Wildfires Classification

The motivation for this project comes from the increase in the amount of wildfires occurring worldwide over time, which not only provokes the direct damage to inhabitants, the native flora and fauna, but also has an indirect effect on the climate change. From one side, the combustion of forests release CO_2 and other greenhouse gasses to the atmosphere. Moreover, a greater impact on the atmosphere is caused when the dead plants decompose over the following decades, plus the amount of CO_2 that these trees will not be removing from the atmosphere as living trees would.

With all the data collection happening currently, I believe that some interesting topics of research could be done in this area. An obvious one is the prediction of the probability of a wildfire happening as it would allow the firefighters to be ready on the right time and the right place with higher accuracy. Also predicting the cause once the catastrophe has taken place can be of help as many fires remain uncertain about its original cause.

1. About the data

The data set I've used for this project contains a detailed description of all wildfires occurred in Mallorca from 2010 to 2019. Each observation has many features such as the origin's coordinates and timestamp, as well as number of affected municipalities, burnt area and cause for instance.

2. Data Wrangling

2.1 Feature Selection

First thing I did was to take a glimpse to all features to select the most relevant and usable ones. Some variables where directly deleted for having one unique value on the whole data set, probably because this data is part of a larger set. Other features were removed due to it's lack of information such as the number of municipalities involved on each wildfire.

Table 1: Number of Municipalities affected by the fires

Number of Municipalities	Frequency
1	686
2	1
3	1

Other variables were just necessary for the administration to identify the fires, yet not for my project.

Next, I gave the corresponding data type to each feature:

- **Number**: *FarmingArea*, *NoTreesArea*, *ForestalTotalArea*, *X*, *Y*, *Year*.
- **Datetime** ('%d/%b/%Y %H:%M:%S'): *Detected*, *Extinted*.
- **Character**: all others

Table 2: Preselected Features

Features	Description
Year	Year
Municipality	Municipality
Square	Origin's location on a map
X	Coordenate X
Y	Coordenate Y
Detected	Origin timestamp
Extinted	Extintion timestamp
Cause	Fire Cause
Motivation	Fire Motivation
FarmingArea	Farming area burnt (ha)
NoTreesArea	Area without trees burnt (ha)
ForestalTotalArea	Forest total area burnt (ha)

2.2 Feature Engineering

In this part of the project I proceeded to modify the *datetime* features. I thought that a more suited variable for the learning of the algorithms would be to compress *Detected* and *Extinted* variables into a variable containing the total time, in minutes, that the fire was alive. Also from the *Detected* values I extracted the day of the week and month of the year in which the fire was originated. In this process I removed two features and added three new ones.

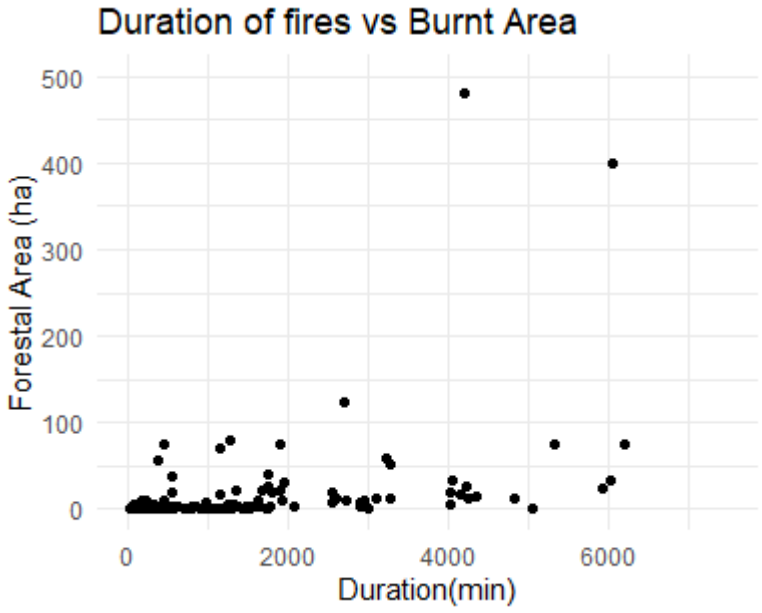

Figure 1: DurationVsArea

2.3 Target Variable

Initially there were 46 different labels describing the cause of the fires, which off course would be overwhelming for any algorithm to classify. So I proceeded to analyze the frequency of the values. First, I removed 10 observations which cause was classified just once, also I confirmed that the total area burnt by these causes was minimum.

Then, after and exhaustive analysis of all different causes I compressed the remaining ones in 8 distinct causes which I found they are the main groups in wildfires classification.

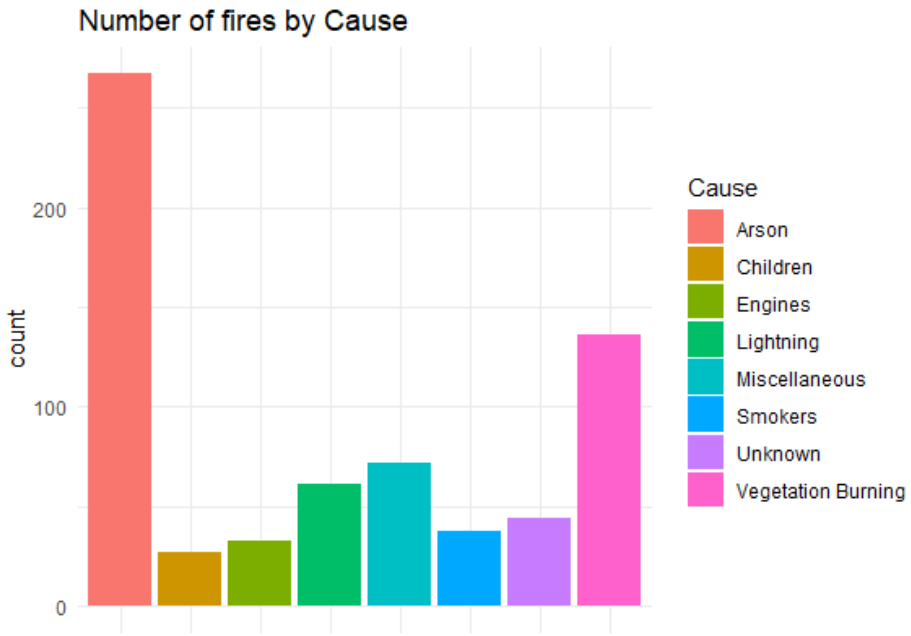

Figure 2: Number of observations by Cause

The causes were reduced from 46 to 8. In addition, during this analysis I found out that the *Motivation* variable is only a short description added to the *arson* wildfires, so I dropped this feature.

