Forest Fire Forecasting

(Final Report)

Gilmo Yang
Data Science
CU Boulder
Boulder, CO, USA
giya5789@colorado.edu

Mahin Kadabi
Data Science
CU Boulder
Boulder, CO, USA
maka1640@colorado.edu

Nikhil Gupta
Data Science
CU Boulder
Boulder, CO, USA
nigu1976@colorado.edu

Vidit Vivek Sharma
Data Science
CU Boulder
Boulder, CO, USA
vish4271@colorado.edu

ABSTRACT

Forest fires are a major environmental issue, creating economic and ecological damage while endangering human lives. Predictive Services could provide decision-support information to the Forest Service and other federal, tribal, state, and local wildland fire management agencies for operational management of and strategic planning for wildland firefighting assets, such as firefighters, aircraft, and engines. The accurate prediction of forest fire propagation is a crucial issue to minimize its effects.

Forest fires importantly influence our environment and lives. The ability of accurately predicting them that may be involved in a forest fire event may help in optimizing fire management efforts. Given the complexity of the task, powerful computational tools are needed for predicting the amount of area that will be burned during a forest fire. This methodology was based on Genetic Algorithms which require the execution of many simulations. Moreover, when the fire is large some input parameters cannot be considered uniform among the whole fire and extra models must be introduced. This model implies more computation time and response time is the main constraint. The prediction must be provided as fast as possible to be useful, thus it is necessary to exploit all available computing resources.

The number of catastrophic wildfires in the U.S. has been steadily rising. As per an official report by Congressional Research Service (CRS) from 2011 to 2020, an average of 62,805 wildfires annually average of 7.5million acres impacted annually. In 2020, 58,950 wildfires burned 10.1 million acres. These forest fires can be man-made or caused by mother nature by different weather conditions, torrential winds.

KEYWORDS

Machine Learning Application, Forest Fire Prediction, Random Forest Regressor, Support Vector Regression, Decision Tree.

1. INTRODUCTION

The first question is why we need machine learning to anticipate forest fires in that specific area in the first place. The question is valid: why is there a need for Machine Learning when there is an experienced forest department that has been dealing with these issues for a long time? The answer is simple: the experienced forest department can check on 3-4 parameters from their human mind, whereas Machine Learning, on the other hand, can handle numerous parameters such as latitude, longitude, satellite, version, and so on, so dealing with this multi-relationship of a param requires Machine Learning. We will also use data mining. Data mining is the process of finding anomalies, patterns and correlations within large data sets to predict outcomes. Various techniques can be used in this information to increase revenues, cut costs, improve customer relationships, and reduce risks and more.

The main motivation of this project is to predict forest fires so that we can optimize fire management efforts and save earth and its inhabitants. The main challenge for this project is to deal with numerous parameters such as latitude, longitude, satellite etc. for forecasting the forest fires. This forecasting model also requires more computation time and response time which makes it an interesting task for this project.

1.1 Causes of Forest Fire

1. Natural causes - Many forest fires start from natural causes such as lightning which set trees on fire. However, rain extinguishes such fires without causing much damage. High atmospheric temperatures and dryness (low humidity) offer favorable circumstance for a fire to start.

2. Man-made causes - Fire is caused when a source of fire like naked flame, cigarette or bidi, electric spark or any source of ignition comes into contact with inflammable material.

1.2 Types of Forest Fire

1. Surface Fire-A Forest fire which primarily burn as a surface fire, spreads along the ground as the surface litter (senescent leaves and twigs and dry grasses etc.) on the forest floor and is engulfed by the spreading flames.

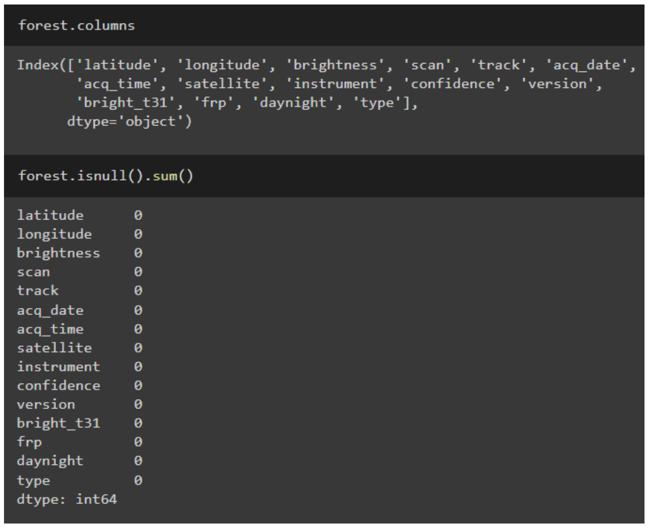
2. Underground Fire - The fires of low intensity, consuming the organic matter beneath and the surface litter of forest floor are sub-grouped as underground fire. In most of the dense forests a thick mantle of organic matter is find on top of the mineral soil. This fire spreads in by consuming such materials. The other terminology for this type of fire is Muck fires.

3. Firestorms - Among the forest fires, the fire spreading most rapidly is the firestorm, which is an intense fire over a large area. As the fire burns, heat rises and air rushes in, causing the fire to grow. More air makes the fire spin violently like a storm.

2. DATA GATHERING AND PREPROCESSING

We gathered past years data from the Fire Information for Resource Management System. The data is available on https://firms.modaps.eosdis.nasa.gov/. NASA FIRMS uses satellite observations from the MODIS and VIIRS instruments to detect active fires and thermal anomalies and deliver this information in near real-time to decision makers through email alerts, analysis ready data, online maps and web services.

