

Supervised Machine Learning:

Classification

IBM Skills Network

Project Report

By:

Fahd Seddik

Table of Contents

[Main Objective 3](#_Toc113374188)

[Brief Description 3](#_Toc113374189)

[Data cleaning & Feature engineering 4](#_Toc113374190)

[Classifier Models 5](#_Toc113374191)

[Recommended Model 6](#_Toc113374192)

[Key Findings and Insights 7](#_Toc113374193)

[Suggestions 7](#_Toc113374194)

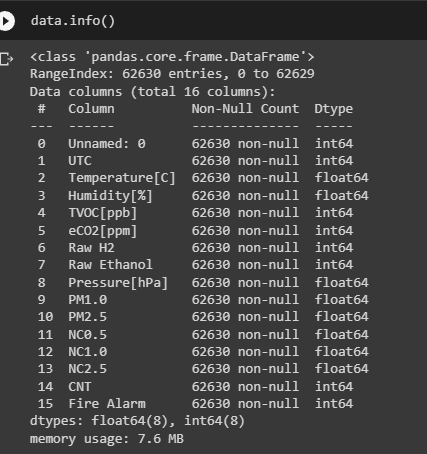
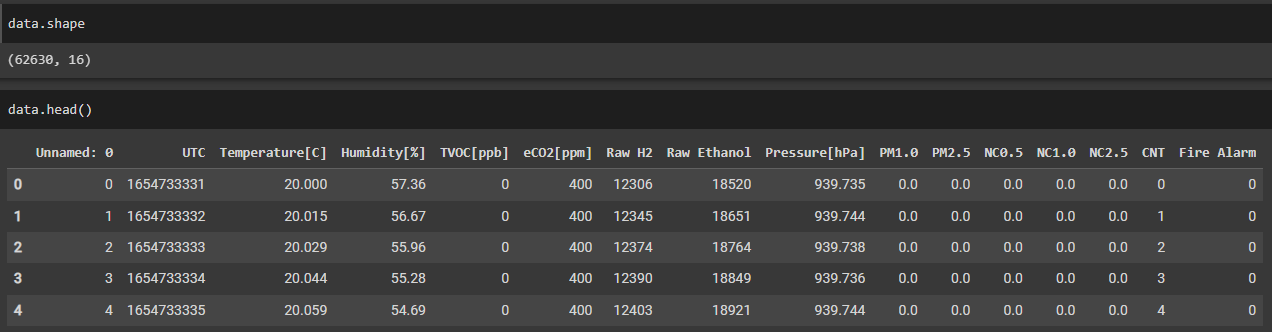
# Main Objective

A smoke detector is a device that senses smoke, typically as an indicator of fire. Smoke detectors are usually housed in plastic enclosures, typically shaped like a disk about 150 millimetres (6 in) in diameter and 25 millimetres (1 in) thick, but shape and size vary.

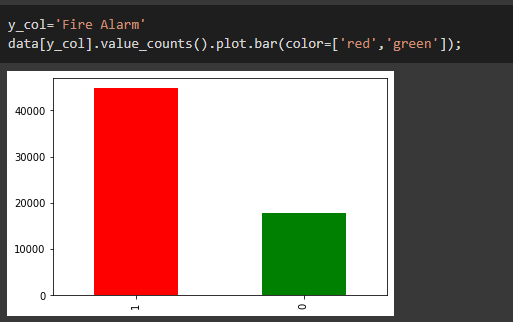
**In this data set (shown next section), we are given some sensor data and based on which we want to try and predict if there is fire or not.**

# Brief Description

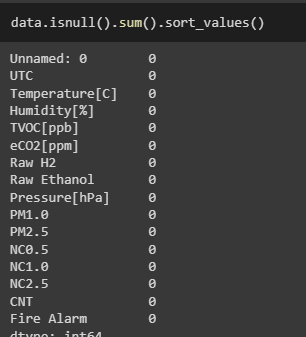
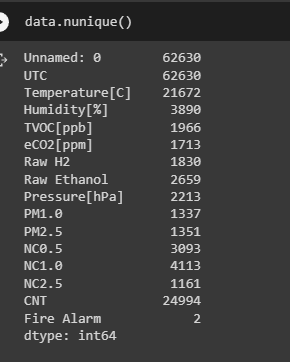
The data set has many useful information that would help us identify and classify our target. Shown below is a snippet of data.head() which shows us a brief about the values of our columns. The data set has **62,630 examples**, the target column is **‘Fire Alarm’** being either **0** or **1** and features include:

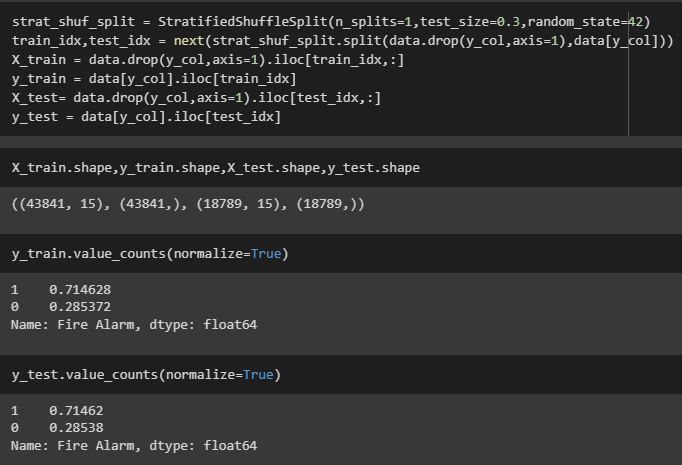
* UTC
* Temperature[C]
* 'Humidity[%]
* TVOC[ppb]
* eCO2[ppm]
* Raw H2
* Raw Ethanol
* Pressure[hPa]
* PM1.0
* PM2.5
* NC0.5
* NC1.0
* NC2.5
* CNT'

# Data cleaning & Feature engineering

We might first want to examine if there exists any imbalance in our target column. We will do that by using a bar plot for our ‘Fire Alarm’ target column. As shown below, we can clearly see that there is in fact imbalance in our target column which might hint about using either over-sampling or under-sampling techniques.