3. EDA

Categorical Variable

Some algorithms can not deal with categorical variables, yet due to the large number of different municipalities in this set a *OHE* wouldn't be convenient. During the model development I used to diferent procedures in order to fit the models with just numeric variables. This process was applyed to the *XGBoost* model for instance.

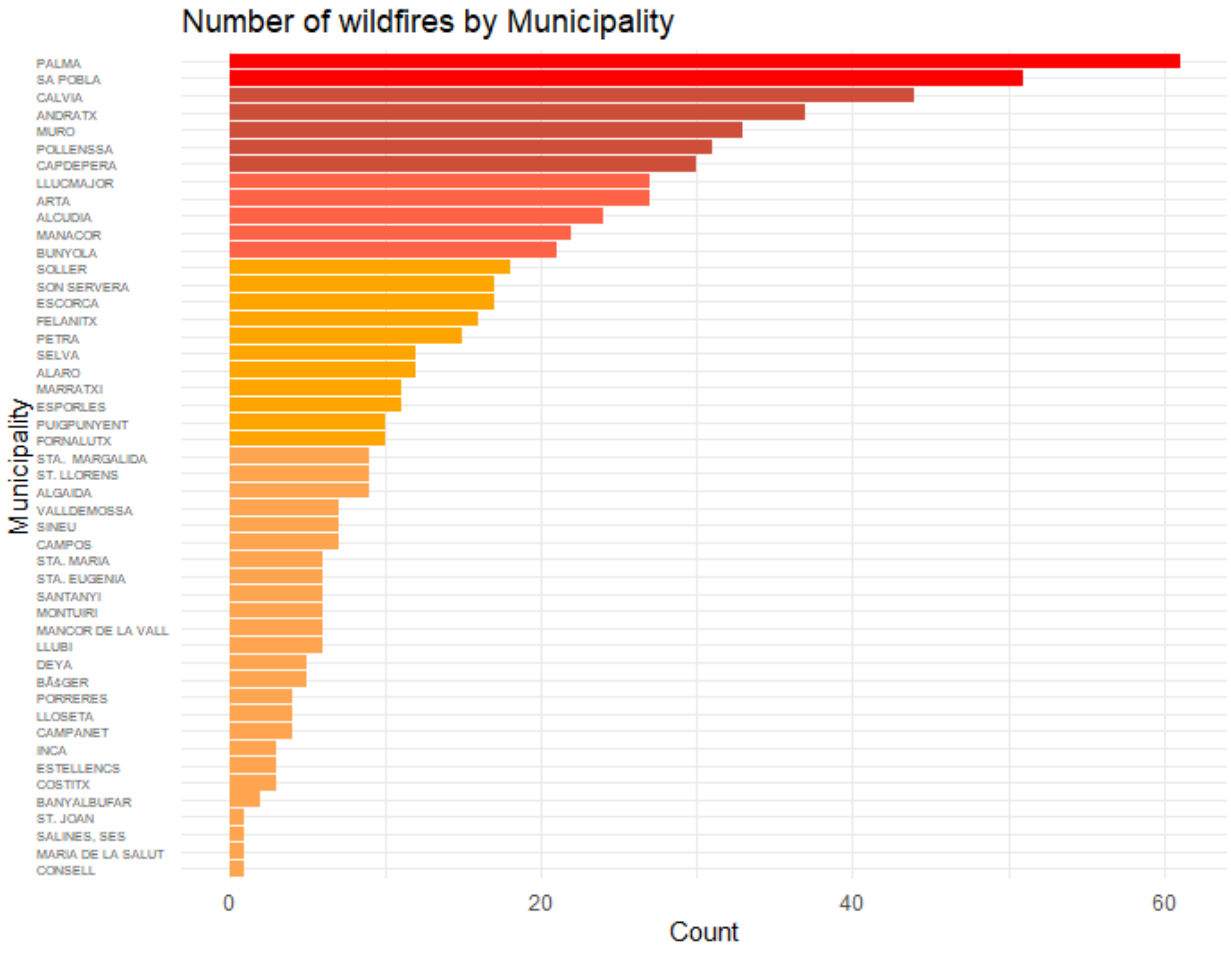

Figure 3: Number of Wildfiresfires by Municipality

Distribution of fires in time.