After selecting the dataset, the next step is to preprocess the dataset and then with data cleaning. While preprocessing the data set, we checked if any of the columns had null values in them. Fortunately, that was not the case. From the above data, we can see that some columns have just one value recurring in them, meaning they are not valuable to us. So, we dropped them altogether.



```
forest.columns
```

```
Index(['latitude', 'longitude', 'brightness', 'scan', 'track', 'acq_date',
       'acq_time', 'satellite', 'instrument', 'confidence', 'version',
       'bright_t31', 'frp', 'daynight', 'type'],
      dtype='object')
```

```
forest.isnull().sum()
```

```
latitude      0
longitude     0
brightness    0
scan          0
track         0
acq_date      0
acq_time      0
satellite     0
instrument    0
confidence    0
version       0
bright_t31    0
frp           0
daynight      0
type          0
dtype: int64
```

Figure 1: Data preprocessing

While cleaning the data, we found correlation between two columns.

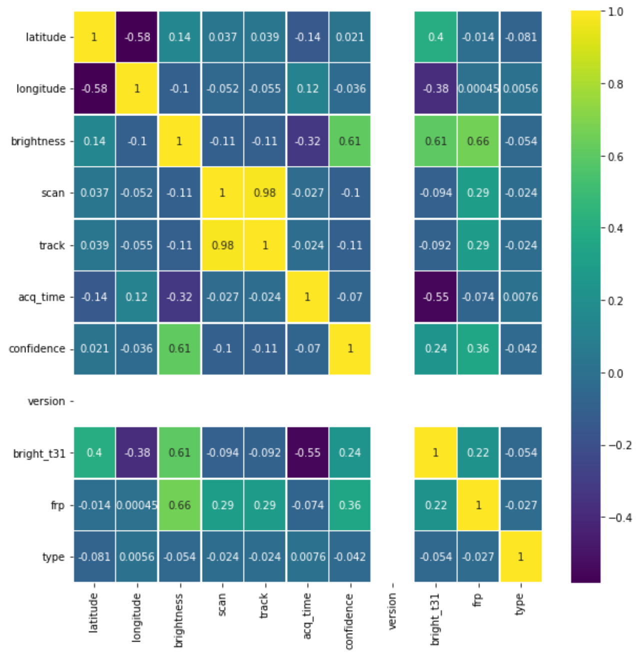
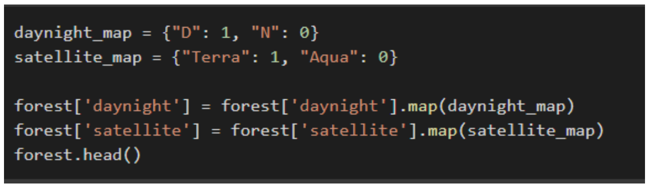


Figure 2: Unscaled Data Correlation Heatmap.

From the data, we can see that some columns have just one value recurring in them, meaning they are not valuable to us. So we will drop them altogether. Thus, only satellite and day-night columns are the only categorical type, so we had to scale data accordingly.



```
daynight_map = {"D": 1, "N": 0}
satellite_map = {"Terra": 1, "Aqua": 0}

forest['daynight'] = forest['daynight'].map(daynight_map)
forest['satellite'] = forest['satellite'].map(satellite_map)
forest.head()
```

Figure 3: Handling Categorical Data

After dropping the unnecessary columns and scaling the data, we will check to see if the correlation is removed or not.

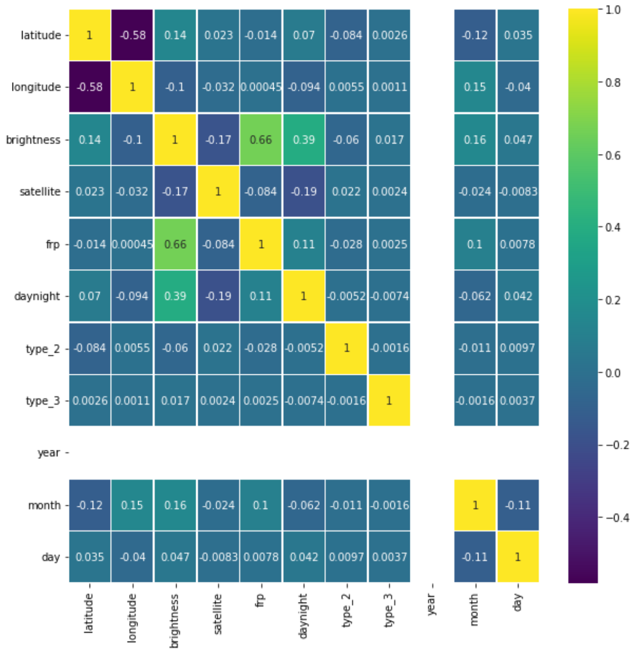
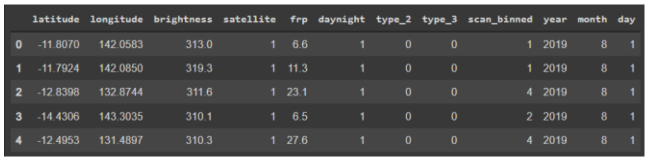


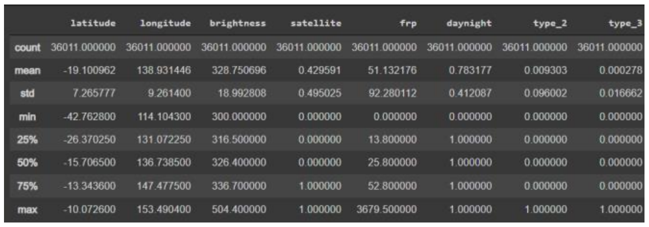
Figure 4: Scaled Data Correlation Heatmap.

