Applied Competitive Lab in Data Science - Project Report

Liel Azulay, Tal Yehezkel, Chaim Weisberg, Jonatan Barr

Preprocess **2**
Feature Exploration **3**
Added Features **4**
Season Column 4
Time Of Day Column 5
Understanding Stat-Cause-Description column **6**
Feature Selection **8**
Evaluation **11**
Model Selection **12**
Imbalanced Data 16
Data Scaling 17
Optimization **18**
Hyper Parameters Tuning 18
Ensemble & Boosting **20**
Another Optimization Attempt 21
Feature Evaluation **22**
Final Model **23**
Error Analysis **24**
Results **26**
Validation Set Results 26
Test Results **27**

Preprocess

1. We removed duplicate rows in the dataset.

2. We started by splitting the dataset into train, test and validation sets, where the test is 20% of the data and the remaining 80% is divided - 20% for the validation and 80% for the train.
   Overall we have:
- train set - 64%
- test set - 20%
- validation set - 16%

3. We separated each data set from the column ‘STAT_CAUSE_DESCR’ which is the column we want to predict.

4. In order to keep the test set unbiased we made sure to replace missing values based on information that appears only in the train data.
   However, there were exceptions where filling in missing values with the most common is less appropriate. For example the date that the fire was contained. After exploring the data we found out that in most cases the contained date is the same as the discovery date, so we decided to fill in the missing values for the contained date using the values from the discovery date.

5. For values with a clear range such as longitude and latitude we made sure to validate that they are in the correct range and if not correcting them by the mode value from the training data.

6. We removed not important features (we will expand on this topic in the next section)

7. We calculated the correlations of the columns, and removed columns that are highly correlated to others.

8. We perform one-hot-encoding and frequency encoding on the categorical features.

9. We wanted all the data to be on the same scale, so in the end of the preprocess, before the model selection part, we tried to normalize the data using feature scaling with the StandardScaler from scikit-learn. (mapping to a fixed range,

between 0 and 1).The results without the scaling were better so we decided to not scale the data. (Later in the report we will note the differences in the results).

Feature Exploration

In order to better understand the features,
we went over the data in X_train, and divided our different features into categories:

Ordinal Features-
FIRE_SIZE_CLASS

Categorical Features-
SOURCE_SYSTEM, SOURCE_SYSTEM_TYPE, NWCG_REPORTING_AGENC, NWCG_REPORTING_UNIT_ID, NWCG_REPORTING_UNIT_NAME, SOURCE_REPORTING_UNIT_NAME,ICS_209_INCIDENT_NUMBER, FIRE_CODE, ICS_209_NAME, MTBS_ID, MTBS_FIRE_NAME, COMPLEX_NAME, STAT_CAUSE_DESCR, OWNER_CODE, OWNER_DESCR, STATE, COUNTY, FIPS_CODE, FIPS_NAME.

Continuous Features-
DISCOVERY_DATE, DISCOVERY_DOY, DISCOVERY_TIME, CONT_DATE, CONT_DOY, CONT_TIME, FIRE_SIZE, LATITUDE, LONGITUDE, FIRE_YEAR.

Deleted features-
We decided to delete all the id's columns:
FOD_ID, FPA ID, LOCAL_FIRE_REPORT_ID, LOCAL_INCIDENT_ID, OBJECTID.
The values of those columns are unique numbers for each incident which do not give us important information for the prediction.

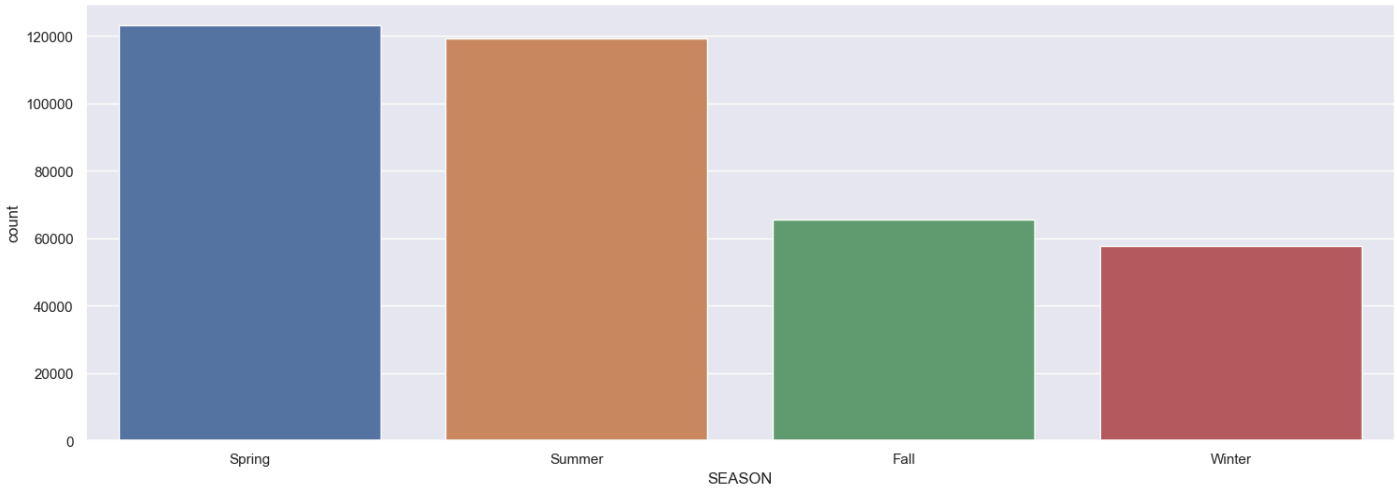
FIRE_NAME- the values of this field are mixed from two different databases -
from the fire report or from ICS-209 report, so the values are not consistent.
There are values that are just numbers, some that are a description of the fire and some that are the name of the location of the fire.
There is no clear indication of what information the feature provides us.

Shape - this column has no meaning and most of the cases have a unique value.

Added Features

Season Column

This column holds the season (Spring, Summer, Fall, Winter) in which the fire occurred.
In order to create this column we used the 'DISCOVERY_DOY' column that contains the day of the year in which the fire occurred.
In the following plot we can see the number of fires in each season from the train set:



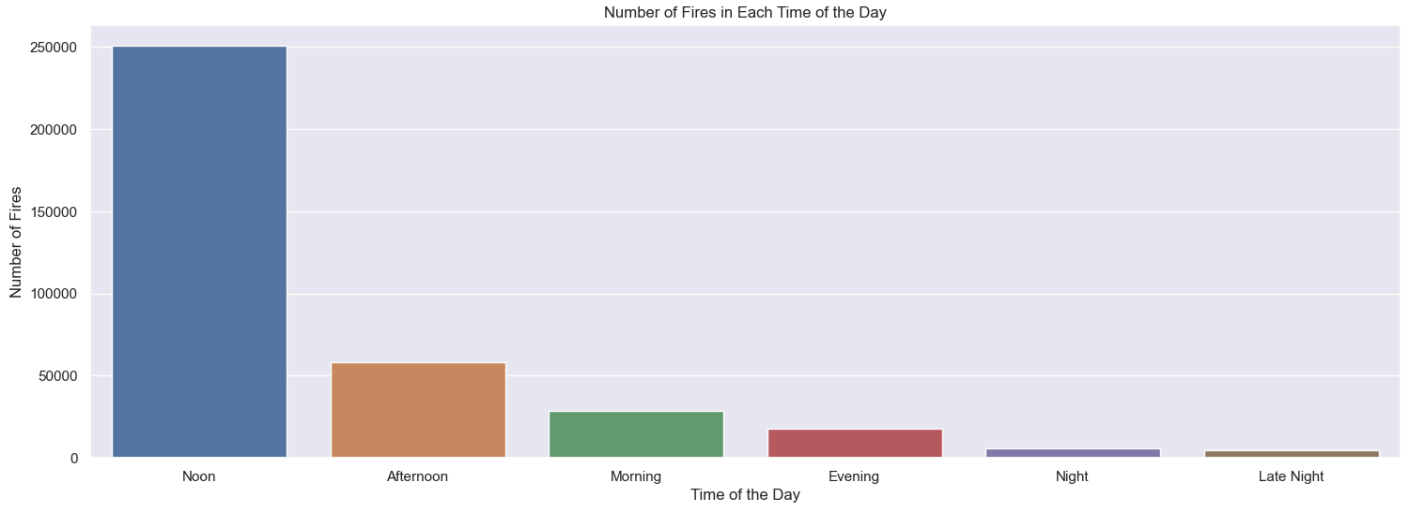
As we can see most of the fires occurred in the spring and summer.

