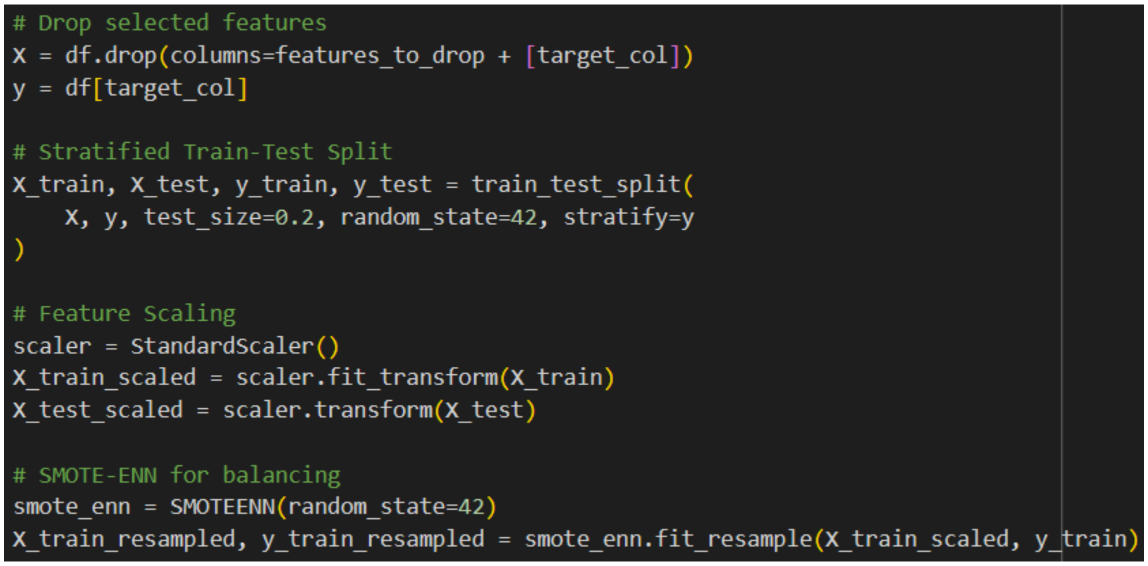


Additionally, the presence of time led to a more in-depth time-based analysis which found that there were no fire alarm triggers on the first 2 days, which may lead the model to inaccurately predict using the UTC time provided.

The decision to drop the UTC feature was eventually made to prevent any unintentional correlation that may occur between those data points as they were not an accurate representation of how fire should be detected. Any future models from here on will not contain any time information.

Balancing the Dataset

Each step to improve the dataset was relatively minor to ensure there were no erratic disparities in our training. To do so, techniques such as stratification, feature scaling, and SMOTE-ENN in that order were used to achieve this goal.



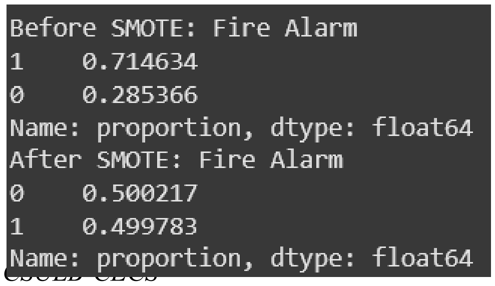
```
# Drop selected features
X = df.drop(columns=features_to_drop + [target_col])
y = df[target_col]

# Stratified Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# SMOTE-ENN for balancing
smote_enn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smote_enn.fit_resample(X_train_scaled, y_train)
```