Once the data is cleaned, it is ready to be utilized for splitting the data and making a predictive model for forest fire forecasting.



	latitude	longitude	brightness	satellite	frp	daynight	type_2	type_3	scan_binned	year	month	day
0	-11.8070	142.0583	313.0	1	6.6	1	0	0	1	2019	8	1
1	-11.7924	142.0850	319.3	1	11.3	1	0	0	1	2019	8	1
2	-12.8398	132.8744	311.6	1	23.1	1	0	0	4	2019	8	1
3	-14.4306	143.3035	310.1	1	6.5	1	0	0	2	2019	8	1
4	-12.4953	131.4897	310.3	1	27.6	1	0	0	4	2019	8	1

Figure 5: Cleansed Data at a Glance



	latitude	longitude	brightness	satellite	frp	daynight	type_2	type_3
count	36011.000000	36011.000000	36011.000000	36011.000000	36011.000000	36011.000000	36011.000000	36011.000000
mean	-19.100962	138.931446	328.750696	0.429591	51.132176	0.783177	0.009303	0.000278
std	7.265777	9.261400	18.992808	0.495025	92.280112	0.412087	0.096002	0.016662
min	-42.762800	114.104300	300.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	-26.370250	131.072250	316.500000	0.000000	13.800000	1.000000	0.000000	0.000000
50%	-15.706500	136.738500	326.400000	0.000000	25.800000	1.000000	0.000000	0.000000
75%	-13.343600	147.477500	336.700000	1.000000	52.800000	1.000000	0.000000	0.000000
max	-10.072600	153.490400	504.400000	1.000000	3679.500000	1.000000	1.000000	1.000000

Figure 6: Cleansed Data Attributes

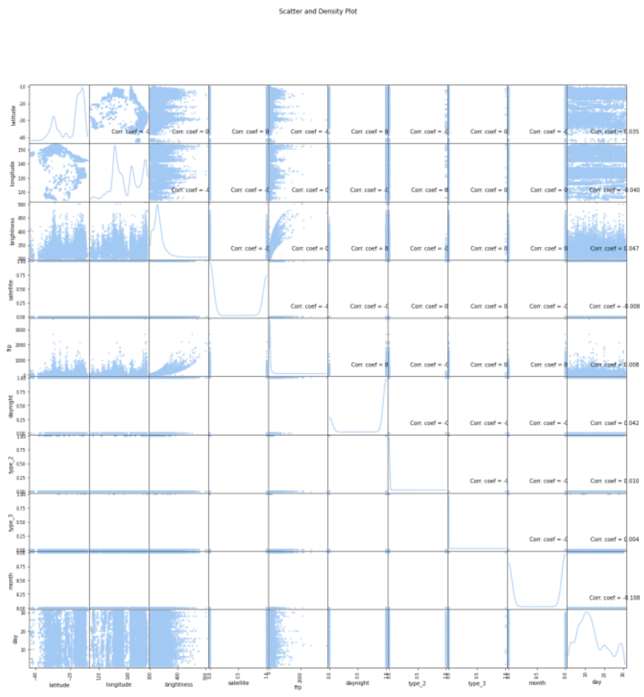
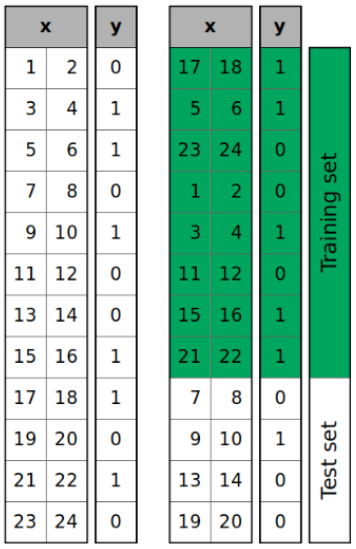


Figure 7: Scatter and Density Plot

3. SPLITTING THE CLEAN DATA INTO TRAINING AND TESTING DATASET

Supervised machine learning is about creating models that precisely map the given inputs (independent variables, or predictors) to the given outputs (dependent variables, or responses). How you measure the precision of your model depends on the type of a problem you're trying to solve. What's most important to understand is that you usually need unbiased evaluation to properly use these measures, assess the predictive performance of your model, and validate the model.

This means that you can't evaluate the predictive performance of a model with the same data you used for training. You need evaluate the model with fresh data that hasn't been seen by the model before. You can accomplish that by splitting your dataset before you use it. Splitting a dataset might also be important for detecting if your model suffers from one of two very common problems, called underfitting and overfitting.