Time Of Day Column

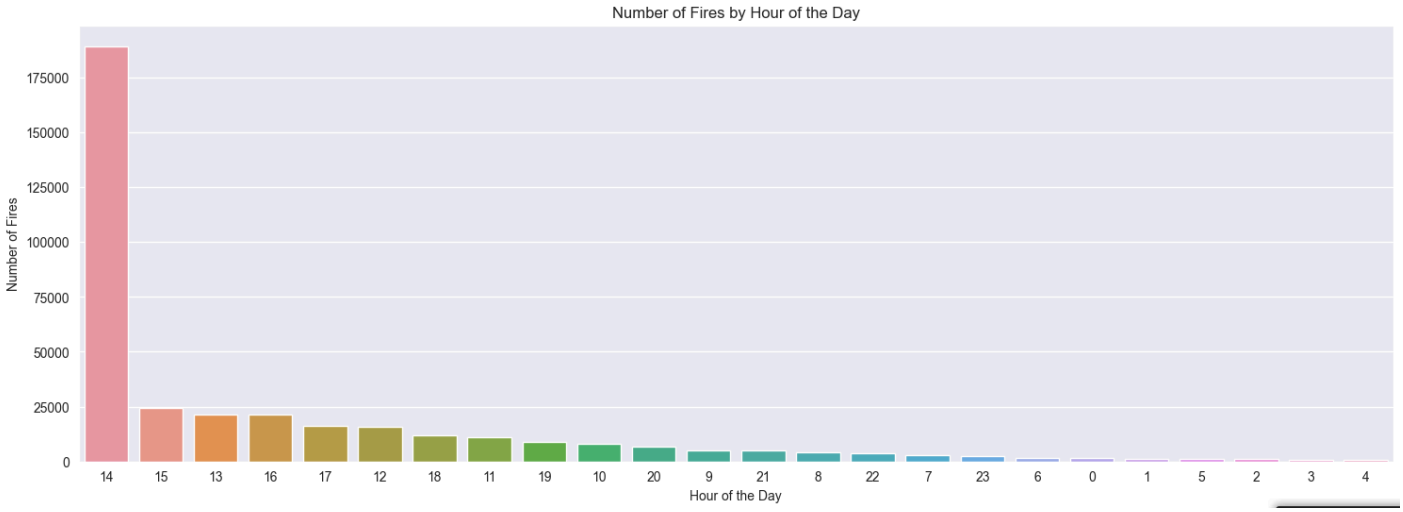
This column holds the period of the day in which the fire was discovered.
We divide a day into the following times:

- Late Night: 0-3
- Night: 4-7
- Morning: 8-11
- Noon: 12-15
- Afternoon: 16-19
- Evening: 20-23
-

In the following plot we can see the number of fires in each of the times:



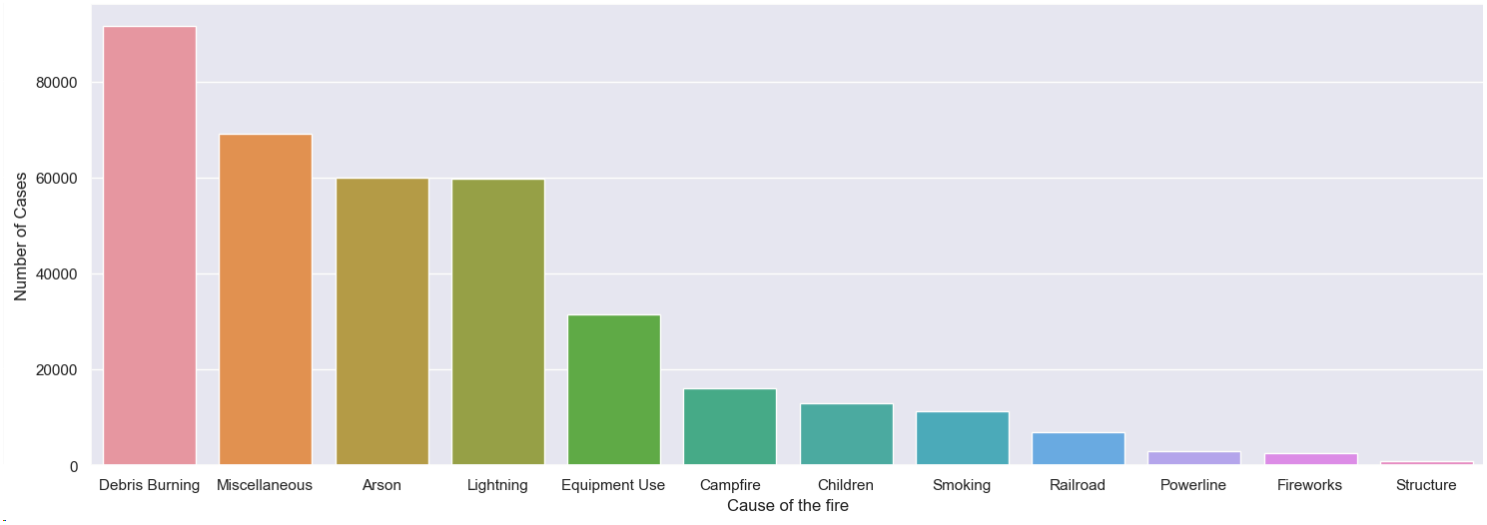
We can conclude that in most of the cases the fire occurred at noon.
We made another graph that shows what are the most common times for fires and found that most of the fires happened at two o'clock at noon.



Understanding Stat-Cause-Description column

STAT_CAUSE_DESCR distribution:
The STAT_CAUSE_DESCR column is our label column that we want to predict, so it is important to understand what the possible labels are, how many examples we have of each label in our training set and what other features can influence or be an indication of the label, etc.

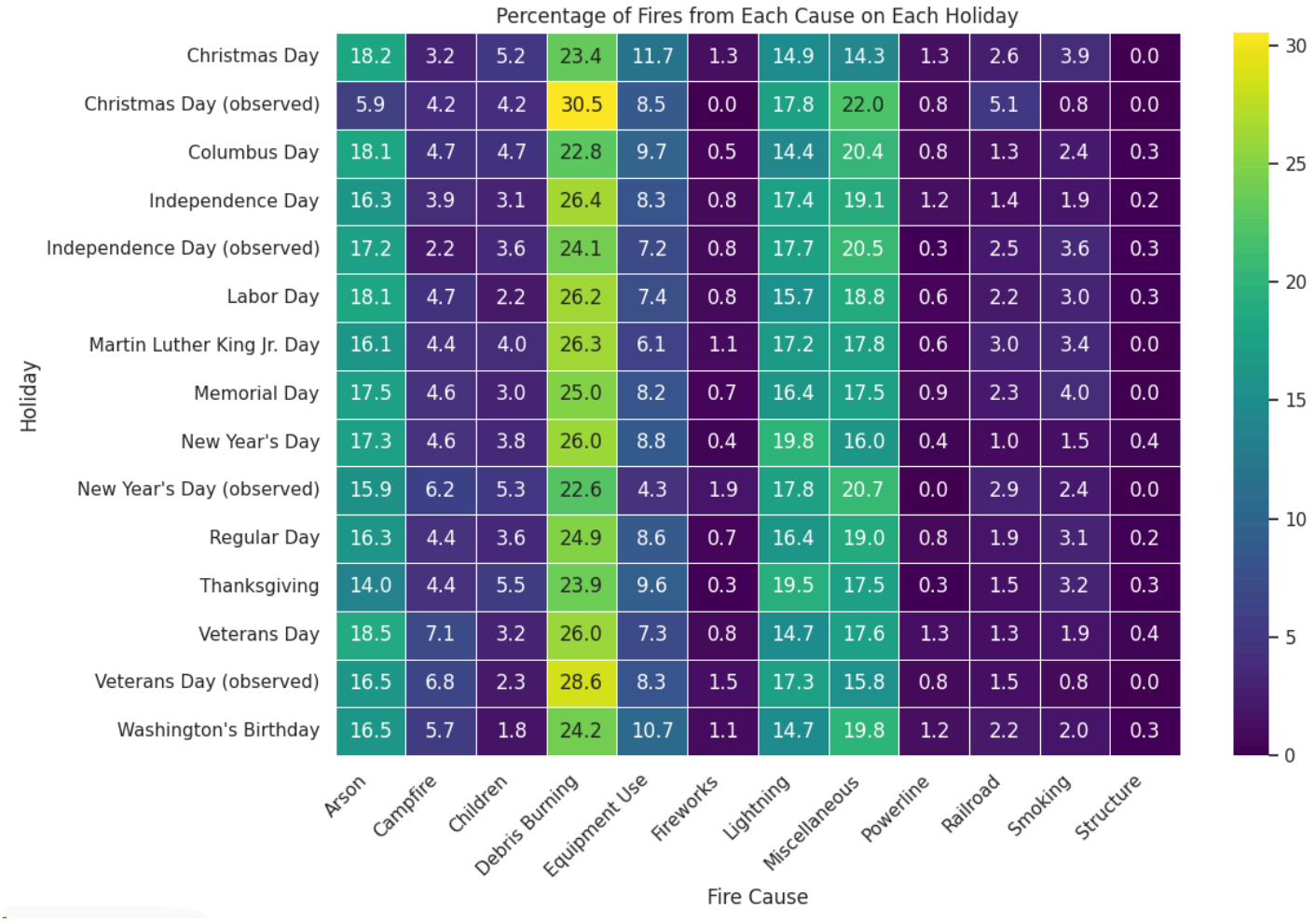
We checked which fire causes are more common in the training data and which fire causes are rare. We created this plot:



Holiday influence:

US holidays are times in which people can go out into the wild with their families, which might increase man made fires.
We used a holidays package to check which holiday was it on each fire instance.