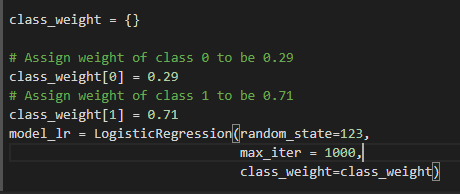
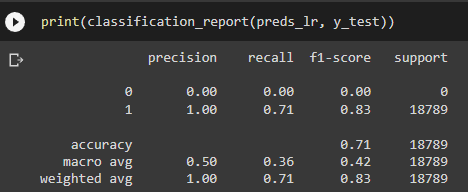
We can check for both missing values or duplicates and handle them right away as shown below by just using the isnull(). It is clear that we did not find any missing values. We can also see that there are no duplicates in our data set which indicates a good quality data set. When it comes to feature engineering, we might use it in our models later in the next section.



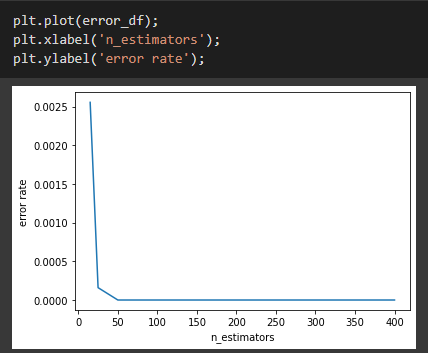
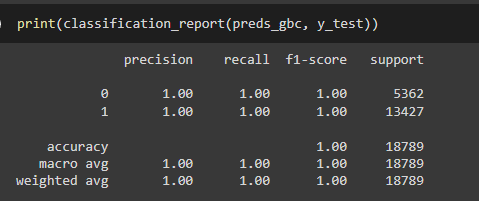
We also will use **StratifiedShuffleSplit** to ensure good distribution between our train and test sets respectively. We managed to split into 71.4% **1**s and 28.5% **0** for both our train and test target variables.

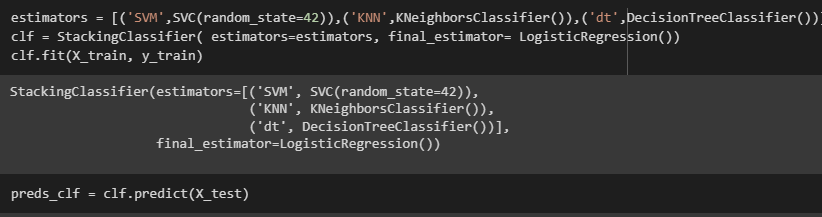
# Classifier Models

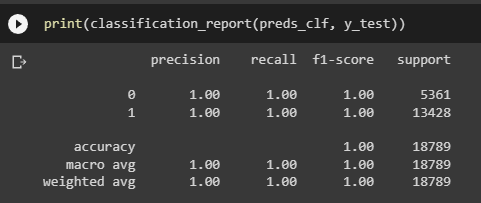
We will start with **LogisticRegression** as our base-line model so we can compare the rest to base-line. We will use class weighting too. Our Logistic Regression model did good enough with statistics shown below.



We will now move on to more complex models. For our second model we will use **GradientBoostingClassifier**. We tried fitting our model to several number of estimators to find the best out of all of them. Results hinted towards n\_estimators=50 being the best number of trees for the GradientBoostingClassifier.



We can see from the error plot above that it seems to plateau after 50 estimators so we will be using that. Our model produced outstanding results. For last model we will be using a **StackingClassifier** this would incroporate SVC, KNN, and DecisionTreeClassifier and will use a final\_estimator of a LogisticRegression model.



# Recommended Model

Based on the statistics shown above we can clearly state that the recommended model is **GradientBoostingClassifier** and this is due to the extraordinary performance demonstrated in the previous section. The model has performed well on all scores with number of estimators of only 50. We choose that model because it is not as complex and still does the needed job and has good scores.

# Key Findings and Insights

We can see that our StackingClassifier did perform well. However, it is prone to over-fitting as the more complex a model gets the more it is likely to over-fit. Thus, even though it performed good on the test set but it would be preferred to choose the less complex model for better interpretability (**next section for improvements**). When it comes to LogisticRegression the model’s performance was moderate. GradientBoostingClassifier obviously performed really well on our train set.

# Suggestions

It is certain that further analysis and more models could be applied to this data set and maybe we can have better results. However, some of the suggestions I see for our next steps include having a **GridSearchCV** to try to eliminate over-fitting while further enhancing our models to choose the best hyperparameters for each of them. We can also use these models and save them using the **pickle** library for later use in more sophisticated models or act as a “teacher model” for data distillation.