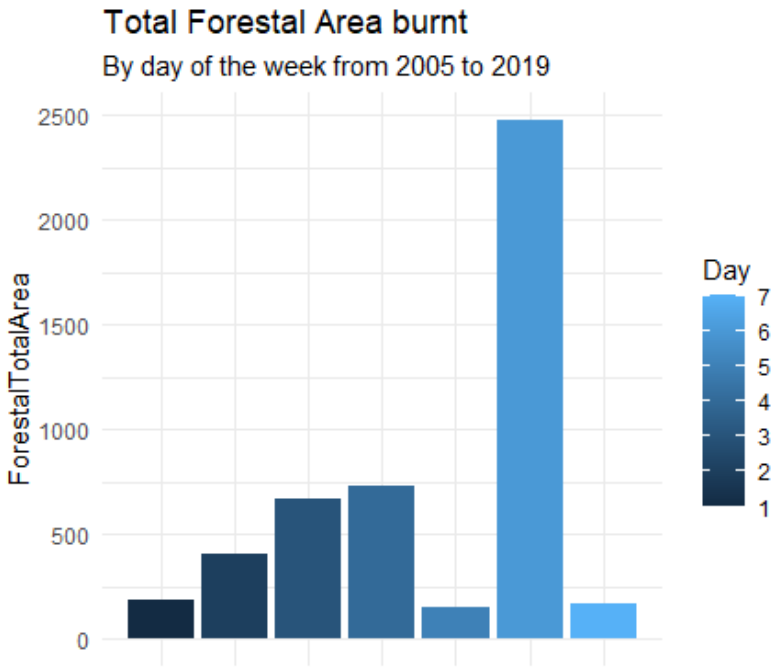

Figure 4: Area of forest burnt by day of the week

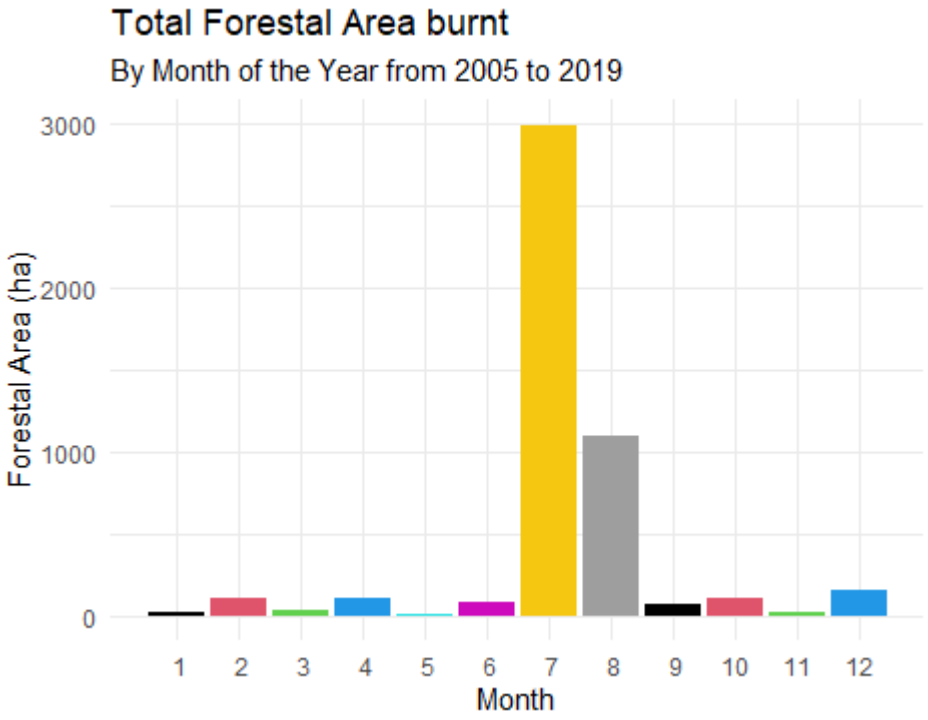

Figure 5: Area of forest burnt by month

Feature Correlation

For the correlation analysis I removed the categorical variables.

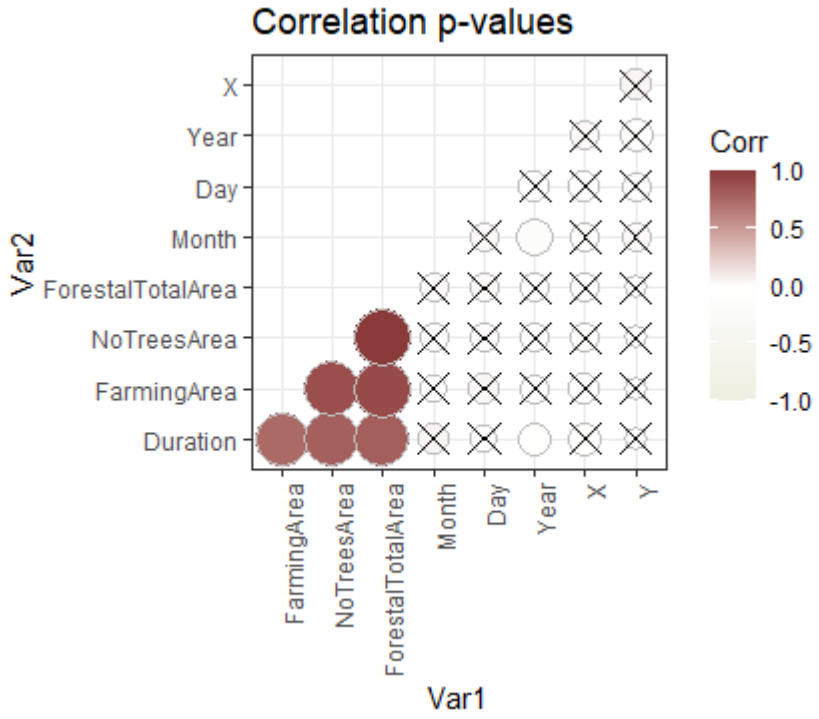

Figure 6: Correlation with significancy

The total area of forest burnt and the area of forest without trees burnt are almost perfectly correlated with statistical significance, thus I removed the *NoTreesArea*. Farming area burnt is correlated with the total area of forest too, yet I kept it for the model development.