In order to check whether there is a cause of fire that occurs more on certain holidays, we created the following graph that shows the percentage of cases for each combination of holiday and cause of fire, and in addition we added a representation for a normal day that is not a holiday

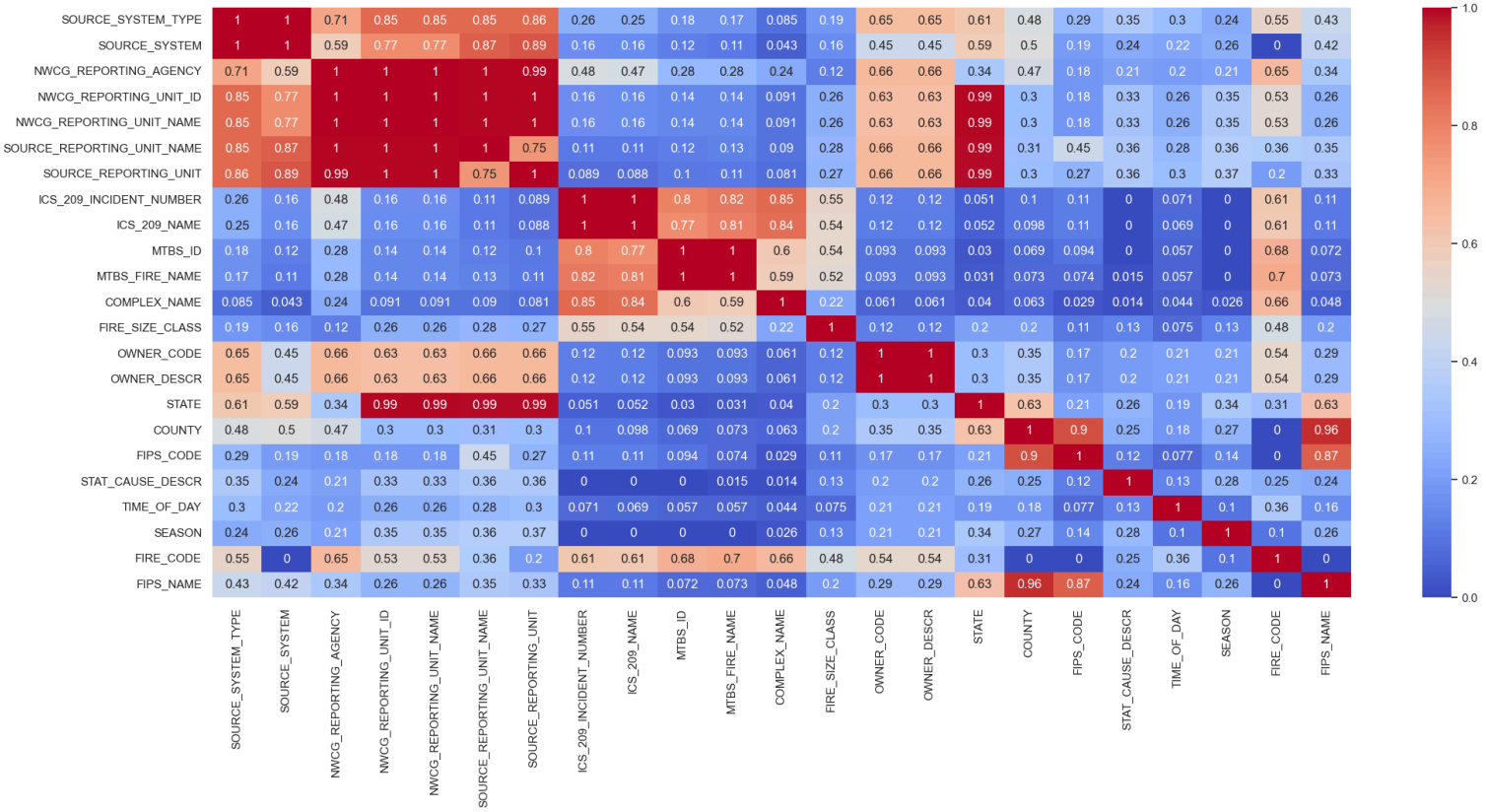


We can see from the graph that, contrary to what we thought, there are no holidays in which a certain type of fire is more common than the other holidays or compared to a normal day.

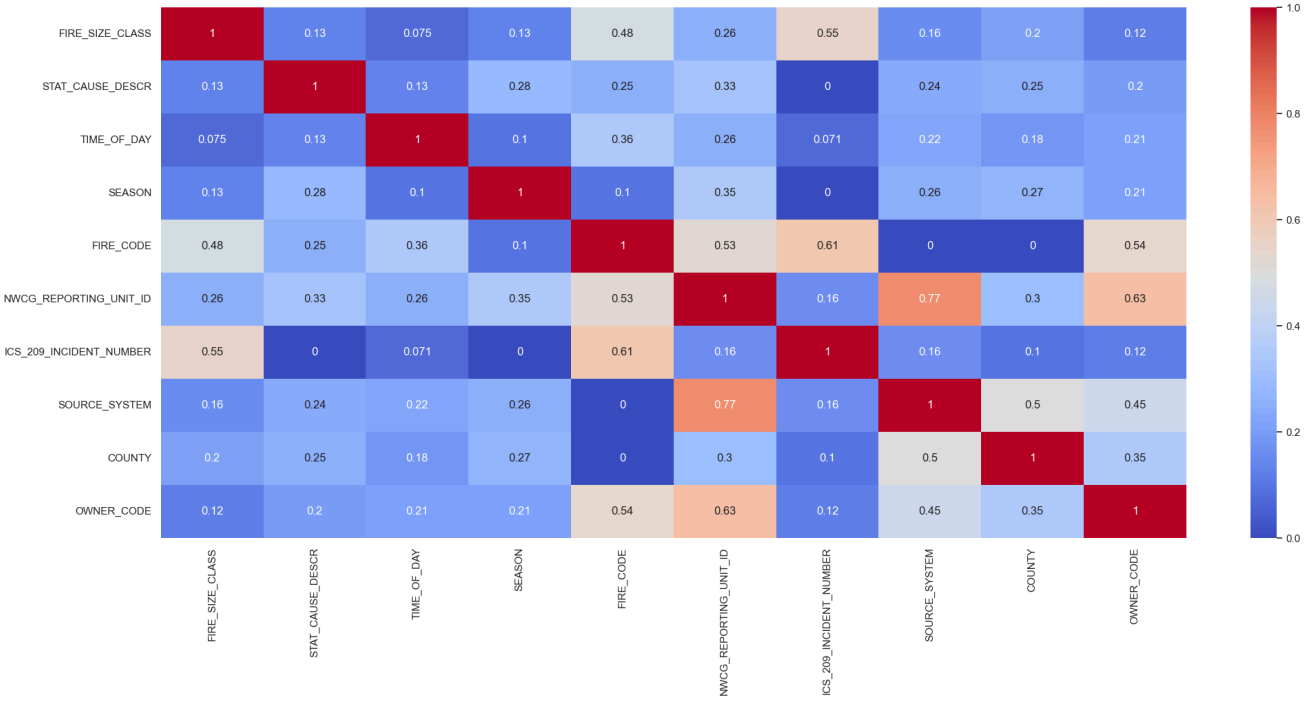
Feature Selection

In order to determine which features we should use in our model, we calculated the correlations of the columns, and removed columns that are highly correlated to others. This method makes our models simpler, and could improve our results.
To calculate the correlations, we divided our features into categorical and numerical features.

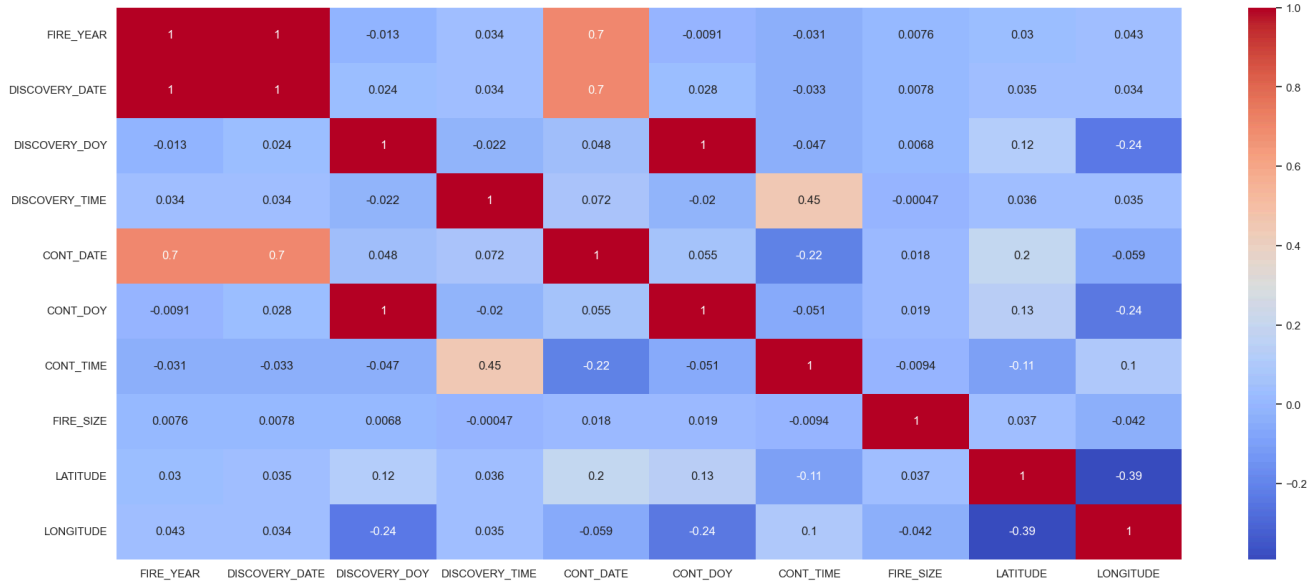
- **Correlation between categorical and categorica**l columns:



After this calculation, we then decided to remove features that had a correlation higher or equal to 0.79.
We then checked the correlations of the remaining columns and got:



- **Correlation between numerical and numerical** columns:



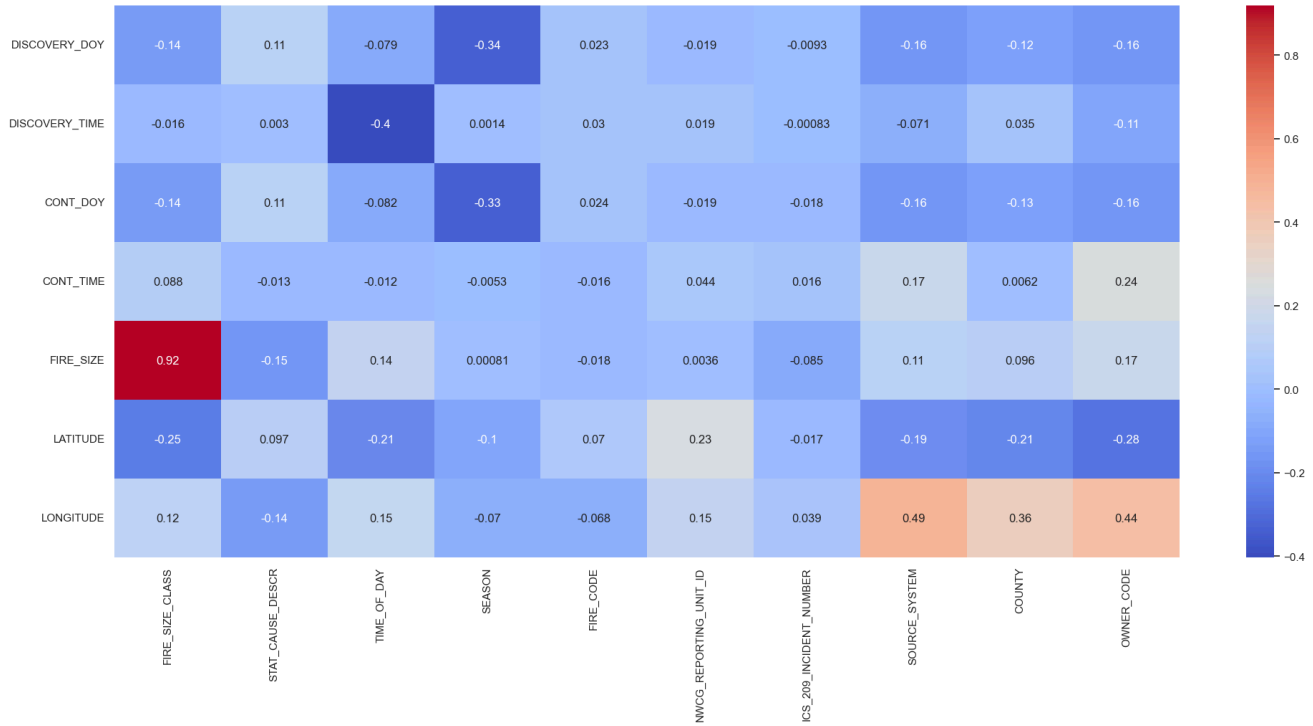
We can see that the only columns with correlation higher than 0.8 are:
FIRE_YEAR, DISCOVERY_DATE, CONT_DATE.

We got here a triangle of correlation between the 3 columns and the correlation of each pair is 1, so we can drop two of them and keep only one.

DISCOVERY_DATE
/ \
CONT_DATE -- FIRE_YEAR

we choose to keep FIRE_YEAR and drop the two other columns.

- **Correlation between categorical and numerical** columns:



We can see that fire size class and fire size are highly correlated, we can drop one of them.
We choose to drop fire size class.

Evaluation

The primary metric for evaluating model performance was the ROC AUC score.
The formula for AUC_ROC score is:

$$\text{AUC-ROC} = \sum_{i=1}^{n-1} \frac{(TPR_i + TPR_{i+1})}{2} \times (FPR_{i+1} - FPR_i)$$

Such that:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

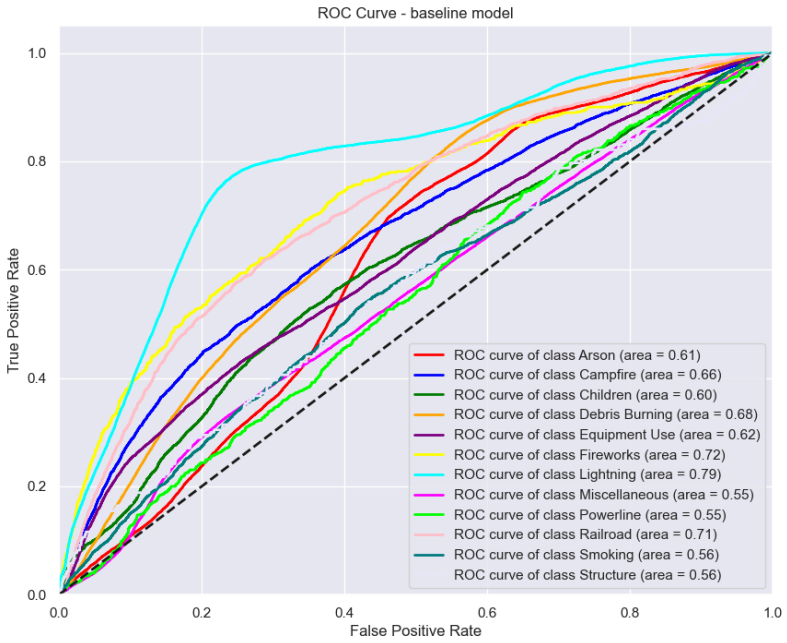
We also use accuracy metric, the accuracy of a classification model is a common metric used to evaluate its overall performance.
It measures the proportion of correctly predicted instances out of the total instances.
The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

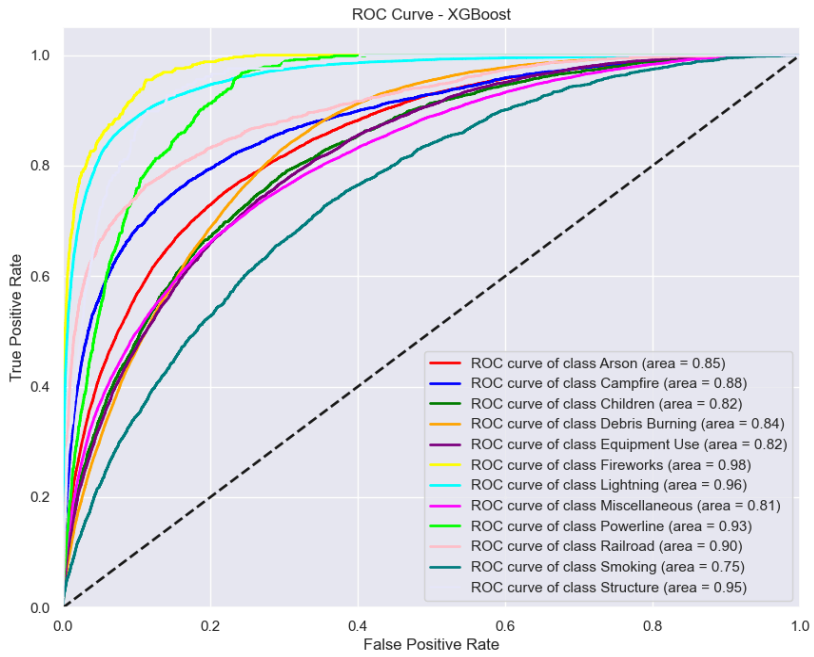
Model Selection

After conducting extensive trials we identified the best performing model.
The evaluation on the validation set yielded promising results.
We used the weighted ROC AUC score for our models, considering a one-vs-rest strategy for multi-class classification.
Starting our selection is the

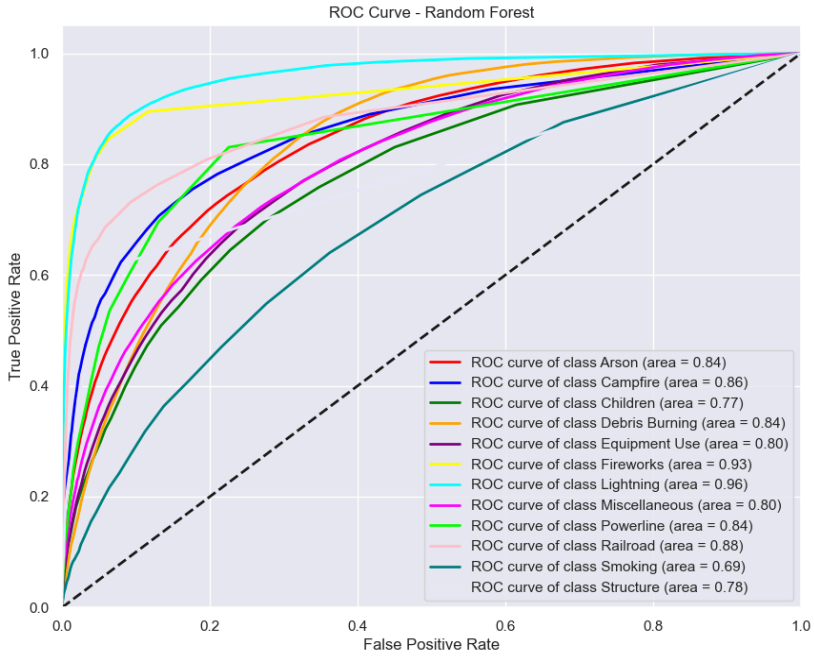
Baseline_model:
Using a baseline model that is guessing the class based on the distribution of the data yielded a result of 0.634 (weighted Roc Auc score on the validation set).
The model also got an accuracy result of 0.34.
So we made the effort to make a model that performed better than this baseline.