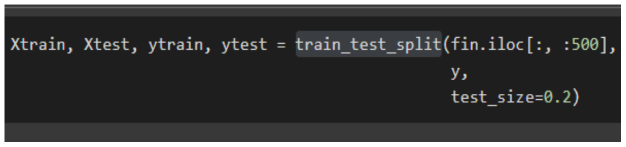


x		y
1	2	0
3	4	1
5	6	1
7	8	0
9	10	1
11	12	0
13	14	0
15	16	1
17	18	1
19	20	0
21	22	1
23	24	0

x		y	
17	18	1	Training set
5	6	1	
23	24	0	
1	2	0	
3	4	1	
11	12	0	
15	16	1	
21	22	1	
7	8	0	Test set
9	10	1	
13	14	0	
19	20	0	

Figure 8: Example of Splitting the data

Here we have used 'train_test_split'. It is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually. By default, Sklearn train_test_split will make random partitions for the two subsets.



```
Xtrain, Xtest, ytrain, ytest = train_test_split(fin.iloc[:, :500],
                                                y,
                                                test_size=0.2)
```

Figure 9: Code of Splitting the data

4. PREDICTIVE MODEL LEARNING

Predictive modeling is the process of using known results to create, process, and validate a model that can be used to forecast future outcomes. It is a tool used in predictive analytics, a data mining technique that attempts to answer the question "what might possibly happen in the future?"

Predictive modeling is a statistical technique using machine learning and data mining to predict and forecast likely future outcomes with the aid of historical and existing data. It works by analyzing current and historical data and projecting what it learns on a model generated to forecast likely outcomes

4.1 Decision Trees

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g., whether a coin flip comes up heads or tails), each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression task.

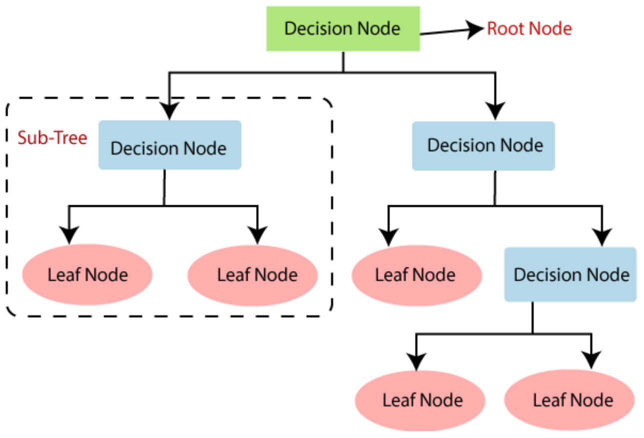


Figure 10: Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

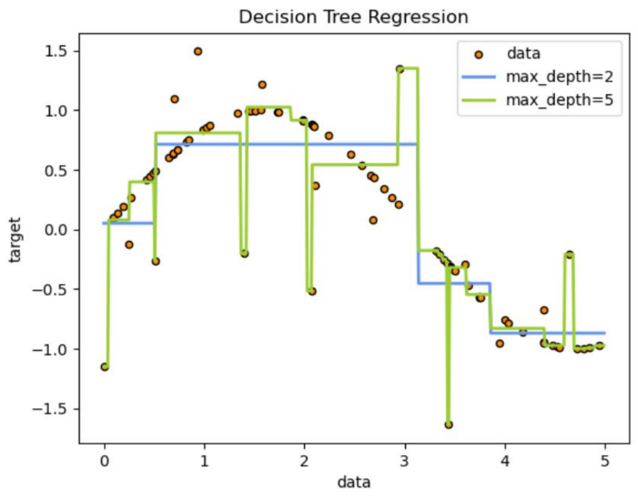
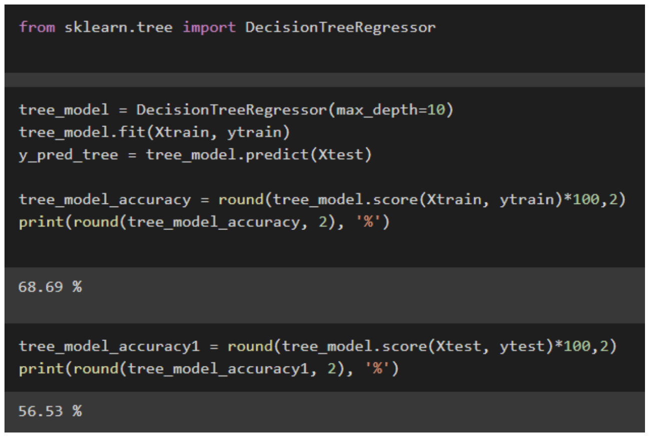


Figure 11: Decision Tree Regression

The decision trees are used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve. We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e., they are overfit.



```
from sklearn.tree import DecisionTreeRegressor
```

```
tree_model = DecisionTreeRegressor(max_depth=10)
tree_model.fit(Xtrain, ytrain)
y_pred_tree = tree_model.predict(Xtest)

tree_model_accuracy = round(tree_model.score(Xtrain, ytrain)*100,2)
print(round(tree_model_accuracy, 2), '%')
```

```
68.69 %
```

```
tree_model_accuracy1 = round(tree_model.score(Xtest, ytest)*100,2)
print(round(tree_model_accuracy1, 2), '%')
```

```
56.53 %
```

Figure 12: Decision Tree Regressor Code used

Upon using the Decision Tree Regressor model, we found that the training set had an accuracy of 68.69%. Also using the testing data set on the Decision Tree Regressor model, we get the accuracy of 56.53%.