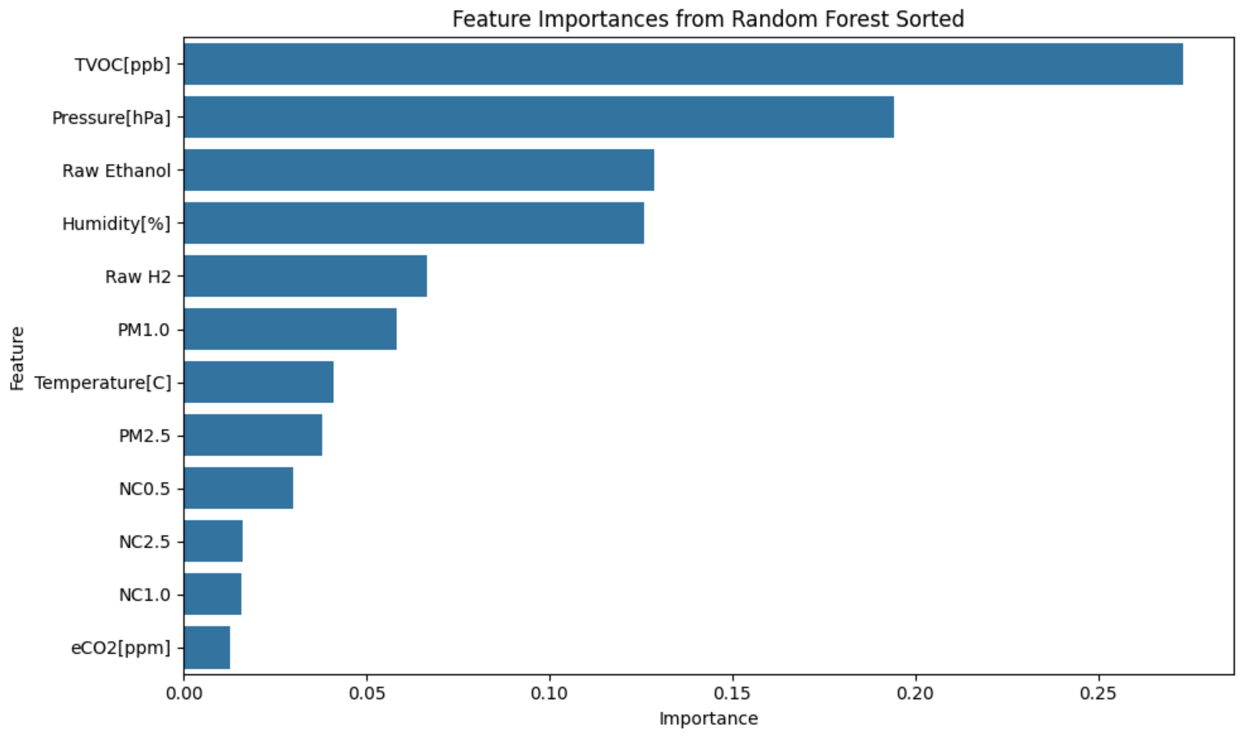
Stratification ensures the train and test sets have the same class distribution present in our original data to avoid any further imbalancing that may occur when splitting the data. Feature scaling standardizes our features with a mean of 0 and a standard deviation of 1. This step is vital for SMOTE-ENN to succeed as it relies on a standardized dataset. SMOTE-ENN is composed of SMOTE (Synthetic Minority Oversampling Technique) and ENN (Edited Nearest Neighbors). SMOTE helps create synthetic examples of our minority class (No fire alarm triggered) between our existing examples. ENN helps to clean our dataset by removing ambiguous or noisy majority class examples by looking at their nearest neighbor to reduce any overlap between classes. These techniques will be present in all future iterations. Using SMOTE-ENN, the dataset is now balanced with a near-even split between the fire alarm classes.



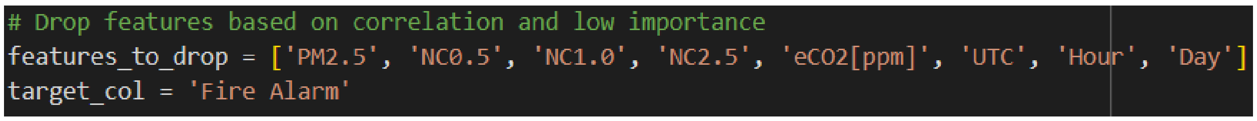
```
Before SMOTE: Fire Alarm
1    0.714634
0    0.285366
Name: proportion, dtype: float64
After SMOTE: Fire Alarm
0    0.500217
1    0.499783
Name: proportion, dtype: float64
```

Feature Selection

As the dataset contains a large number of features, it is important to perform a more in-depth analysis to determine which features are necessary for the success of our model. A random forest classifier was utilized to determine feature importance by determining how vital each feature was to the splitting of data across the trees in the forest.