4. Model Development

This section presents the analysis of different algorithms' performance on classifying the wildfires in 8 distinc categories. The algorithms I decided to compare are the followings:

- Linear Discriminant Analysis
- Random Forest
- Boosting
- XGBoosting
- KNN

This selection is based on the multiclass classification algorithms introduced in the text book *Introduction to Statistical Learning*. Regarding the first four algorithms, they all learn in a supervised way looking for

patterns in the training data and are listed in descending order of complexity. While the KNN is also a supervised learning algorithm, it uses a proximity measure method in order to classify the observations. Due to the problem's nature, the **accuracy** will be measured in the **proportion of true positives** classified by each model.

First, I loaded the final data, modified the feature's order and transformed the character variables to factor, which is demanded by many algorithms in *R*.

```
## 'data.frame':    678 obs. of  11 variables:
##  $ X                : int  504173 470960 463583 533237 499461 505017 532206 538067 471124 477109 ...
##  $ Y                : int  4405455 4401319 4388686 4388359 4396769 4413469 4385752 4397290 4393316 4
##  $ Square           : Factor w/ 52 levels "A09","B06","B07",..: 10 6 15 33 19 4 33 22 16 7 ...
##  $ Municipality     : Factor w/ 48 levels "ALARO","ALCUDIA",..: 36 15 17 42 23 33 42 12 8 20 ...
##  $ Day              : int  6 1 6 1 6 1 2 5 6 6 ...
##  $ Month            : int  2 2 2 2 2 2 2 2 3 3 ...
##  $ Year             : int  2019 2019 2019 2019 2019 2019 2019 2019 2019 2019 ...
##  $ Duration         : int  107 449 81 90 45 155 66 41 48 36 ...
##  $ FarmingArea      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ForestalTotalArea: num  0.1 0.32 0.02 0.1 0 0.08 0.01 0.01 0.01 0.01 ...
##  $ Cause            : Factor w/ 8 levels "Arson","Children",..: 1 8 8 8 1 8 1 1 8 3 ...
```

Then I divided the data in training and test sets in a **75-25% random split**.

Table 3: Training set, explanatory variables

	X	Y	Square	Municipality	Day	Month	Year	Duration	FarmingArea	ForestalTotalArea
136	467450	4379408	E04	PALMA	5	7	2017	366	0.00	0.35
592	477688	4404806	C05	FORNALUTX	1	9	2011	90	0.00	0.01
269	504189	4405175	C08	SA POBLA	1	11	2015	75	0.00	0.03
261	498166	4400226	C07	BÃšGER	6	7	2015	87	0.00	0.01
340	508494	4408790	C08	ALCUDIA	7	6	2014	2553	0.00	19.30
264	506832	4417931	B08	POLLENSSA	6	8	2015	191	0.05	0.20

Table 4: Training set, target

	Cause
136	Smokers
592	Miscellaneous
269	Arson
261	Arson
340	Lightning
264	Arson

As you can see from the tables above, the set in use is relatively small, thus I decided to use a **Nested CV** approach for the hyperparameter tuning of the *boosting* and *XGB* algorithms. Hence, I randomly separated the training data again in 5 equal-sized folds and saved it for later on.

The followed procedure to find the best model was to train all algorithms where the best training performance resulted with the *Boosting* with a 48.84% of train accuracy. However, I decided to compute the test accuracy of the 5 other models too, so that I could better compare their performances with a confusion matrix of their classification on the test data. In the last section of model developing I did a grid search to find the optimal parameters for the *boosting* algorithm on the training data and finally compute the test accuracy of the best model.

1. LDA

LDA requires all variables to be numeric, thus I transformed the factors to numbers. Next, in order to avoid problems of collineality between variables and keep maximum information possible, I computed the principal components of my features to fit the *LDA* model.

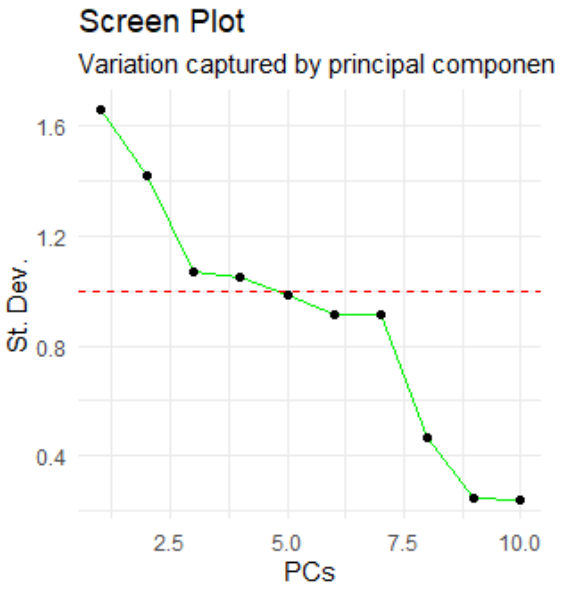

Figure 7: Principal Component's st.dev

I kept the first four principal components as they have an *st.dev* over 1. Then, at this point I trained two models, one using the four first principal components and a second one using the original variables. Both training accuracy was quite similar, slightly better on the second model. Hence, I used the second model to predict the test data.

$$Cause(PC1, PC2, PC3, PC4)\ (M1)$$

$$Cause(X, Y, Square, Municipality, Day, Month, Year, Duration, FarmingArea, ForestalTotalArea)\ (M2)$$