4.2 Support Vector Regressor

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

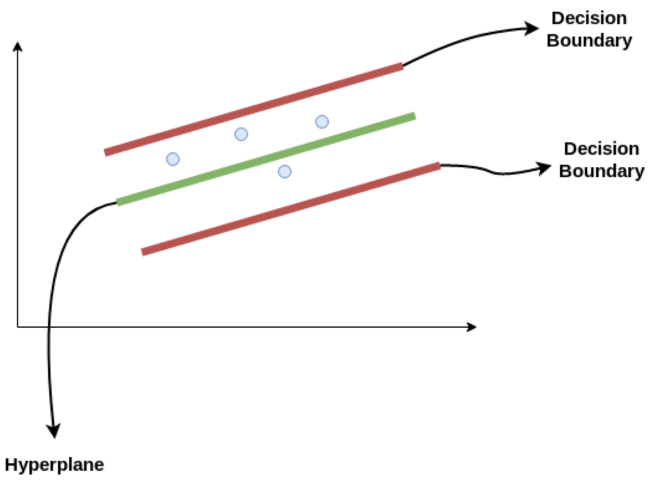


Figure 13: Support Vector regression (SVR)

Support Vector Machine can be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities.

In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

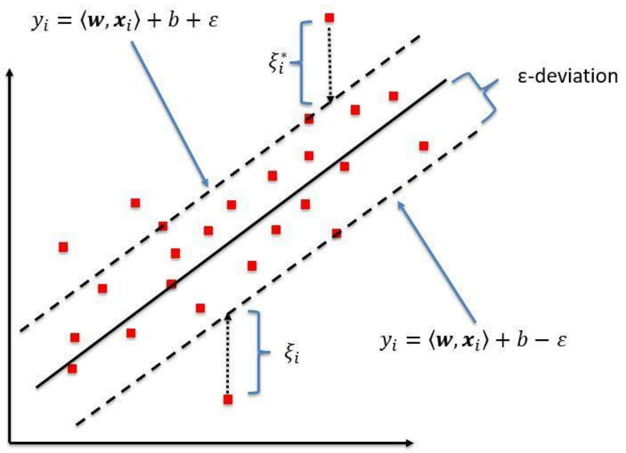

$y_i = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b + \varepsilon$
ξ_i^*
ε-deviation
ξ_i
$y_i = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - \varepsilon$

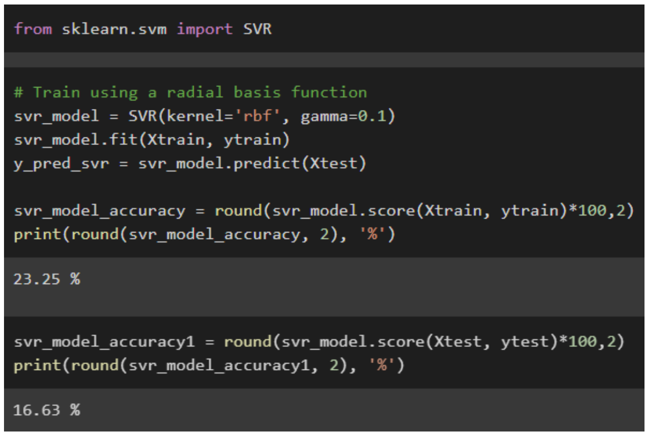
Figure 14: Support Vector regression (SVR) model

A kernel is a function that takes the original non-linear problem and transforms it into a linear one within the higher-dimensional space. RBF is the default kernel used within the sklearn's SVM classification algorithm and can be described with the following formula:

$$K(x, x') = e^{-\gamma ||x-x'||^2}$$

Where $||x - x'||^2$ is the squared Euclidean distance between two feature vectors and γ(gamma) is a scalar that defines how much influence a single training example (point) has.

In the Support Vector Regressor, we have calculated the accuracy of the model using the Radical Basis Function Kernel. RBF kernels are the most generalized form of kernelization and is one of the most wisely used kernels due to its similarity to the Gaussian Distribution.



```
from sklearn.svm import SVR

# Train using a radial basis function
svr_model = SVR(kernel='rbf', gamma=0.1)
svr_model.fit(Xtrain, ytrain)
y_pred_svr = svr_model.predict(Xtest)

svr_model_accuracy = round(svr_model.score(Xtrain, ytrain)*100,2)
print(round(svr_model_accuracy, 2), '%')

23.25 %

svr_model_accuracy1 = round(svr_model.score(Xtest, ytest)*100,2)
print(round(svr_model_accuracy1, 2), '%')

16.63 %
```

Figure 15: Support Vector Tree Regressor Code used

Upon using the Support Vector Regressor model, we found that the training set had an accuracy of 23.25%. Also using the testing data set on the Support Vector Regressor model, we get the accuracy of 16.63%. As compared to the Decision Tree Regressor, the accuracy of both the training and testing data is really not good, and this can be due to the fact that SVR underperforms when the number of features for each data point exceeds the number of training data samples. Thus, Support Vector Regressor are not suitable for large datasets.

4.3 Random Forest Regressor

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. The diagram below shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them.

A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.

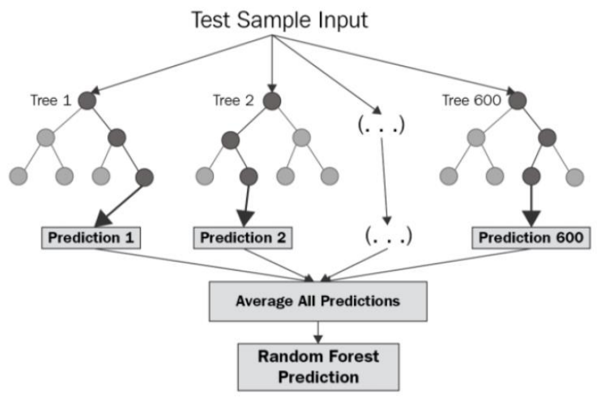


Figure 16: Random Forest Regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole data set is used to build each tree.