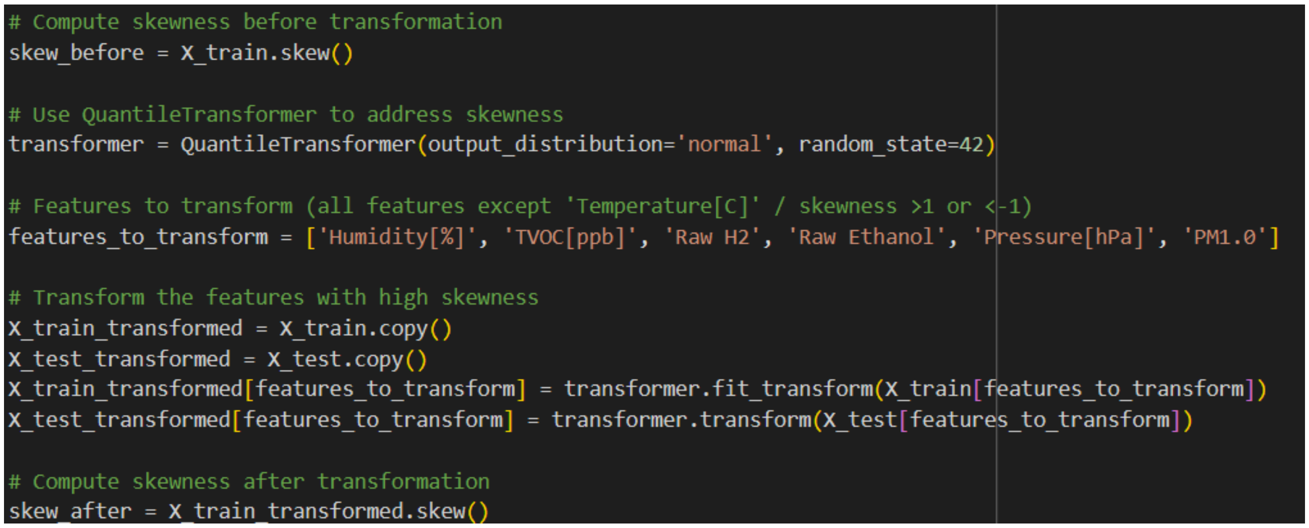
Notably, TVOC and pressure were of the highest importance while eCO2 had the lowest. Among the highly correlated features stated previously, PM1.0 has the highest importance. Therefore, it will be kept while the others will be dropped. Keeping them all would lead to redundancy and bias as the high correlation is indicative of capturing the same information. So, PM2.5, NC0.5, NC2.5, NC1.0 and eCO2 are dropped.



```
# Drop features based on correlation and low importance
features_to_drop = ['PM2.5', 'NC0.5', 'NC1.0', 'NC2.5', 'eCO2[ppm]', 'UTC', 'Hour', 'Day']
target_col = 'Fire Alarm'
```

Addressing Feature Skewness

With nearly all features being skewed, the need to address this issue was necessary as highly skewed data can hurt model performance if a model were to assume any form of normality or linearity. Essentially, using highly skewed data could lead to bias in the model.