Table 5: LDA Performance

Models	Training_Acc	Test_Acc
M1	0.4271654	NA
M2	0.4409449	0.4765

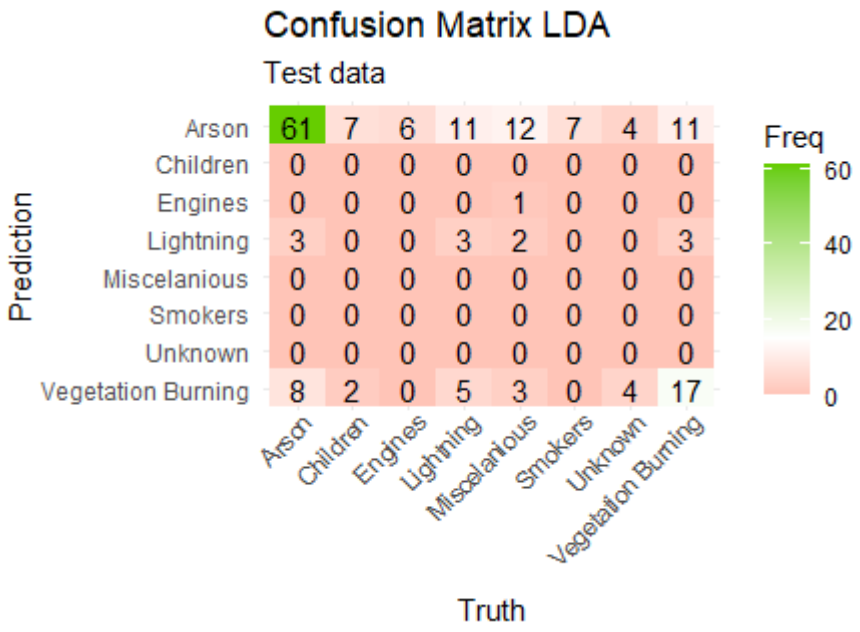

Figure 8: Confusion Matrix LDA

2. Random Forest

Regarding the Tree based models, I jumped directly to *Random Forest* and *Boosting* for **better accuracy** and **lower variability**.

By default the number of trees is 500 and the predicted class for each observation is the most frequent of all resulting values from the trees in which an observation has been evaluated, generally in 333.33 out of the 500 trees, i.e. 2/3. This means that each tree is not developed using the whole data set, the other third of the observations which are not used to develop an specific tree, namely the **Out of Bag** sample, is then used to evaluate the model. Thus, the validation error can be computed without using a CV approach which would be more computationally expensive.

This algorithm has a tuning parameter m, which is the number possible features to use in each split. If the m is smaller than the total number of variables, the algorithm randomly selects m variables to compete for the next split.

After performing a hyperparameter searching, the optimal value for m happened to be 6 in two of the 5 iterations and the following plot shows it's training performance:

Figure 9: Out of Bag Error

It seems clear that the error rate is stable at 500 trees so it is not necessary to develope a larger forest. Finally, after training the optimal model and testing with the respective sets, the model's performance was:

Table 6: Random Forest Performance

Models	Training_Acc	Test_Acc
m=5	0.4805069	0.4411765

For this algorithm the $TrainingAcc = \frac{100 - OBB_{Error}}{100}$, where the OBB_{Error} is in %.

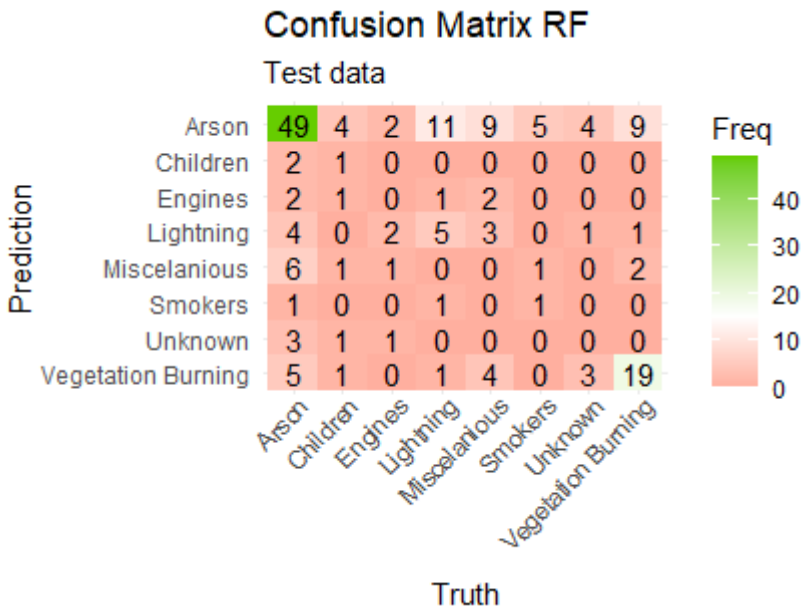

Figure 10: Confusion Matrix Random Forest

3. Boosting

The idea in boosting is to develop multiple weak prediction models sequentially, in a way that each of them uses the error of the previously developed ones in order to generate a stronger model, with better predictive power and better results stability.

Applying boosting to decision trees rises the complexity of the algorithm. In this case there are four important parameters to take into account: the **maximum depth of the each individual tree**, the **number of boosting iteration**, i.e trees, the **learning rate** and the **minimum size of each leaf**. Hence, again I performed a hyperparameter searching to find the optimum combination of parameters for my problem. Now, the boosting algorithm do not uses a bootstrap approach to validate each model, so I performed a nested CV with an inner loop to tune the paramaters and an outer to have an unbiassed evaluation of the models.

Table 7: Best Train Performing Boosting Model

Parameters	Optimum_Value
Interaction Depth	6e+00
Boosting Iterations	1e+02
Learning rate	1e-02
Minimum leaf size	1e+01

Finally, I trained the optimum model with the training set and predicted the observations from the test set, leading to the following results:

Table 8: Boosting Performance

Training_Acc	Test_Acc
0.4883699	0.4529412

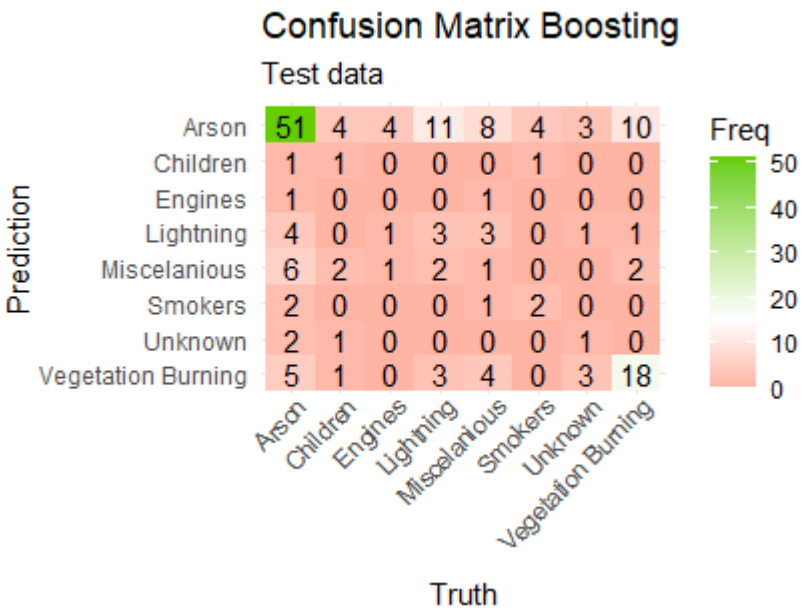

Figure 11: Confusion Matrix Boosting

4. XGB

The **Extreme Gradient Boosting** uses the idea of the *boosting* algorithm, optimizing its learning with a *Cost function* to minimize. Thus this algorithm still uses the error of previous trees to develope new ones, yet has an addition restriction and is that the performance of a new trees has to be better than the performance of the previous in order to be used, trying to find the minimum value on a cost function. This algorithm has usually better accuracy with heterogeneous data.

This model requires their variables to be numeric, thus I've compared two alternatives. First, I created some dummies out of the two qualitative variables. Secondly, as the qualitative variables are already factors I've transformed them in to *numeric* representing the level of the respective factor. Results showed that the second option was optimal with this specific data.

Using both data sets just described, I performed a *nestedCV* combined with a grid search to find the best model. The hyperparemeters are: **boosting iterations**, **max tree depth**, the **learning rate**, **minimum loss reduction**, subsample ratio of columns, minimum sum of instance weight and subsample percentage. Regarding the last 3 parameters I have mentioned I used the default values of 0.75, 0 and 1.

Finally, I trained the optimum model with the trained set and predicted the observations from the test set, leading to the following results:

Table 9: Best Train Performing XGBoosting Model

Parameters	Optimum_Value
Boosting Iterations	7e+02
Max Tree Depth	2e+00
Learning rate	1e-02
Minimum loss reduction	1e-04

Table 10: XGB Performance

Data	Training_Acc	Test_Acc
Dummies	0.4645881	NA
Númeric Factor	0.4725473	0.5117647

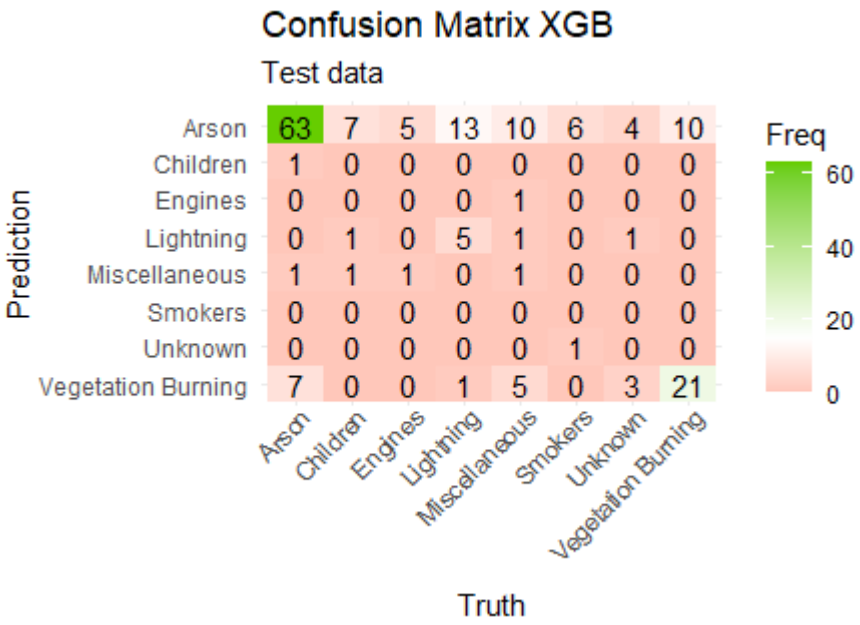

Figure 12: Confusion Matrix XGB

5. KNN

I decided to compare this algorithm too, as it uses a different approach compared to the tree-based algorithms. In this case the classification is done using proximity measures within the observations.
The only tuning parameter here is the number of neighbors that two observation must have in order to consider them similar. Thus, I trained the algorithm with different k values and evaluated the performance with a $LOOCV$ approach.