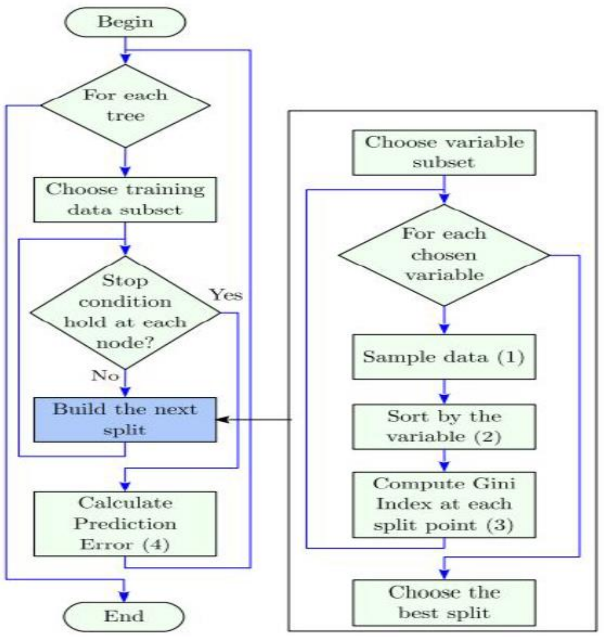


Figure 17: Random Forest working

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

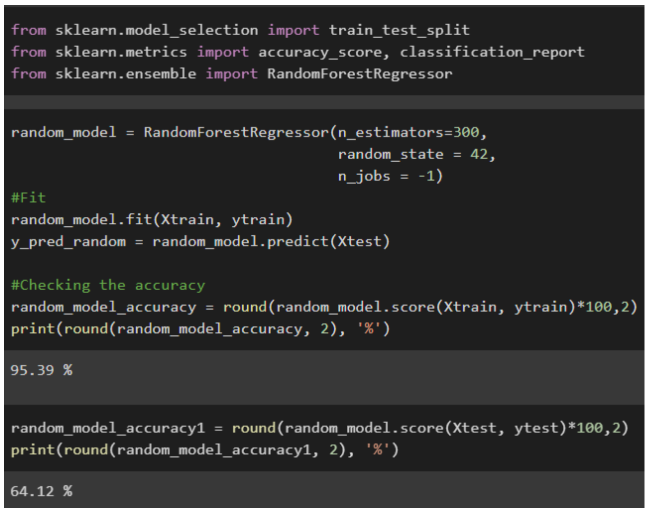
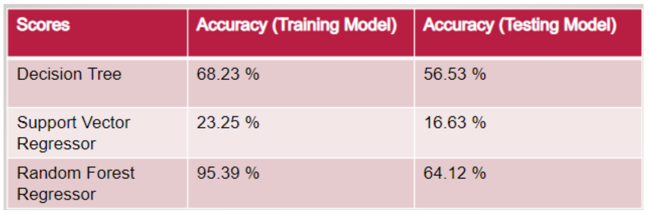

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestRegressor

random_model = RandomForestRegressor(n_estimators=300,
                                     random_state = 42,
                                     n_jobs = -1)
#Fit
random_model.fit(Xtrain, ytrain)
y_pred_random = random_model.predict(Xtest)

#Checking the accuracy
random_model_accuracy = round(random_model.score(Xtrain, ytrain)*100,2)
print(round(random_model_accuracy, 2), '%')

95.39 %

random_model_accuracy1 = round(random_model.score(Xtest, ytest)*100,2)
print(round(random_model_accuracy1, 2), '%')

64.12 %
```

Figure 18: Random Forest Regressor Code

Upon using the Random Forest Regressor model, we found that the training set had an accuracy of 95.39%. And this is quite good as we are able to fit our model in accordance with the training data set. Also using the testing data set on the Random Forest Regressor model, we get the accuracy of 64.12%. This value is quite good compared to the accuracies that we got from using the Decision Tree and Support Vector Regressor.



Scores	Accuracy (Training Model)	Accuracy (Testing Model)
Decision Tree	68.23 %	56.53 %
Support Vector Regressor	23.25 %	16.63 %
Random Forest Regressor	95.39 %	64.12 %

Figure 19: Accuracy of different models at a glance

We will finalize random forest as our final model for the predictions and calculations. The next step would be to tune the model. As decision trees are highly prone to overfitting and random forest regressor works with combining multiple decision tree, we might have overfitting present in our current model. Thus tune the model can help us reduce overfitting and also increase the accuracy of testing data set.

5. TUNING THE MODEL

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. It is possible and recommended to search the hyper-parameter space for the best cross validation score. Cross-Validation permits us to evaluate and improve our model.

GridSearchCV is an effective method for adjusting the parameters in supervised learning and improve the generalization performance of a model. With Grid Search, we try all possible combinations of the parameters of interest and find the best ones. The GridSearchCV object searches for the best parameters and automatically fits a new model on the whole training dataset. GridSearchCV implements a "fit" and a "score" method. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

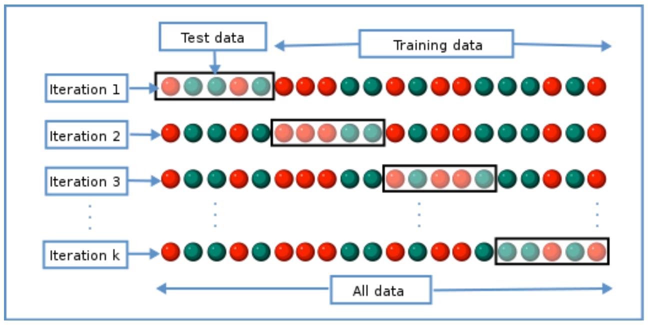


Figure 20: GridSearchCV

RandomizedSearchCV is very useful when we have many parameters to try, and the training time is very long. RandomizedSearchCV also implements a "fit" and a "score" method. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.

If all parameters are presented as a list, sampling without replacement is performed. If at least one parameter is given as a distribution, sampling with replacement is used. It is highly recommended to use continuous distributions for continuous parameters.

In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter. The grid search provided by GridSearchCV exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter.

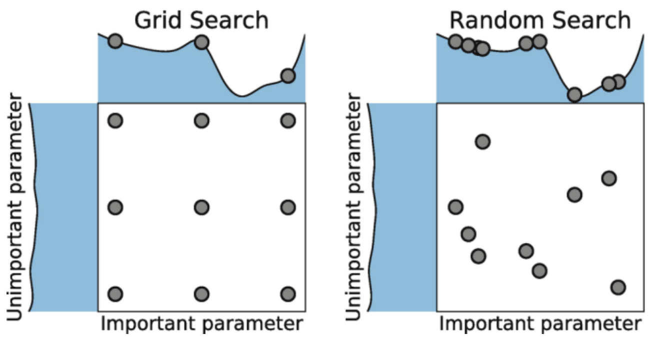
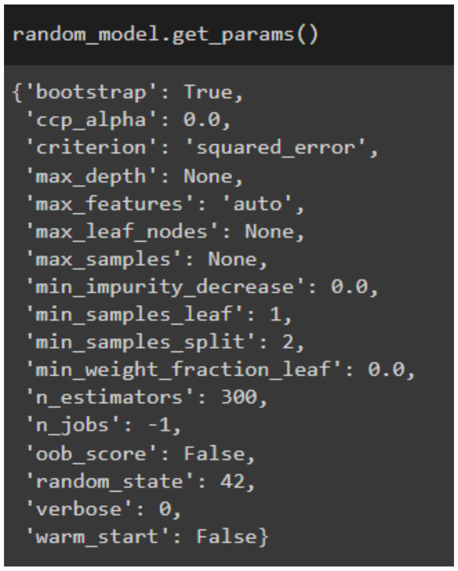


Figure 21: GridSearchCV vs RandomSearchCV

So, Grid Search is good when we work with a small number of hyperparameters. However, if the number of parameters to consider is particularly high and the magnitudes of influence are imbalanced, the better choice is to use the Random Search.

Thus, here we would be using RandomSearchCV as GridSearchCV is exhaustive and takes a lot of time. Below is the list of parameters that are present for a random forest regressor model.

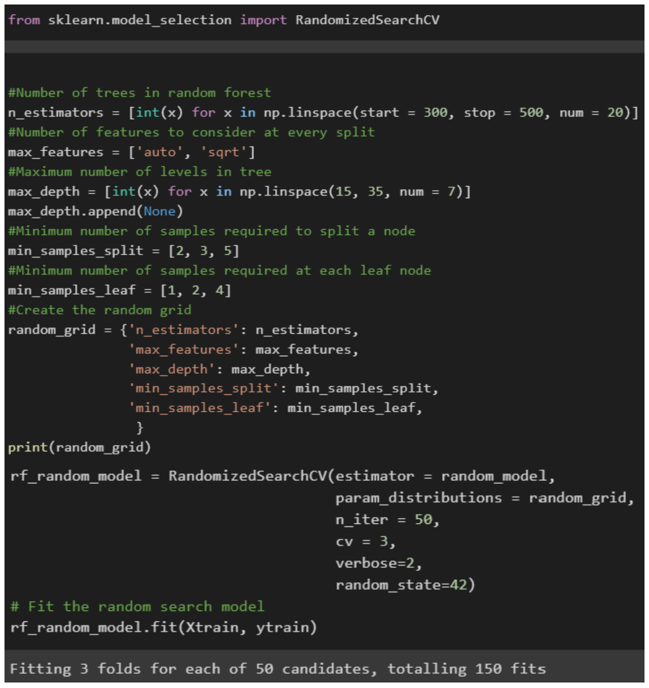


```
random_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 300,
 'n_jobs': -1,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Figure 22: Parameters for Random Forest model

Among the various available parameters for a random forest regressor, we choose a selected few for hyper parameter tuning our model. In this case, we have chosen 'n_estimators', 'max_features', 'max_depth', 'min_samples_split' and 'min_samples_leaf' as parameters and set the number of iterations to 50 and also a 3-fold cross validation. A random search across 100 different combinations and use all available cores



```
from sklearn.model_selection import RandomizedSearchCV
```

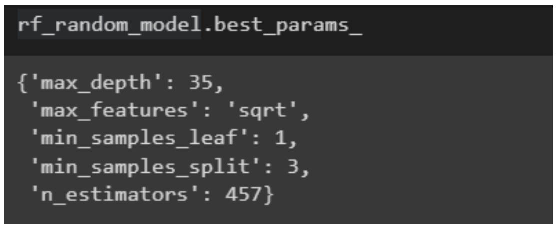
```
#Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 300, stop = 500, num = 20)]
#Number of features to consider at every split
max_features = ['auto', 'sqrt']
#Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(15, 35, num = 7)]
max_depth.append(None)
#Minimum number of samples required to split a node
min_samples_split = [2, 3, 5]
#Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
#Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
                }
print(random_grid)
```

```
rf_random_model = RandomizedSearchCV(estimator = random_model,
                                     param_distributions = random_grid,
                                     n_iter = 50,
                                     cv = 3,
                                     verbose=2,
                                     random_state=42)
# Fit the random search model
rf_random_model.fit(Xtrain, ytrain)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Figure 23: Setting parameters for Hyper Parameter Tuning

After running the RandomSearchCV, we can use 'best_params' to get a set of parameters for which the model scores the best score and then we can use those parameters to create a model specific to those specifications so as to reduce the overfitting that was happening earlier.

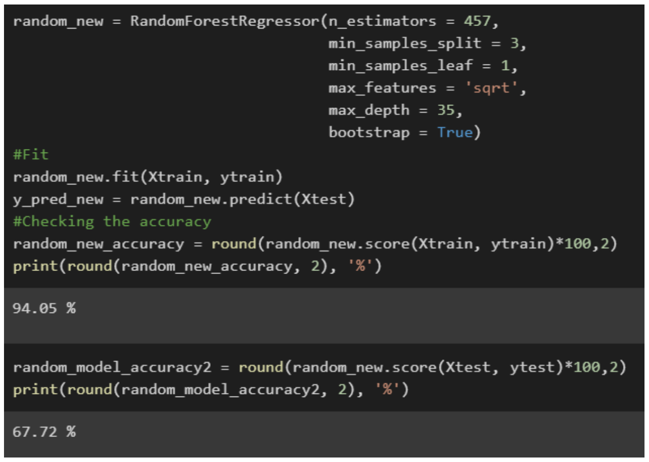


```
rf_random_model.best_params_
```

```
{'max_depth': 35,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 3,
 'n_estimators': 457}
```

Figure 24: Best Parameters according to the RandomSearchCV

Using the above parameters, to create a new Random Forest Regressor Model, and check the accuracy for both the training data set and the testing data set.



```
random_new = RandomForestRegressor(n_estimators = 457,
                                   min_samples_split = 3,
                                   min_samples_leaf = 1,
                                   max_features = 'sqrt',
                                   max_depth = 35,
                                   bootstrap = True)
#Fit
random_new.fit(Xtrain, ytrain)
y_pred_new = random_new.predict(Xtest)
#Checking the accuracy
random_new_accuracy = round(random_new.score(Xtrain, ytrain)*100,2)
print(round(random_new_accuracy, 2), '%')
```

```
94.05 %
```

```
random_model_accuracy2 = round(random_new.score(Xtest, ytest)*100,2)
print(round(random_model_accuracy2, 2), '%')
```

```
67.72 %
```

Figure 25: New Random Forest Model Code

Using the new Random Forest Regressor model, we see that the accuracy for the training dataset has reduced from 95.39% to 94.05% and this is as expected as the model previously was overfitting and reducing overfitting has resulted in reduction of training data accuracy.

Upon check the accuracy on the testing data set, we see that the accuracy has increased from 64.12% to 67.72%. Although there is an increase in the accuracy of the Random Forest Regressor model, we can use further techniques in future to increase the accuracy more.

6. CONCLUSION AND FUTURE WORK

Forest fires cause a significant environmental damage while threatening human lives. A substantial effort was made in the last two decades to build automatic detection tools that could assist Fire Management Systems (FMS). The three major trends are the use of satellite data, infrared/smoke scanners and local sensors (e.g., meteorological). In this work, we propose a Machine Learning approach that uses the forest fire dataset from the Fire Information for Resource Management System.

Our final result is the accuracy of 67.72% using the Random Forest Regressor with RandomSearchCV hyperparameter tuning. In the future, we would like to implement more methods to increase the accuracy of the current model and also later on to include various other factors too that affect the possibility of forest fire in the model. The future of the project can be acceding many other variables as we won't be able to include all of them. This issue is a global issue, and this can be used for other nation's data

7. REFERENCES

[1] https://firms.modaps.eosdis.nasa.gov/

[2] G. E. Sakr, I. H. Elhajj, G. Mitri and U. C. Wejinya, "Artificial intelligence for forest fire prediction," 2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Montreal, ON, 2010, pp. 1311-1316.

[3] fs.fed.us/research/highlights/highlights_display.php?in_high_id=67

[4] https://towardsdatascience.com/machine-learning-basics-support-vector-regression-660306ac5226

[5] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[6] https://www.netsuite.com/portal/resource/articles/financial-management/predictive-modeling.shtml

[7] https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

[8] https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

[9] https://towardsdatascience.com/svm-classifier-and-rbf-kernel-how-to-make-better-models-in-python-73bb4914af5b

[10]https://scikit-learn.org/stable/modules/svm.html

[11]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[12]https://scikit-learn.org/0.16/modules/generated/sklearn.grid_search.RandomizedSearchCV.html

[13]https://www.geeksforgeeks.org/random-forest-regression-in-python/

[14]https://towardsdatascience.com/machine-learning-gridsearchcv-randomizedsearchcv-d36b89231b10

[15]https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/