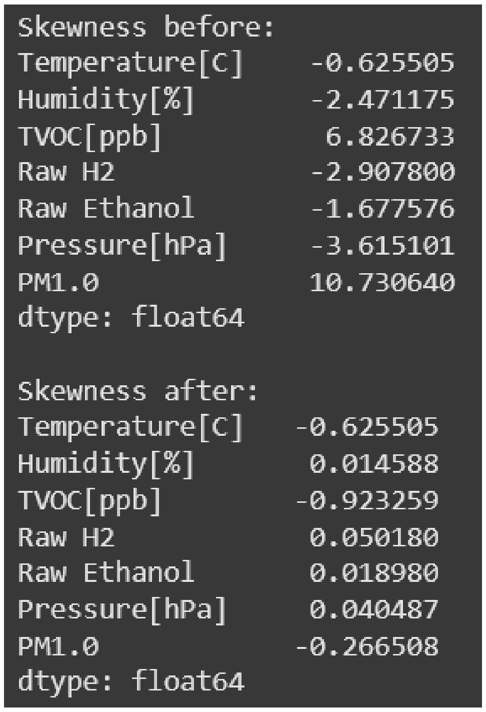
```
# Compute skewness before transformation
skew_before = X_train.skew()

# Use QuantileTransformer to address skewness
transformer = QuantileTransformer(output_distribution='normal', random_state=42)

# Features to transform (all features except 'Temperature[C]' / skewness >1 or <-1)
features_to_transform = ['Humidity[%]', 'TVOC[ppb]', 'Raw H2', 'Raw Ethanol', 'Pressure[hPa]', 'PM1.0']

# Transform the features with high skewness
X_train_transformed = X_train.copy()
X_test_transformed = X_test.copy()
X_train_transformed[features_to_transform] = transformer.fit_transform(X_train[features_to_transform])
X_test_transformed[features_to_transform] = transformer.transform(X_test[features_to_transform])

# Compute skewness after transformation
skew_after = X_train_transformed.skew()
```



```
Skewness before:
Temperature[C]    -0.625505
Humidity[%]       -2.471175
TVOC[ppb]          6.826733
Raw H2            -2.907800
Raw Ethanol       -1.677576
Pressure[hPa]     -3.615101
PM1.0             10.730640
dtype: float64

Skewness after:
Temperature[C]   -0.625505
Humidity[%]       0.014588
TVOC[ppb]        -0.923259
Raw H2            0.050180
Raw Ethanol       0.018980
Pressure[hPa]     0.040487
PM1.0            -0.266508
dtype: float64
```

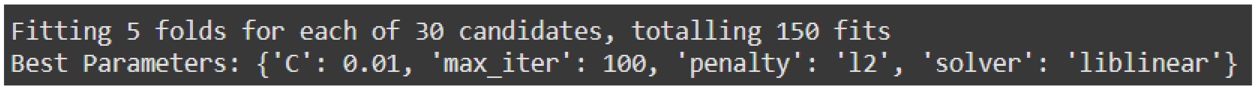
To fix the skewness in the features, QuantileTransformer was utilized to help properly distribute our data to follow a normalized distribution. In a high level, the transformer replaces each value with its quantile, which is the sorted rank of the value divided by the total number of values in the feature. Then the quantiles are mapped to a normal distribution. Doing this method transforms the data so there is less skew while also maintaining the order of those numbers. In exchange, the units and corresponding interpretability of the data is lost. However, as the goal is only to improve the models' accuracy, not analyze things like thresholds, plus the data is already scaled, this is worth it. All features previously determined to be important to our model are now within ± 1.

Hyperparameter Tuning

After adjusting the data to this point, the last step was to tune the hyperparameters to find the best fit for each of the chosen models. 5 folds were used for each of the models in the process.

Logistic Regression:

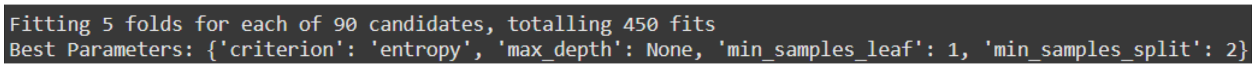
The penalty to provide regularization to avoid overfitting was chosen between l1 and l2. C (the strength of regularization) was chosen between 0.1, 1, 10, 100. And the max iterations for the model to converge was chosen between 100, 500, 1000



```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'C': 0.01, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
```

Decision Tree:

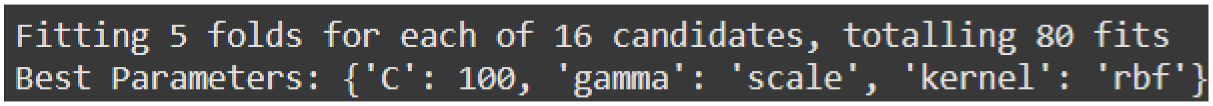
The function to measure each split quality was chosen between entropy and gini. The max depth of the tree to halt overfitting was chosen between none at all, 5, 10, 20, 30. The minimum number of samples required to split a node was chosen between 2, 5, 10. A higher number could have prevented premature splitting, reducing overfitting. Lastly, the number of samples required for a leaf node was chosen between 1, 2, 4.



```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

SVM:

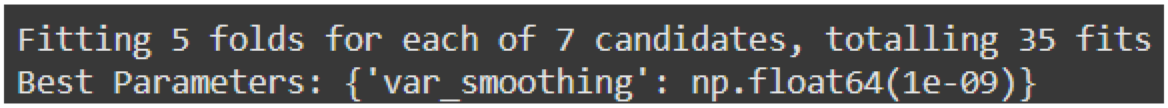
C, the regularization parameter was chosen between 0.1, 1, 10, 100. The lower it is, the more generalization and less overfitting will occur, but the cost may be underfitting. The kernel for the decision boundary was chosen between linear and rbf. This will essentially be chosen based on how linear the data is. And lastly, gamma controls the influence of each training sample on the kernel making the decision boundary either smooth with more bias or complex with more variance and overfitting. This value was chosen between scale and auto. One thing to not is that unlike the others, this tuning was done with only 20% training data as it was taking too long otherwise.



```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Best Parameters: {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
```

Naive Bayes:

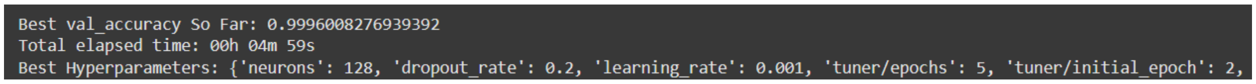
For naive bayes var smoothing was tuned. This is a miniscule value added to the variance of each feature to improve numerical stability and prevent division by 0. As naive bayes relies on a normal distribution, the variance and stability matters. The values were chosen from 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3.



```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
Best Parameters: {'var_smoothing': np.float64(1e-09)}
```

MLP:

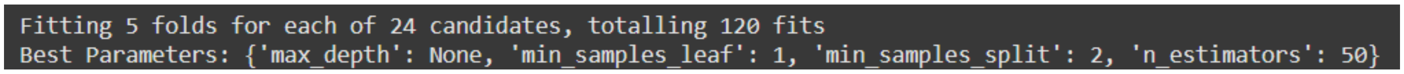
The number of neurons in the hidden layers was chosen between 32, 64, 96, 128. Too high can mean high complexity, overfitting and longer training while too low may be underfitted. Learning rate / the step size used during gradient descent can decide how quickly the cost function converges and whether or not a minima is missed via overshooting. This was chosen between 0.001 and 0.001. Drop out rate controls what percent of neurons are dropped in an iteration, controlling overfitting. This was between 0.2, 0.3, 0.4, 0.5. One thing to note was that the keras tuner was used for this unlike the others which which utilized grid search.



```
Best val_accuracy So Far: 0.9996008276939392
Total elapsed time: 00h 04m 59s
Best Hyperparameters: {'neurons': 128, 'dropout_rate': 0.2, 'learning_rate': 0.001, 'tuner/epochs': 5, 'tuner/initial_epoch': 2,
```

Random Forest:

The number of trees in the forest was chosen between 50, 100. The max depth of each tree was between None, 10 and 20. The minimum samples per split was 2, 5 and minimum samples per leaf was 1 and 2.