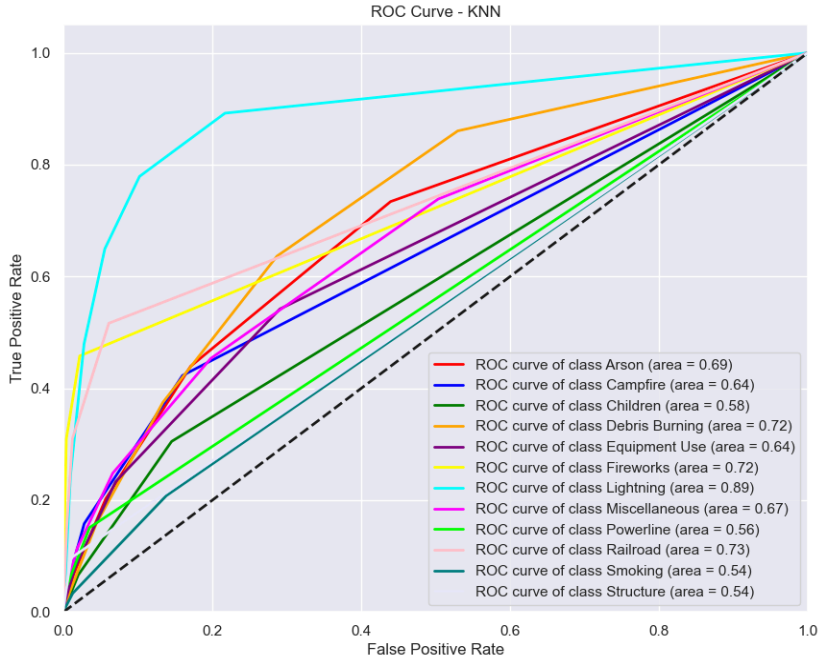
The result of the rest of the models (with default hyper parameters) are:
Best model: XGBoost
weighted ROC AUC score on the Validation set: 0.874
Accuracy score on the Validation set: 0.55



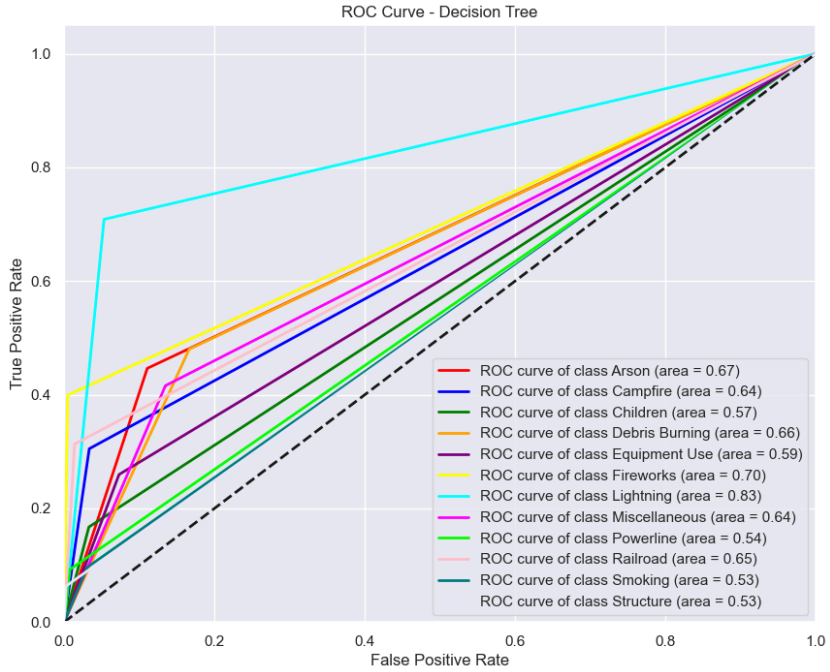
Model name: Random Forest.
weighted ROC AUC score on the Validation set: 0.836
Accuracy score on the Validation set: 0.55



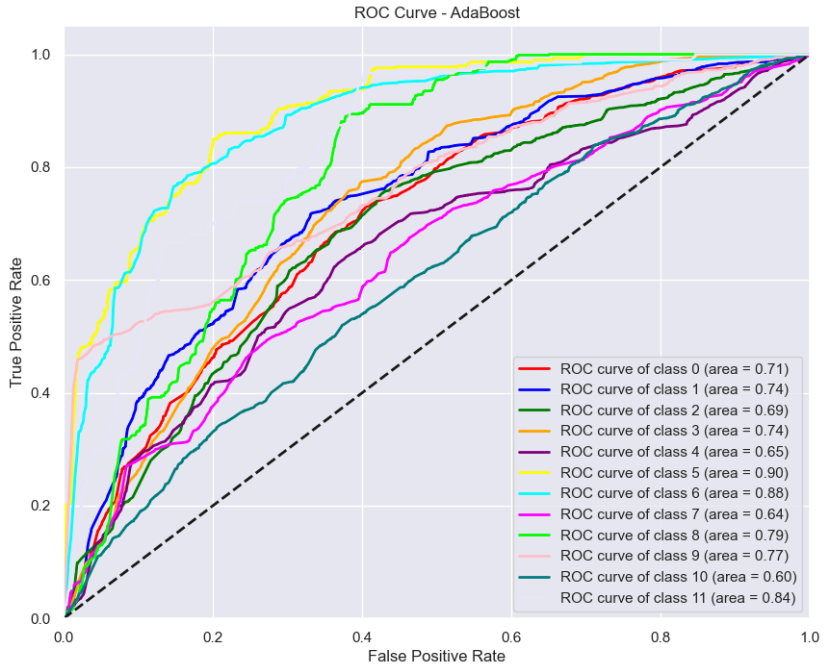
Model name: KNN.
weighted ROC AUC score on the Validation set: 0.660
Accuracy score on the Validation set: 0.41



Model name: Decision Tree.
weighted ROC AUC score on the Validation set: 0.629
Accuracy score on the Validation set: 0.44



Model name: Adaboost Decision Tree.
weighted ROC AUC score on the Validation set: 0.746
Accuracy score on the Validation set: 0.4



We have **decided to select the XGBoost model** since he had the best result.
This model is a very efficient model, and it uses regularized gradient boosting.
We can also see that it already outperforms the baseline model, in both score types.

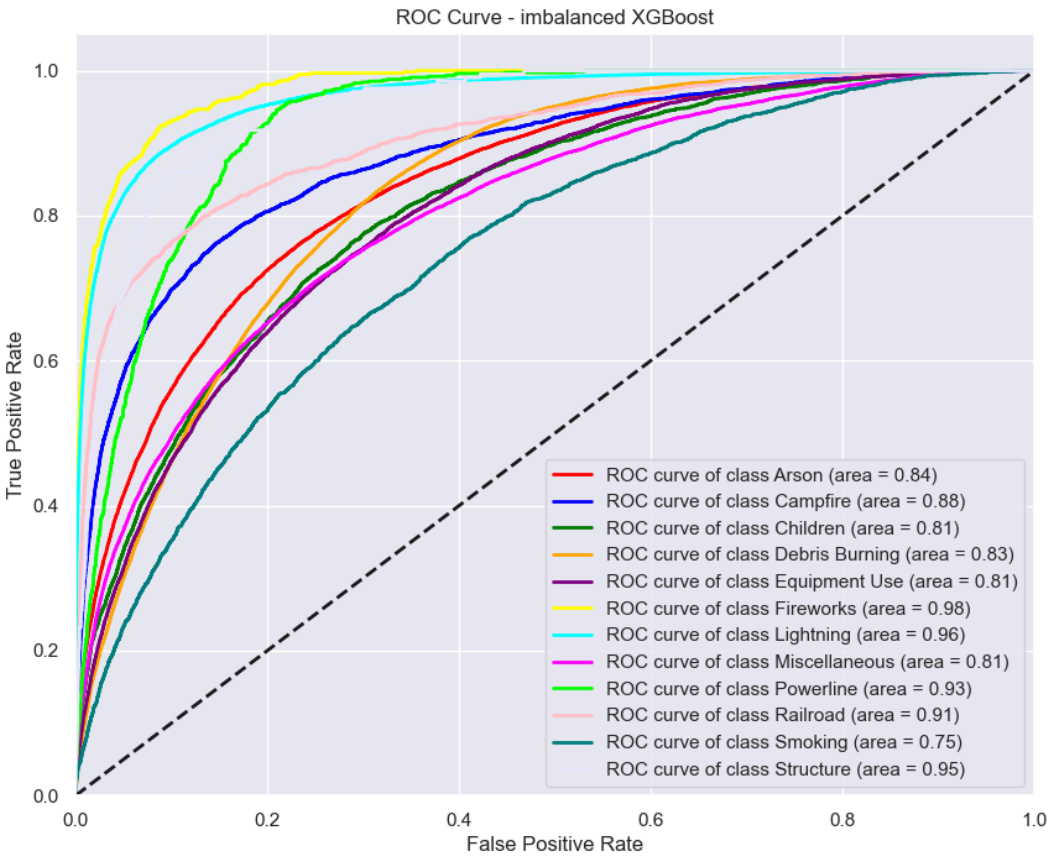
Imbalanced Data

As we saw at the beginning of the report, we do not have an equal distribution of the possible classes. Therefore, we checked whether addressing the unbalanced data would improve the predictive ability of the model.

We chose to perform the test on the XGBoost model which produced the best results. To handle the imbalance dataset, we tried to use weighted XGBoost, by setting the scale_pos_weight parameter according to the appearances number of each class.

We discovered that according to the AUC measure, the correction of the unbalanced data does not improve the score and even decreases it.
The result after adding the correction to Imbalanced data:

weighted ROC AUC score on the Validation set: 0.871



Data Scaling

We tried scaling our data in to see if it could improve our results and fix any scaling issues with the data. The result we got were:

Model name	**Extra Remarks**	**weighted ROC AUC score**
baseline	Default parameters	0.79
XGBoost	Default parameters	0.874
Random Forest	Default parameters	0.834
KNN	Default parameters	0.712
Decision Tree	Default parameters	0.627
Adaboost Decision Tree	Default parameters	0.746
XGBoost	Optimized Hyper Parameters	0.879

We can already see that except baseline, KNN, and Adaboost Decision Tree, the rest of our model yielded a slightly worse result and therefore we decided not to use data scaling technique.

Optimization

Hyper Parameters Tuning

After selecting the XGBoost model we wanted to optimize its hyper parameters, to improve our results.
In order to do that we used the optuna library, to check the result of multiple different hyperparameters.

The hyper parameters and ranges we checked are:

- n_estimators: 50-200
- max_depth: 3-10
- learning_rate: 0.1-1
- subsample: 0.5-1
- gamma: 0.01-1
- min_child_weight: 1-10

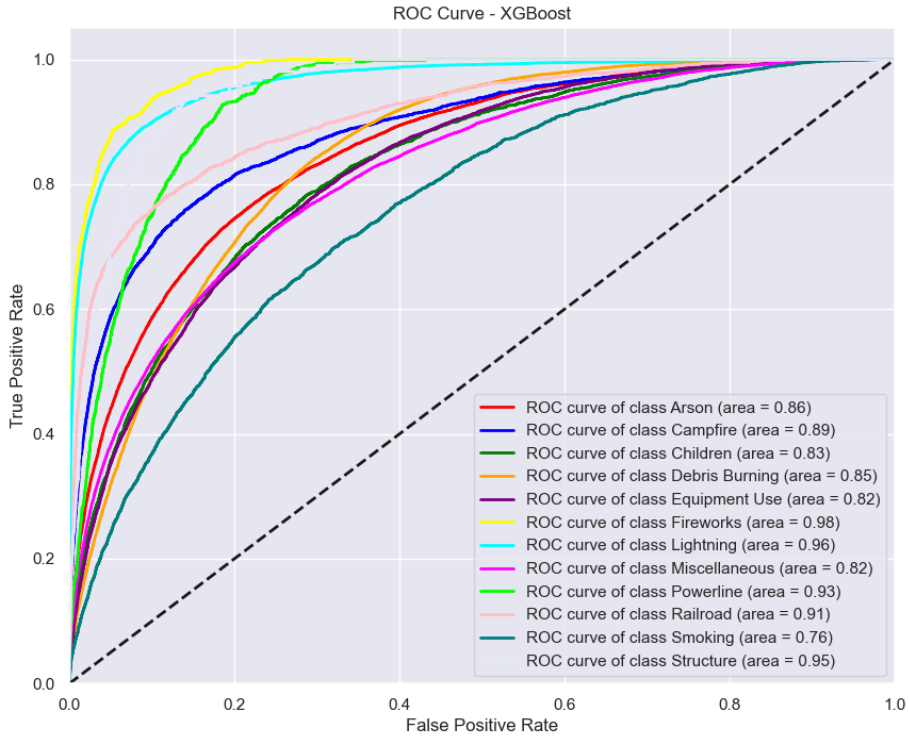
We used K-fold-cross-validation with k=3.

The selected hyper parameters are:

- n_estimators: 156, 'max_depth': 9
- learning_rate: 0.11236596365407153
- subsample: 0.8925920206669298
- gamma: 0.0605871121329831
- min_child_weigh: 5

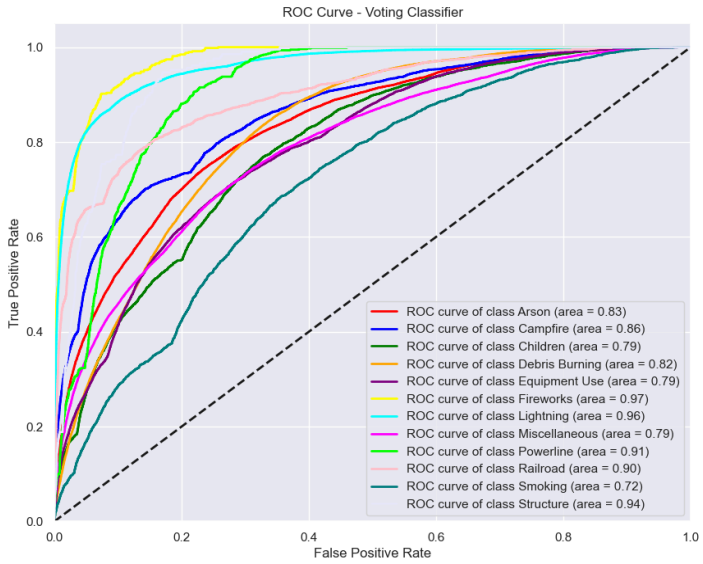
Using the best hyperparameters we managed to increase the result of the model on the validation set by approximately 0.06.

AUC score on the Validation set: 0.880
Accuracy score on the Validation set: 0.56

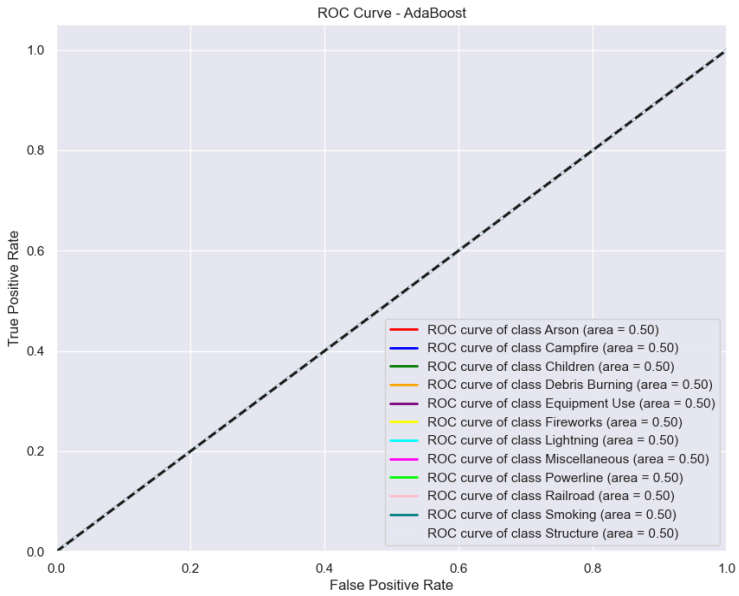


Ensemble & Boosting

After tuning the Hyper Parameters we tried improving the model using Ensemble and Boosting methods.
Both methods did not improve the performance of our XGBoost model, so we decided not to use them in our final model.
The results are as follow:
Ensemble method:
Model name: Voting Classifier - with XGBoost, Random Forest, Decision Tree and KNN estimators.
weighted ROC AUC score on the Validation set: 0.857



Boosting Method:
Model name: Adaboost with XGBoost estimator, “SAMME” algorithm.
AUC score on the Validation set: 0.5



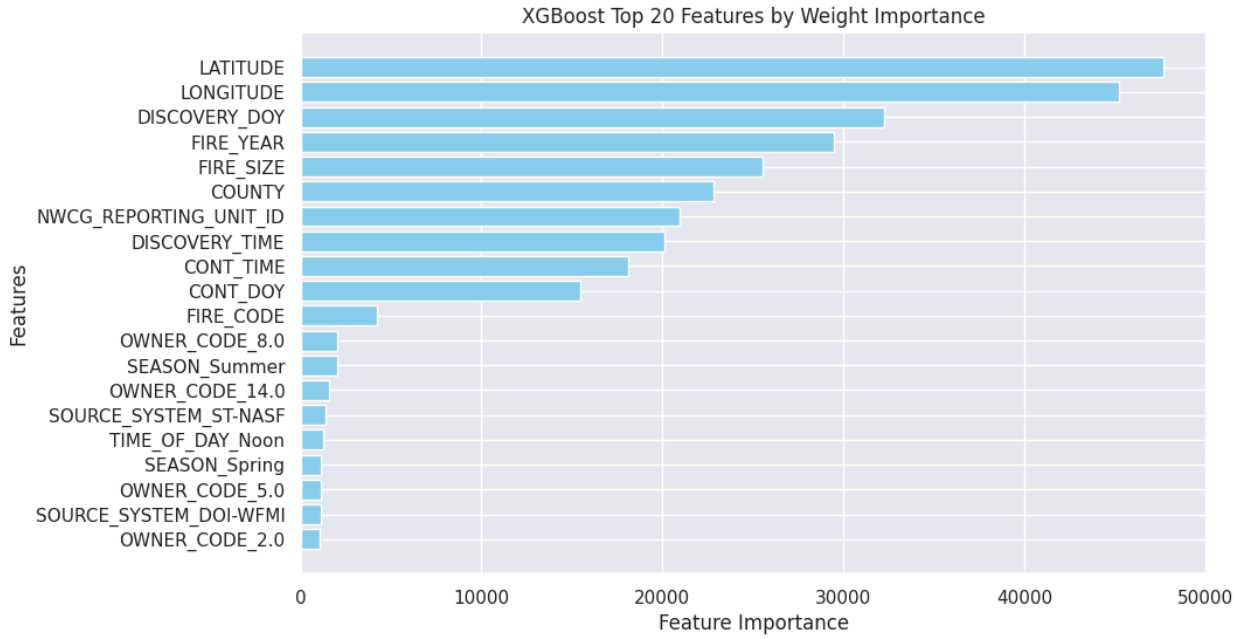
Another Optimization Attempt

We wanted to make sure that during the process we didn't remove any important columns and "missed" the possible data they can provide to the learning process.
In order to do that we tried adding each of the removed columns separately to our data and observe how it affected the model performance.
The results:

- SOURCE_SYSTEM_TYPE: 0.8798121472050408
- NWCG_REPORTING_AGENCY: 0.8797980843796034
- NWCG_REPORTING_UNIT_NAME: 0.8792178609538528
- SOURCE_REPORTING_UNIT: 0.879023169157814
- SOURCE_REPORTING_UNIT_NAME: 0.8790354737568675
- ICS_209_NAME: 0.8799874477686096
- MTBS_ID: 0.8800675051597792
- MTBS_FIRE_NAME: 0.8798996578008934
- COMPLEX_NAME: 0.880069644404383
- DISCOVERY_DATE: 0.8791358394885274
- CONT_DATE: 0.8792015837272413
- FIRE_SIZE_CLASS: 0.8795507065448707
- OWNER_DESCR: 0.8797577469301995
- STATE: 0.8796116500288952
- FIPS_CODE: 0.8791539755781965
- FIPS_NAME: 0.8792826125451421

As we can see, the changes in the result, compared to the original best performing model are very minor.
There are very few features that increased the result by approximately 0.00006, and the rest yielded a worst result.
Because the increase in score is so minor we decided not to add any of the removed features back, to keep our model as simple as possible.

Feature Evaluation



By looking at our best performing model we tried to understand which features ended up being important and which seemed mostly redundant.

By assessing our winning XGBoost model on only the top 10 features (sorted based on their weight in the given model) we were able to discover that a model that was learning on the training set with only the top 10 features could achieve a similar result to the previous result (in which we trained our model on all of our columns).
The results:

```
AUC for model trained with top 10 features by gain: 0.7238531630854497
AUC for model trained with top 10 features by weight: 0.8747828880464161
AUC for model trained with top 10 features by cover: 0.5333646259961162
```

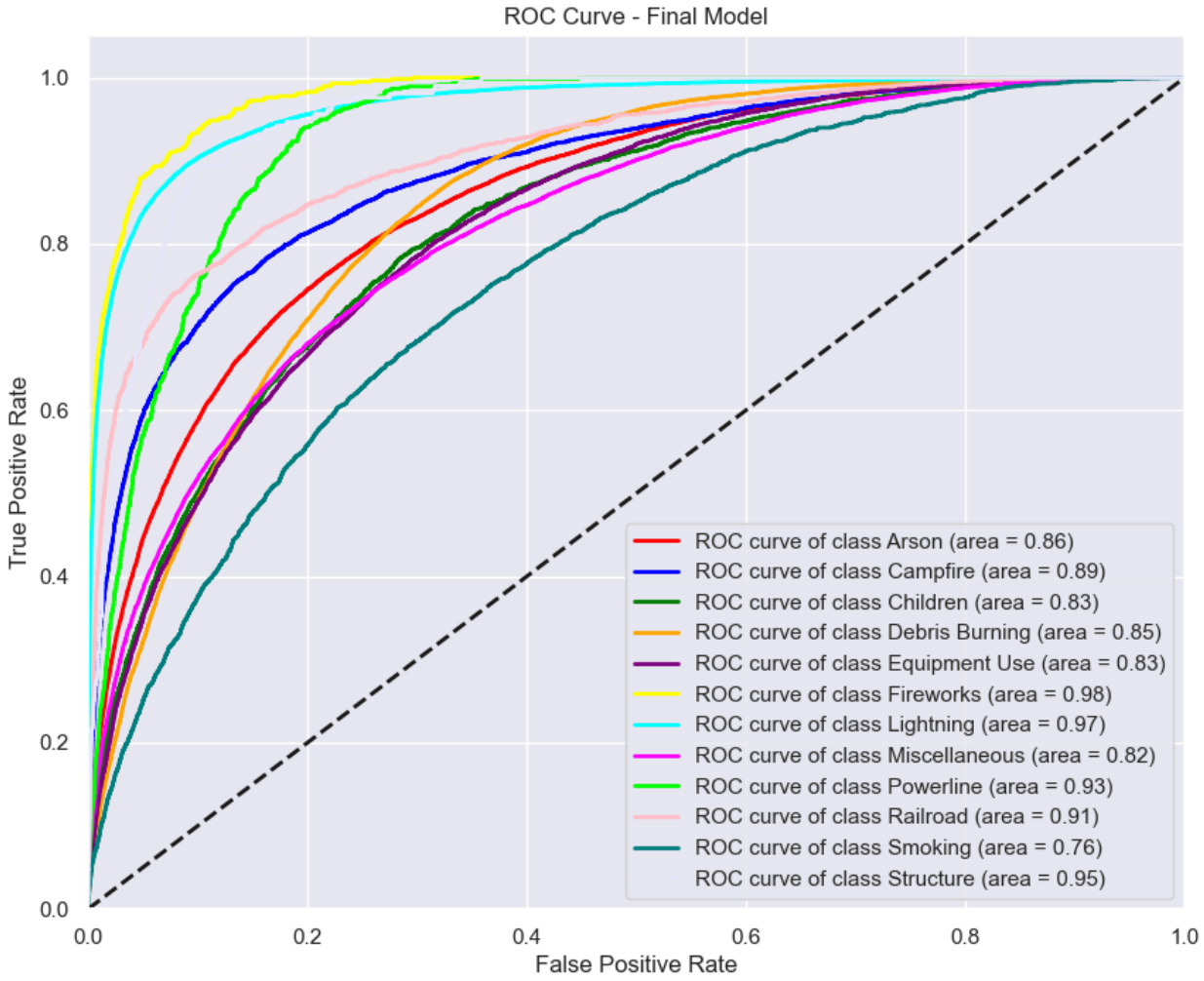
We tried similar values of the number of features as a hyperparameter and found that roughly 40 features got the best results.

```
AUC for model trained with top 10 features by weight with 60 features: 0.8801960513077159
AUC for model trained with top 10 features by weight with 50 features: 0.8809149967580487
AUC for model trained with top 10 features by weight with 40 features: 0.8810023276881158
AUC for model trained with top 10 features by weight with 30 features: 0.8805597576181617
AUC for model trained with top 10 features by weight with 20 features: 0.8803687912184306
```

We decided to use the top 40 features (remove the less important features such that only 40 features will remain) in our final model, since the result using those features was the best (improved a little), and it will also make our model less complex.

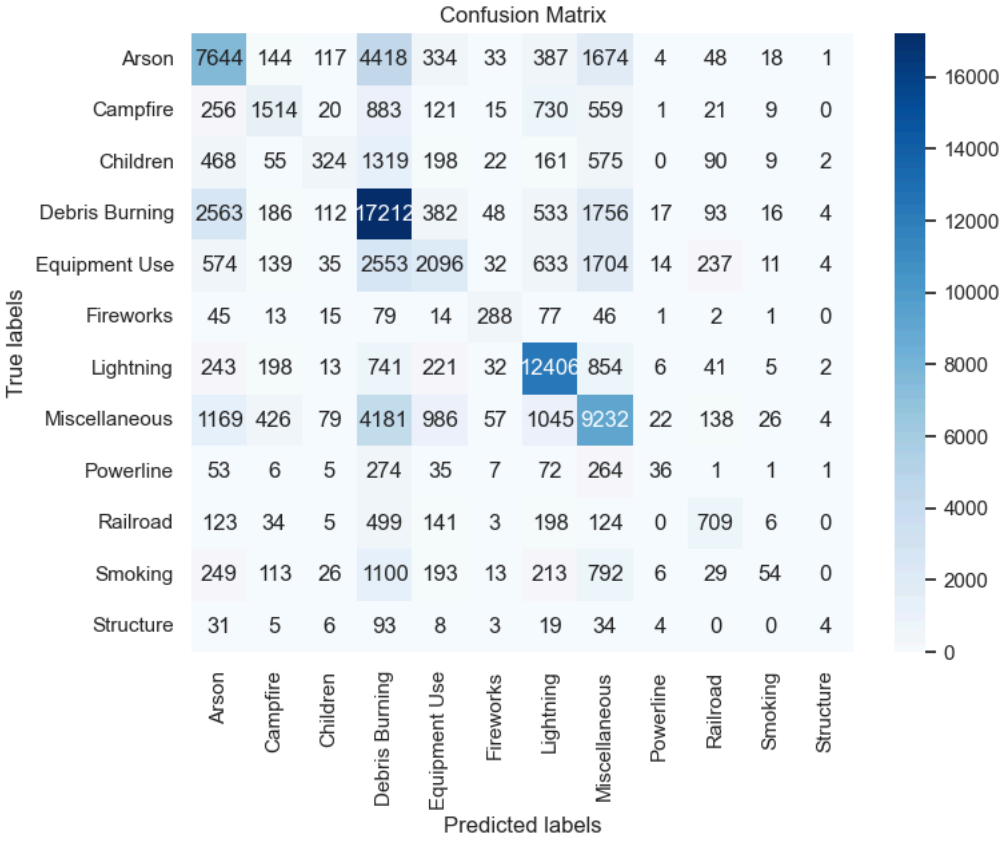
Final Model

Using the top 40 most important features, and the optimized XGBoost model, we get our final model with the results:
AUC score on the Validation set: 0.881
Accuracy score on the Validation set: 0.56



Error Analysis

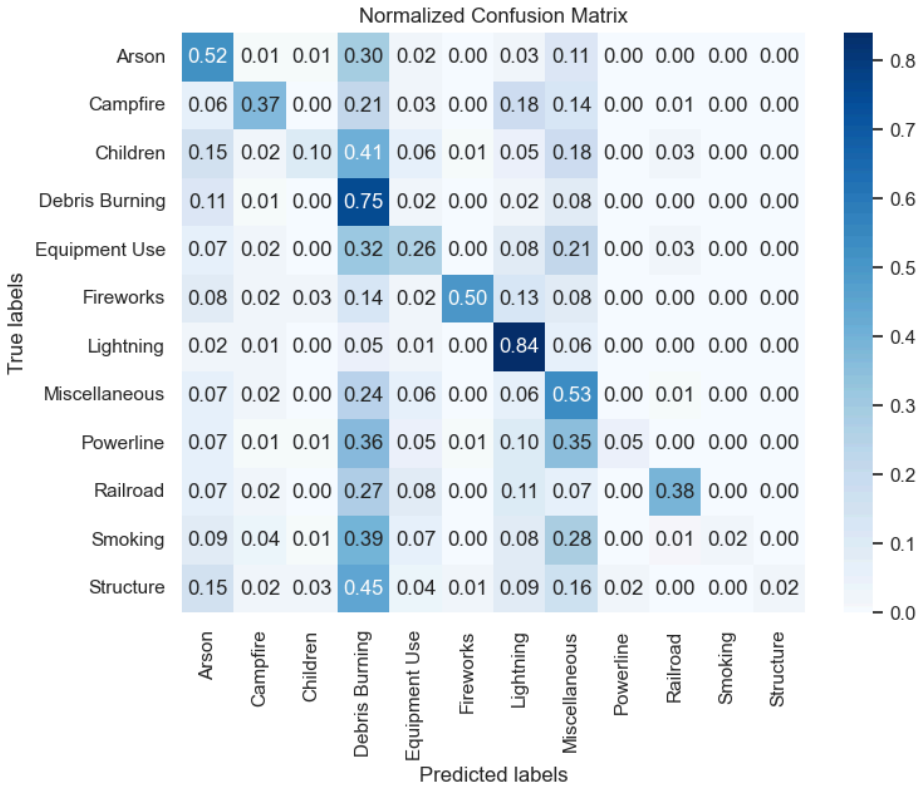
We noticed that in all the ROC graph we did, for all the models we checked
The class “Smoking” got the lowest ROC-AUC value in relation to the other labels.
We create a confusion matrix in order to better understand what the model predicts when the real label is smoking, and for comparison between actual vs predicted values for all the classes.



As we can see in the confusion matrix most of the times when the true label was Smoking, the model predicted Debris Burning.
We assume that the cause of this is that most of the data that the model was trained on is tagged with the label Debris Burning.

The values are high In the diagonal cells of the matrix, So We can conclude that there were many times that the prediction was successful.

We also created a confusion matrix such that cell (i,j) represents the percentage of cases when the predicted label was j in relation to the sum of all cases with true label i.



We can see that the model predicts these two classes Miscellaneous and Debris burning a lot even though it is not true.
This means that the model does not always know how to generalize well examples with labels that are not these two labels.

As we saw before the features that had the most effect on the prediction are latitude and longitude which represents a specific place.
And that's why we assume that another reason why the prediction for smoking is less successful is that compared to other causes of fire, smoking has a weaker connection to a specific place. Smoking is something that is relatively common all over the world and less dependent on the geographical area compared to other causes of fire.

For example, lightning, there are areas in the world where winter weather and lightning are more common therefore if the model receives information about the area, it is easier for him to predict lightning if the location is suitable for it.
Another example is fireworks, there are areas in the world where it is more customary to celebrate in the streets and celebrate with fireworks.

Results

Validation Set Results

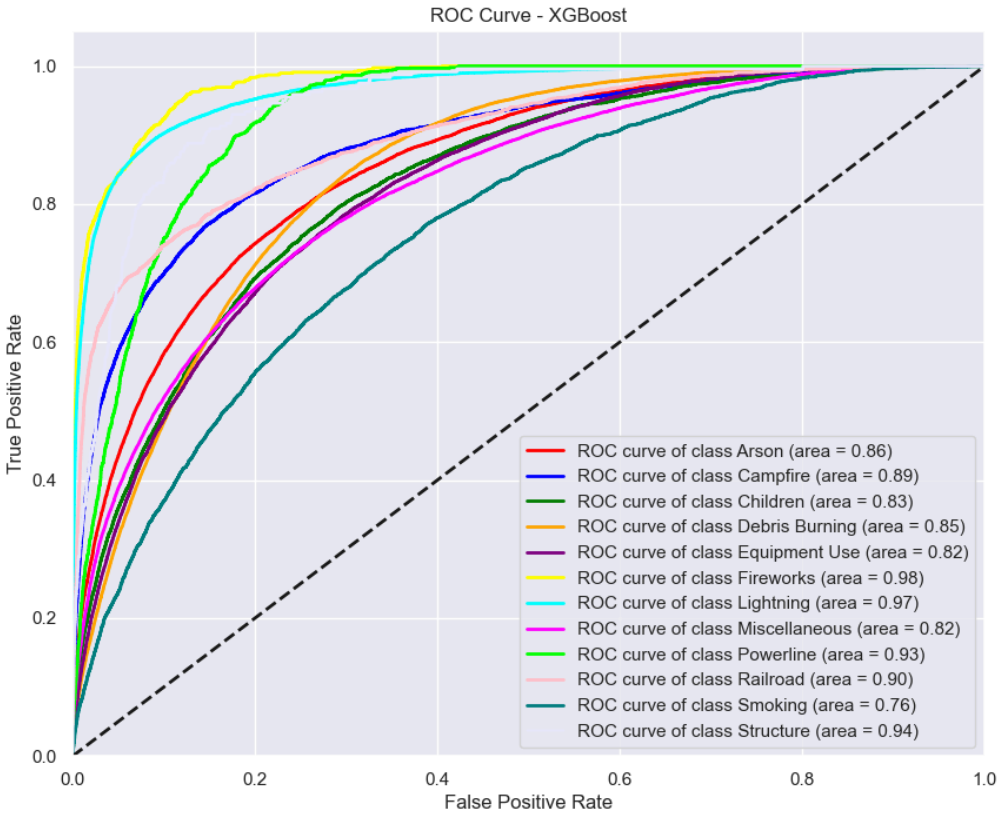
For convenience we organized all the results over the validation set in a table:

Model name	**Extra Remarks**	**weighted ROC AUC score**	**Accuracy score**
baseline	Default parameters	0.634	0.34
XGBoost	Default parameters	0.874	0.55
Random Forest	Default parameters	0.836	0.55
KNN	Default parameters	0.660	0.41
Decision Tree	Default parameters	0.629	0.44
Adaboost Decision Tree	Default parameters	0.746	0.4
XGBoost	Imbalanced Data	0.871	
XGBoost	Optimized Hyper Parameters	0.880	0.56
Voting Classifier	Ensamble	0.857	
Adaboost with XGBoost	Boosting	0.5	
XGBoost	Final model fitted over the top 40 features	0.881	0.56

Test Results

Eventually we ran our model **once** to predict the result of the test set (during the whole work process we didn't look or touch the test in order to not contaminate it).
The results are as follow:

weighted ROC AUC score on Test set: 0.879



The resulting score is very close to the result of the validation set, which is a good sign for our model. it means that it performed relatively well even on a data set that wasn't "seen" during the process.