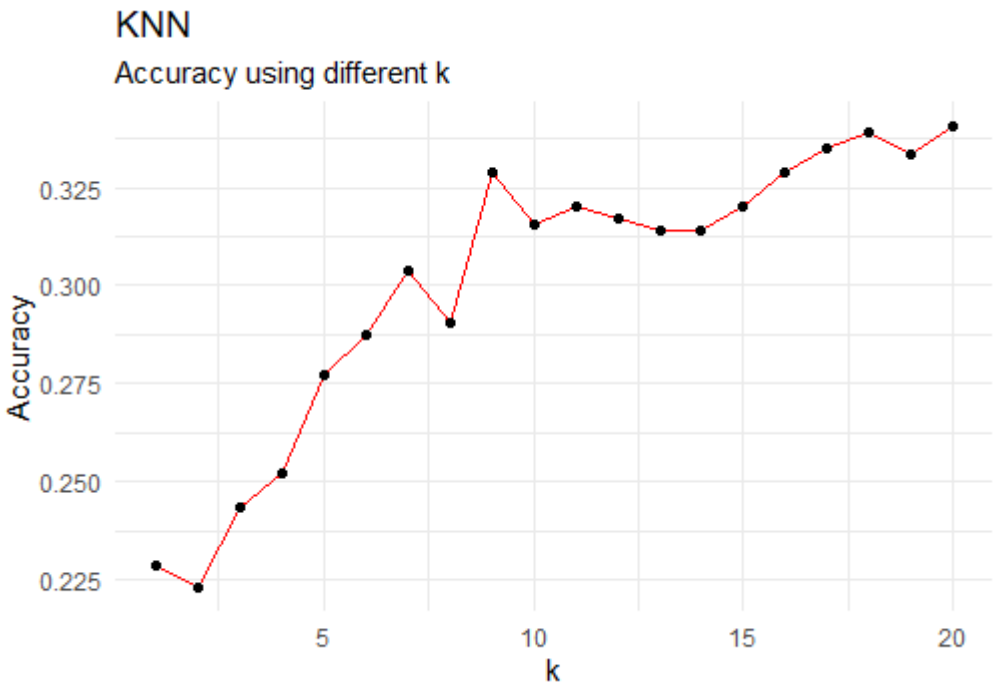

Figure 13: KNN Accuracy

The k which showed better accuracy was $k = 20$, yet in the previous plot we see that $k = 9$ already has a similar accuracy. With the latter model the performance was the following:

Table 11: KNN Performance

	Model	Training_Acc	Test_Acc
k	9	0.3289086	0.3823529

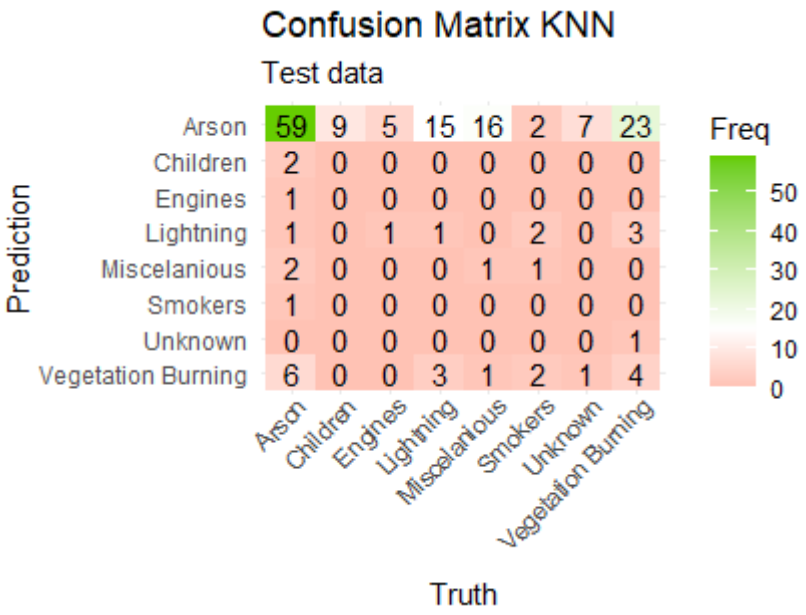

Figure 14: Confusion Matrix KNN

Best Model

Comparing five different algorithms resulted in **Boosting** as the best one, i.e. with the highest training accuracy. With a simple Cross-Validation on all the training data resulted in following optimal parameters:

Table 12: Best Train Performing XGBoosting Model

Parameters	Optimum_Value
Boosting Iterations	1e+02
Max Tree Depth	5e+00
Learning rate	1e-02
Minimum loss reduction	1e+01

The final model classified the test set with an **accuracy of 55.30%**.

Conclusions

Table 13: Frecuency of the whole dataset

Arson	Children	Engines	Lightning	Miscellaneous	Smokers	Unknown	Vegetation Burning
267	27	33	61	72	38	44	136

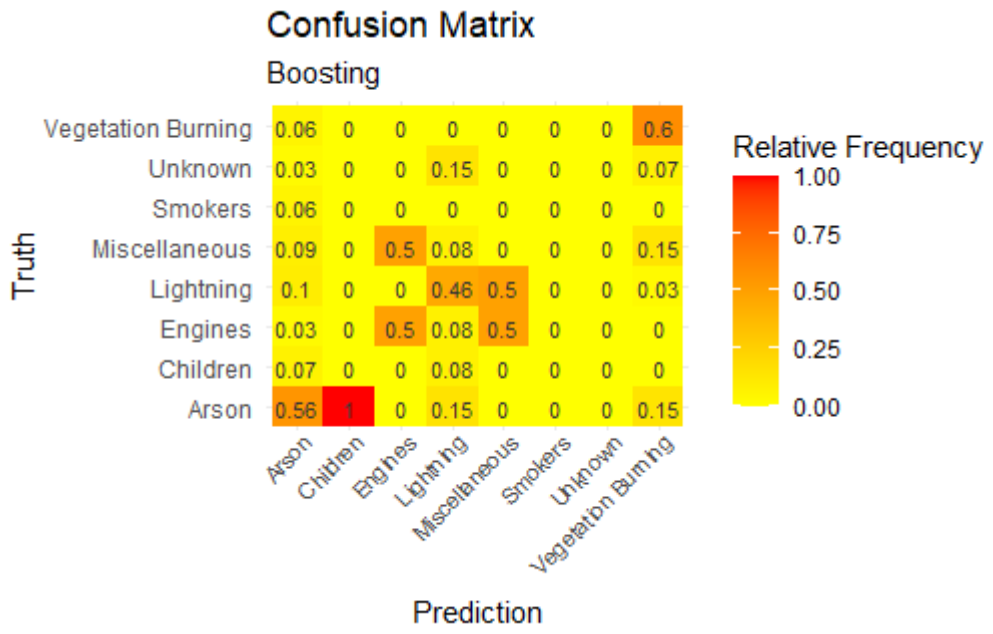

Figure 15: Confusion Matrix in proportion values, on the test data

The performance of the distinct models is pretty similar even though there is a large difference in complexity from *XGBoost* and *LDA* for instance. This similarity reflexes the complexity of the data, being 8 target categories in a relatively small set of 678 observations and 10 independent variables. Moreover, the information added by some variables is quite similar such as in the case of the X and Y coordinates and the *Square* which is a measure of location within a map. Also, the topic, predicting a wildfire's cause, is undeniably complex and requires many additional variables such as metrics describing the combustion of the vegetation of the area for instance. To sum up, all these factors lead to a similar performance among the algorithms.

Each model have a different confusion matrix, that is some algorithms classify better in some categories than others yet the *Boosting* model happened to have the highest proportion of true values. This model achieves to classify true positives in a wider range of categories, **XGB** and **LDA** had similar performance focused more in *Arson* and *Vegetation Burning*, whilst **Random Fores** and again **XGB** did a slightly better job on *Lighting*. Anyway, the model with highest proportion of true values on the training data is the **XGB**.

Analyzing the best performing model we have that the main predicted category is *Arson* followed by *Vegetation Burning* and *Unknown*. Whiles the categories with higher precision are *Arson* and *Vegetation Burning* and *Lightning*. My explanation for this fact is that, *Arson* clearly has the bast majority of the observations, actually, if a model predicts that all observations belong to this category the accuracy would already by roughly 39.38%. Then, regarding the other two, they are easily differentiated from the others as they usually occur in a specific time of the year.

Focusing on the **Boosting** model, from the confusion matrix we see that 4 of the classes have *0* true positives with an error rate of *1* that's probably due to the similarity between these categories; *Children*, *Engines*, *Miscellaneous*, *Smokers* and *Unknown.* Even though the miscellaneous category has more observations than the lightning one, the latter has a higher test accuracy that's probability because in the former category there is a mix of distinct unusual categories so it's extremely difficult to find a pattern, same with the *Unknown* one. The lack of accuracy predicting the fires originated by a child, an engine or a smoker can be associated with the model performance or the small amount of observation available for this three categories.

Still with the best performing model, I've extracted the feature's importance with the proportional amount of gain given by each feature with the tree's splits.

Variable Importance Relative influence that the boosting model associates to each variable.

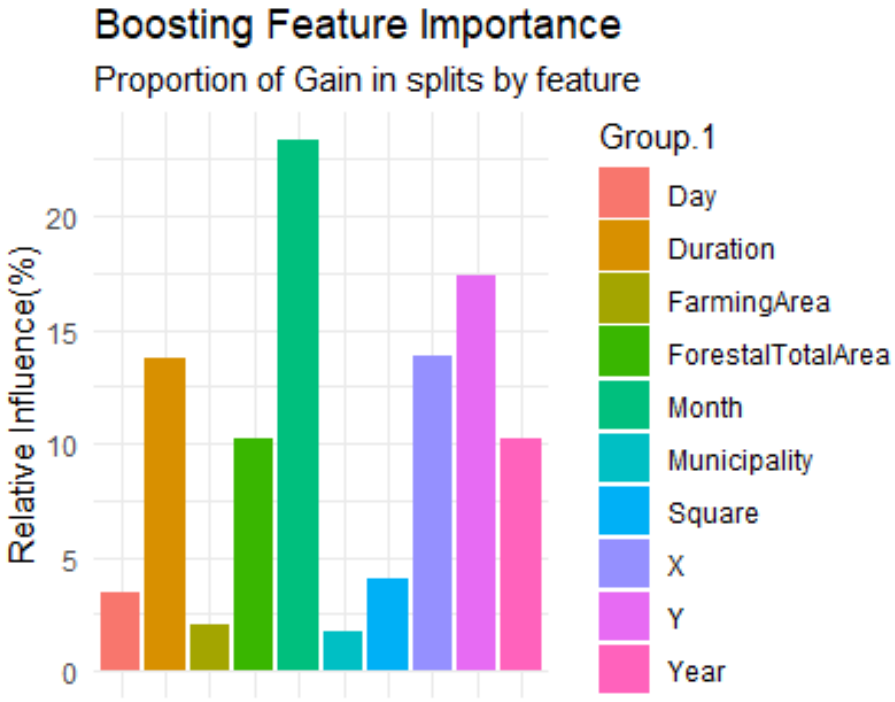

Figure 16: Proportion of Gain in splits by feature

Probably a larger data set with a wider description of each observation would lead to a better classification of what did originate each fire.

For the project's closures, I classified all observations corresponding to the **unknown observations** using a *Boosting* model, trained with the known observations.

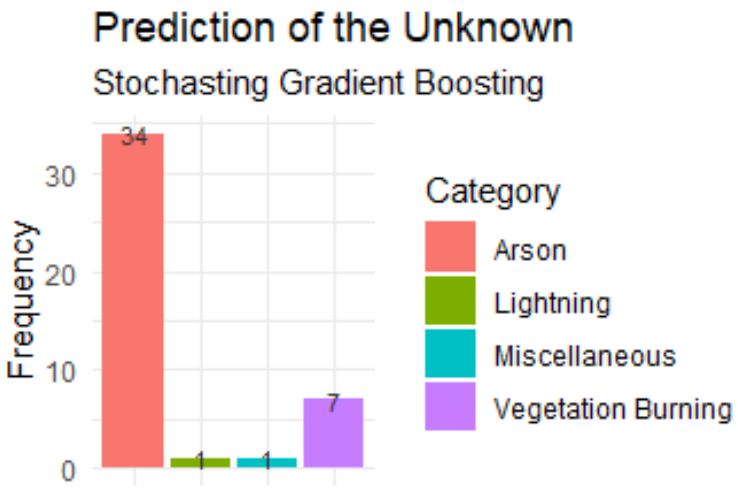

Figure 17: Prediction of the Unkown observations