```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
```

Model Fusion

Combining the models of Logistic Regression, Decision Tree, SVM, and Naive Bayes was another attempt to improve model performance and build a final ensembled model with the highest performance. Instead of utilizing a majority vote, a weighted average soft vote of the probability outputs was found to be more effective in which each model's contribution is proportional to how well it performed (F1-Score).

The soft voting consists of using the probability prediction of each label from each model rather than simply using the predicted label like in hard voting. With this, the models' confidence was able to be captured in the ensemble, potentially leading to better results.
To get the weights for each of these models' prediction percentages F1 score was calculated using the model's prediction. Precision focuses on minimizing false positives and recall focuses on false negatives. While an argument could be made that false negatives are more important in regards to detecting fires, F1 was still chosen for the balance between the two. Then the weights were found and normalized by dividing each F1 score by the total of F1 scores.
Putting it all together, the prediction probabilities were multiplied by the weights and that product was used for the final prediction.

Experimental Results

Logistic Regression:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	0.8956	0.8970	2733	861	447	8485	0.88	0.86	0.87	0.96
Balanced	0.9011	0.9095	3298	277	962	7989	0.87	0.91	0.88	0.97
Feature Selection	0.8983	0.9059	3258	317	957	7994	0.87	0.90	0.88	0.96
Skewness	0.8434	0.8348	2886	689	1272	7679	0.81	0.83	0.82	0.90
Hyperparameter Tuning	0.8433	0.8349	2884	691	1272	7679	0.81	0.83	0.82	0.90

Decision Tree:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	0.9998	1	3593	1	1	8931	1	1	1	1
Balanced	0.9996	1	3574	1	4	8947	1	1	1	1
Feature Selection	0.9996	1	3575	0	5	8946	1	1	1	1
Skewness	0.9992	1	3569	6	4	8947	1	1	1	1
Hyperparameter Tuning	0.9998	1	3574	1	1	8950	1	1	1	1

SVM:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	0.9685	0.9678	3281	313	81	8851	0.97	0.95	0.96	1
Balanced	0.9551	0.9677	3548	27	535	8416	0.93	0.97	0.95	1
Feature Selection	0.9715	0.9797	3548	27	330	8621	0.96	0.98	0.97	1
Skewness	0.9993	0.9996	3573	2	7	8944	1	1	1	1
Hyperparameter Tuning	0.9998	1	3574	1	2	8949	1	1	1	1

Naive Bayes:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	0.7637	0.7657	870	2724	236	8696	0.77	0.61	0.61	0.94
Balanced	0.7712	0.6104	933	2642	224	8727	0.79	0.62	0.63	0.94
Feature Selection	0.8563	0.7625	2017	1558	242	8709	0.87	0.77	0.80	0.94
Skewness	0.9103	0.9056	3156	419	704	8247	0.88	0.90	0.89	0.96
Hyperparameter Tuning	0.9103	0.9056	3156	419	704	8247	0.88	0.90	0.89	0.96

MLP:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	0.9952	0.9944	3536	58	2	8930	1	0.99	0.99	1
Balanced	0.9926	0.9935	3558	17	76	8875	0.99	0.99	0.99	1
Feature Selection	0.9943	0.9937	3522	53	18	8933	0.99	0.99	0.99	1
Skewness	0.9991	0.9996	3527	3	8	8943	1	1	1	1
Hyperparameter Tuning	0.9996	0.9996	3572	3	2	8949	1	1	1	1

Random Forest:

	Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
Unbalanced	1	1	3594	0	0	8932	1	1	1	1
Balanced	0.9999	1	3575	0	1	8950	1	1	1	1
Feature Selection	0.9999	1	3575	0	1	8950	1	1	1	1
Skewness	0.9998	1	3574	1	1	8950	1	1	1	1
Hyperparameter Tuning	0.9998	1	3574	1	1	8950	1	1	1	1

Fused Model:

Test Acc	Train Acc	TN	FP	FN	TP	Precision	Recall	F1	AUC
0.9998	NA	3574	1	2	8949	1	1	1	1

Discussion/Analysis of Results

Starting with Logistic Regression, the baseline model with no changes started at an accuracy of 89%. Balancing the dataset allowed it to reach it's peak of around 90% accuracy, mainly attributed to finding more True Negatives. Feature selection had minimal impact on the results. There was a large dip in performance of about 5% after addressing the skewness, causing accuracy to drop to 84% as well as dropping precision, recall and F1 from around high 80s to low 80s. Hyperparameter tuning had no effect. Throughout the whole process generalization error remained low, meaning no overfitting. And AUC reamined near 1, so class differentiation was solid. The likely reason why the model accuracy dropped after transforming the data is because the relationships became distorted and Logistic Regression does not rely on normalized data, so it did not benefit from the change.

Moving onto the Decision Tree, there is not much to say about these results. None of the changes had much effect on any of the metrics, with the model being near perfect from start to end with very little (<10) false negatives or false positives. There was basically no generalization error either.

For SVM, the model started at 96% accuracy and ended at 99% accuracy. The first increase in accuracy was the feature selection, going from 535 False Negatives to 330. Then, the second increase was fixing the skewness which saw a drop from 27 false positives to 2 and then 330 false negatives to only 7. This is a massive increase. Throughout the whole process, there was basically no generalization error and difficulty in differentiating the classes.

Naive Bayes saw the most improvement, starting with 76% accuracy and 0.61 F1-score to ending with 91% accuracy and 0.89 F1-score. Balancing the classes did not have much effect besides actually lowering training accuracy to 61% while the test accuracy stayed at 76%. The test accuracy being so much higher than the training accuracy, shows the model was heavily underfitted and without much learning. On the other hand, the feature selection had a massive effect increasing the accuracy by nearly 10%. This makes sense as Naive Bayes relies on feature independence and removing highly correlated features would help with that. However, the difference from training accuracy was still the same.The last big jump was from fixing the skewness, giving about a 5% increase in accuracy while also fixing the generalization error issue. This also makes sense as Naive Bayes relies on a normal distribution of data and transforming the data to fix the skewness creates that.

MLP was much like a decision tree in that it was near perfect from the start with very little generalization error. The only notable improvements was after feature selection the number of false negatives dropped from 76 to 18 and after fixing the skewness the number of false positives dropped from 53 to 3.

Random Forest is the best baseline model created. It was near perfect from beginning to end with at most 1 false negative and 1 false positive at a time.

In comparison to random forest, the fused model is barely worse, with only 1 more false negative. Note the model has no training accuracy itself, as it only uses the predictions of the other models. It does not directly train. Still, this makes the ensemble models the best.

Conclusions

Overall, our model performed well from the start, and the improvements made helped to fine-tune our models to perform slightly better. The largest improvements made were to Naive Bayes by removing highly correlated features, and addressing any skew that may have been present. It started at roughly ~76% and ended at ~91%. On the other hand, the model that performed the worst was Logistic Regression as it actually got worse after the changes, dropping from 90% accuracy to 84%. The best models (excluding random forest) that performed well for all improvements were MLP and Decision Tree, which maintained a near perfect accuracy from the very start. SVM was eventually able to match their accuracy by the end of the hyperparameter tuning. Even when combining multiple models in an attempt at improvement, random forest still barely beats it out when comparing the confusion matrices. This means the basic random forest is still the best. Regardless, the project was successful in creating a model that can detect fires as only 3 misclassifications out of 12,526 samples is amazing. And even without the fusing, most of the other models like decision tree, MLP and SVM are perfectly capable of near the same results.

As for future improvements and things that can be researched further, more options in the hyperparameter tuning may yield better results. Trying to implement MLP into the fusion may be the final push to beat random forest. There are also a number of different transformations to address skew that may produce different results like Yeo-Johnson or simple log. Lastly, as the dataset already came extremely clean and refined, perhaps even less features could yield similar results. Performing PCA and keeping the top few features may be worth looking into. Finally, trying to analyze the few missed samples from random forest and the fused model may give insight into how to improve the model further.

Acknowledgments

While attempting to improve this model further, Dr. Amir Ghasemkhani suggested many of the improvement techniques attempted in this paper such as using ENN and feature selection analysis.
Stefan Blattman also created the dataset for his own project of similar nature and his blog gave insight into features and their relevance.

References

1. Blattmann, S. (2022, August 3). Real-time smoke detection with AI-based sensor fusion. Hackster.io. https://www.hackster.io/stefanblattmann/real-time-smoke-detection-with-ai-based-sensor-fusion-1086e6#toc-data-collection-6
2. Contractor, D. (2022, August 21). *Smoke detection dataset*. Kaggle. https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset