Prediction of Forest fire occurrences using Machine Learning

Data Set : Algerian forest fires

Jugal Krishna Kakarla, jkakarla@usc.edu

May 4, 2022

1 Abstract

A forest fire or a wildfire includes unplanned, uncontrolled fire in an area of combustible vegetation and thereby destroying everything in its wake such as vegetation, human and animal lives, etc. The resulting economic, ecological and human life damages can be reduced if we can accurately predict the fire on the future dates and thereby adopting the precautionary measures and mitigating this threat. In this project, I propose five classification models: **Support Vector Machines (SVMs), Logistic Regression, Random Forest, K Nearest Neighbors(KNNs) and Multi-layer Perceptron (MLP)** to predict the possibility of fire in the future. All these models are compared based on the performance measures: Test Accuracy, F1-score, and confusion matrices on the Algerian Fires dataset, which originally consisted of 9 real-valued input features. In addition to these features, I have performed feature engineering to generate 9 more new features, which are the previous 3-day rolling averages of the original 9 features. This has enabled me to obtain a 3% higher test accuracy of 91.67% on my SVM model over unnormalized original data and thereby giving the highest test accuracy measure among all the models. Moreover, the final models of SVM, logistic regression, Random Forest that I have implemented after preprocessing and feature engineering have demonstrated test set accuracies of **91.67% i.e, over 10% higher than the baseline system** given, i.e, Nearest Means Classifier

Keywords: Support Vector Machines (SVMs), Logistic Regression, Random Forest, K Nearest Neighbors (KNNs), feature engineering

2 Introduction

2.1 Problem Assessment and Goals

Forest fires are prevalent in countries like Algeria because the dry climate, abundant winds, and dried vegetation provide prime conditions for a wildfire—and it only takes a single ember to ignite and destroy hundreds of thousands of acres. In the year 2021, Algeria experienced some of the worst wildfires in years when more than 120,000 hectares were burned in a single week[1] and at least 90 people have reportedly died as a consequence[2]. Over a period of 25 years, from 1985 to 2010, Algeria recorded 42,555 fires that burned a total area of 910,640 hectares[2]. Even in the US, states like California experience destructive forest fires every year. In the year 2022 alone, 1,402 fires, that have destroyed 6,507 acres of forest were reported[3]. With an average increase in the number of fires of over 11% per year[3], forest fires are becoming increasingly destructive and thereby escalating the need for models to accurately predict the future fires given some conditions.

The field of machine learning makes building such models possible. Various classification algorithms have been developed over time that can be used for such classification tasks. Some of them include Nearest Means Classifier, Logistic Regression, SVMs, Random Forest, KNNs, Multi-Layer Perceptrons, etc. The basic idea of a classification machine learning model can be described as : We train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data and once the training is done, the model is used to assign classes to the data that it hasn't seen before. In this project, I have attempted to use some of these classification algorithms to perform the task of accurately predicting the possibility of future forest fires.

2.2 Literature Review

Significant work on this problem is done by Faroudja Abid and Nouma Izeboudjen [4], where they have used a decision tree based system for forest fire prediction. They have integrated the decision tree classifier as a part of the smart sensor node architecture with Wireless Sensor Networks (WSN) that allows fire prediction in an automated and intelligent way without requiring human intervention and have reported accuracy of 82.92%.

3 Approach and Implementation

3.1 Dataset Usage

The Algerian forest fire dataset actually consists of 184 datapoints with 11 real-valued features namely:
1. Date: Date when the data point was collected (in day/month/year format)
2. Temperature: Max. temperature of the day in degrees Celsius
3. RH: Humidity
4. Ws: Wind speed in km/h
5. Rain: rain level in mm
6. FFMC: Fine Fuel Moisture Code
7. DMC: Duff Moisture Code
8. DC: Drought Code
9. ISI: Initial Spread Index
10. BUI: Buildup Index
11. Classes: The label of the dataset (1: there was a fire, 0: not a fire)

I have performed feature engineering for all the classes except Date and Classes for both the training set and the test set. The test set consisted of 60 datapoints. Both the train and test datasets are quite balanced with 45% of the data points in the "fire" class and 55% in the "not fire" class. During the validation phase, I used K-fold cross-validation, (with some changes of my own as mentioned in the section 3.5.1), with 10 folds with a single run, where I coded the function by myself. The size of the validation set in each case was 18 and the size of the training set was 163 during every fold since the 3 dates prior to the start date of the validation set were removed after feature engineering as otherwise, the validation set would have prior information about the outcome of the dates. For every model, the test dataset has only been used once to evaluate the model with optimum parameters obtained from the grid search in the cross-validation.

3.2 Preprocessing

I have coded the function for Data Normalization partly by myself using the Standard Scaler function from sklearn. Using this function, I have normalized the input data and have observed an increase in accuracy of over 1.67% for SVMs, Logistic regression classifier and MLP and, 1% for KNNs with a significantly lower training and inference times, thus making it on par with the engineered-feature dataset in terms of accuracy and overwhelmed the latter in terms of training and inference speeds.

3.3 Feature engineering

I have performed feature engineering for all the features except Date and Classes for both the training set and the test set by computing the rolling averages of the past 3 days. The 3 dates prior to the start date of the validation set and the test set were removed from the training dataset after feature engineering as otherwise, the validation set or the test set would have prior information about the outcome of the dates. Thus, we must use only past dates when collecting the statistics. Therefore, the total number of features in my new datasets are 18 excluding the Date and Classes. Though this improved accuracies in SVM over the normalized original data, the training time was comparitively much higher. In the remaining models, it did not result in accuracy improvement over the normalized original data.

3.4 Feature dimensionality adjustment

I have opted not to do dimensionality reduction or feature selection because, the K-best features that best define the training set may not be the same K-best features for the test or validation sets. To verify my hypothesis, I have selected the 9-best features for the first 160 datapoints and the remaining 24 datapoints of the training dataset and 2 of the 9 features did not match in both the cases. In further support of my hypothesis, the actual validation and test accuracies have dropped by around 2.2% for the SVM, 1.5% for logistic regression, 3% for Random Forest and 4.7% for KNN models.

3.5 Training, Classification or Regression, and Model Selection

The models I have used for this project are : Support Vector Machines (SVMs), Logistic Regression, Random Forest, K Nearest Neighbors (KNNs).

3.5.1 Model Selection and hyperparameter tuning

I have opted to use the K-fold cross validation algorithm for hyperparameter tuning and model selection in addition to using a grid search. However, one change I have made in my implementation is the removal of shuffling the dataset step for reproducibility of the results. The algorithm is as follows:

```
1. Split the training data $D$ into $K$ disjoint subsets i.e, $D_1, D_2, ...., D_K$ .
2. For $i = 1, 2, ....K$, begin:
i.      $D_v = D_i; D_{tr} = \bigcup_{j \neq i} D_j$
ii.     For each parameter in the grid of parameters, begin: {
iii.            Initialize the training procedure.
iv.             Train the classifier using the parameter on $D_{tr}$.
v.              Evaluate the classifier on $D_v$.
vi.             Store error $E_{pi}$.
vii.            Repeat for all parameters while storing $pi^* = argmin_p E_{pi}$
}
3.Repeat for all i and the average error for the parameter value $p$ is: $E_p = \frac{1}{K}\sum_{i=1}^{K} E_{pi}$
4. Then pick $p^* = argmin_p E_p$.
```

3.5.2 Classification Models

As mentioned earlier, I have used 4 models for this project namely, Support Vector Machines (SVMs), Logistic Regression, Random Forest, K Nearest Neighbors (KNNs) in addition to the baseline model given i.e; Nearest Means Classifier.

1. Nearest Means Classifier (Baseline Model):

A nearest means / nearest centroid classifier is a classification model that assigns the label of the class of training samples to the observations with respect to the closest mean (centroid) to the observation. The metric used in this project is the 'Eucledian Distance'.
Algorithm:[5]
Training: Given labeled training samples $\{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ with class labels $y_i \in \mathbf{Y}$, compute

the per-class centroids $\vec{\mu}_\ell = \frac{1}{|C_\ell|} \sum_{i \in C_\ell} \vec{x}_i$ where C_ℓ is the set of indices of samples belonging to class $\ell \in \mathbf{Y}$.
Prediction: The class assigned to an observation $\vec{x}$ is $\hat{y} = \arg\min_{\ell \in \mathbf{Y}} \|\vec{\mu}_\ell - \vec{x}\|$. The best training accuracy I could obtain on this classifier is 88.04% and the best test accuracy is 81.67%.

Performance metrics on train set:
Accuracy on train set = 88.04347826086956%
F1 score = 0.8981481481481481

True/Predicted	Class 0	Class 1
Class 0	65	4
Class 1	18	97

Table 1 : Confusion Matrix of NMC on train dataset

Performance metrics on test set:
Accuracy on test set = 81.66666666666667%
F1 score = 0.7027027027027025

True/Predicted	Class 0	Class 1
Class 0	36	1
Class 1	10	13

Table 2 : Confusion Matrix of NMC on test dataset

2. Support Vector Machines(SVMs):

Support-vector machines are one of the more robust supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. SVM maps training examples to points in space in order to to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can also perform non-linear classification, equally efficiently, using various kernels, by implicitly mapping their inputs into high-dimensional feature spaces[7].

In this project, I have used both linear and non-linear kernels as parameters in my model selection and selected the optimum kernel along with other optimum parameters like the C-value, gamma and degree.

Algorithm for Linear SVM:[7]
Consider a training dataset of n datapoints of the form $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where, $\mathbf{x}_i$ is a p-dimensional real vector and y_i is either 1 or $-\mathbf{1}$, indicating the class to which the point $\mathbf{x}_i$ belongs. We should find the "maximum-margin hyperplane" that divides the group of points $\mathbf{x}_i$, for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\mathbf{x}_i$ from either group is maximized.
Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying $\mathbf{w}^T\mathbf{x} - b = 0$, where $\mathbf{w}$ is the (not necessarily normalized) normal vector to the hyperplane. This is much like Hesse normal form [8], except that $\mathbf{w}$ is not necessarily a unit vector. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

Non-Linear Kernels:[7]
Some common kernels include:
Polynomial (homogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$. Particularly, when $d = 1$, this becomes the linear kernel.
Polynomial (non-homogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + r)^d$.

Gaussian radial basis function: $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$ for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/(2\sigma^2)$.
Sigmoid function (Hyperbolic tangent): $k(cx_i, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} + c)$ for some (not every) $\kappa > 0$ and $c < 0$ ¡ 0.
The kernel is related to the transform $\varphi(\vec{x_i})$ by the equation $k(\vec{x_i}, \vec{x_j}) = \varphi(\vec{x_i}) \cdot \varphi(\vec{x_j})$. The value w is also in the transformed space, with $\vec{w} = \sum_i \alpha_i y_i \varphi(\vec{x}_i)$. Dot products with w for classification can again be computed by the kernel trick, i.e. $\vec{w} \cdot \varphi(\vec{x}) = \sum_i \alpha_i y_i k(\vec{x}_i, \vec{x})$.

For this model, I have performed the grid-search along with K-fold cross validation over the hyperparameters: Inverse regularization strength (C), Kernel type, gamma value, degree, and have obtained the optimum model for the parameters: Kernel = polynomial, C = 0.5, Gamma = 0.1, Degree = 1. The respective performance metrics on the training and test dataset are shown below:

Performance metrics on train set:
Accuracy on train set = 92.81767955801105%
F1 score = 0.9427312775330398

True/Predicted	Class 0	Class 1
Class 0	61	6
Class 1	7	109

Table 3 : Confusion Matrix of SVM on train dataset

Performance metrics on test set:
Accuracy on test set = 91.66666666666666%
F1 score = 0.888888888888889

True/Predicted	Class 0	Class 1
Class 0	35	2
Class 1	3	20

Table 4 : Confusion Matrix of SVM on test dataset

3. Logistic Regression:[9]

The logistic model is a machine learning model that models the probability of one event (out of two possibilities) taking place by having the log-odds function for the event to be a linear combination of one or more predictors or input features. In binary logistic regression there is a single binary dependent variable, coded by a indicator variable, where the two values are labeled "0" and "1", while the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). Logistic regression applies the logistic sigmoid function to weighted input values to generate a prediction of the data class.
The logistic function is of the form:
$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}}$, where is a location parameter (the midpoint of the curve, i.e, where $p(\mu) = 1/2$) and s is a scale parameter.
This expression may be rewritten as: $p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$, where $\beta_0 = -\mu/s$ and is known as the intercept (it is the vertical intercept or y-intercept of the line $y = \beta_0 + \beta_1 x$), and $\beta_1 = 1/s$ (inverse scale parameter or rate parameter): these are the y-intercept and slope of the log-odds as a function of x. Conversely, $\mu = -\beta_0/\beta_1$ and $s = 1/\beta_1$.

Train / Fit phase:
The logistic regression uses, the negative log-likelihood loss to measure the 'goodness' of a fit. For a given input x_k, we write the probalility as $p_k = p(x_k)$ where, $p(x)$ is the logistic function and p_k

represent the probabilities that the corresponding outputs, y_k will be unity and $1 - p_k$ represent the probabilities that they will be zero. Next, we should find the values of β_0 and β_1 which give the least loss value on the data.
The negative log-likelihood loss (or the log loss) for the k-th point is given by:

$-\ln p_k$ if $y_k = 1$,
$-\ln(1 - p_k)$ if $y_k = 0$.

The above two expressions can be combined into a single function to give:
$l = -y_k \ln p_k - (1 - y_k)\ln(1 - p_k)$.
This equation is also known as the cross entropy of the predicted distribution $\big(p_k, (1 - p_k)\big)$
One key point to note is that the negative log-likelihood loss is always greater than or equal to 0 as $p_k \in [0, 1]$.
The sum of such loss functions over the entire dataset, is the overall negative log-likelihood loss, and our aim is to find the values of β_0 and β_1 that minimize this loss. Another method to do this is by maximizing its inverse, i.e, the (positive) log-likelihood loss:

$$\ell = \sum_{k:y_k=1} \ln(p_k) + \sum_{k:y_k=0} \ln(1 - p_k) = \sum_{k=1}^{K} \left(y_k \ln(p_k) + (1 - y_k)\ln(1 - p_k)\right)$$

, which is similar to maximizing the likelihood function itself:

$$L = \prod_{k:y_k=1} p_k \prod_{k:y_k=0} (1 - p_k)$$

This method is known as maximum likelihood estimation.

Estimation of β_0 and β_1:
Estimation of β_0 and β_1 is a minimization problem and moreover, the loss function L is non-linear with respect to β_0 and β_1. Therefore, this can be solved by equating the gradient of the loss function with respect to β_0 and β_1 to zero i.e,

$$\frac{\partial \ell}{\partial \beta_0} = \sum_{k=1}^{K} (y_k - p_k) = 0$$

$$\frac{\partial \ell}{\partial \beta_1} = \sum_{k=1}^{K} (y_k - p_k)x_k = 0$$

and solve for β_0 and β_1.

Prediction:
The values of β_0 and β_1 can now be entered into the logistic function and the probability estimate for each class can be computed for an input.

In this project, I have performed the grid-search along with K-fold cross validation over the hyperparameters: Inverse regularization strength (C), penalty function, solver, and have obtained the optimum validation accuracy of 92.22% for the model with the parameters: Optimum C = 2, Optimum Solver = liblinear, Optimum Penalty = l2. The respective performance metrics on the training and test dataset are shown below:

Performance metrics on train set:
Accuracy on train set = 94.02173913043478%
F1 score = 0.9511111111111111

True/Predicted	Class 0	Class 1
Class 0	66	3
Class 1	8	107

Table 5 : Confusion Matrix of Logistic regression on train dataset

Performance metrics on test set:
Accuracy on test set = 91.66666666666666%
F1 score = 0.8837209302325583

True/Predicted	Class 0	Class 1
Class 0	36	1
Class 1	4	19

Table 6 : Confusion Matrix of Logistic regression on test dataset

4. Random Forest:[10]

Random forests is an ensemble learning algorithm, used for classification, regression and other tasks. It operates by building a multiple number of decision trees during the training phase and to perform classification tasks, it produces the output class that is selected by the most number of trees. One key advantage of random forests is that they do not need normalization. This claim is further supported by its performance of normalized and un-normalized data in section[4] of this report.

Algorithm:

(i) Decision Trees:[12]
A decision tree classifies the input data based on one input feature at a time and this process is repeated until all the input features are utilized. One key issue with the decision trees is that their depth increases proportionally with the increase in the number of input parameters and with this increasing depth, they overfit the data i.e, incur low bias but high variance. Random forests solve this problem as they consider a number of such trees trained on the same dataset and reduces variance with an expense of minor increase in bias due to the Variance-Bias trade-off.

(ii) Feature Bagging:[13]
Bagging is an ensemble machine learning algorithm that improves the stability of the model by reducing the variance and thus help avoid overfitting. Given the training datapoints $X = x1, ..., xn$ with outputs $Y = y1, ..., yn$, bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples.
For $b = 1, ..., B$:
1. Sample X_b, Y_b from the training data X, Y, with replacement.
2. Train a classification tree f_b on X_b, Y_b.
After training, predictions for unseen samples X' can be made by by taking the majority vote of the predictions from all the individual trees on X'.
Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on X':
$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}$, where B is the number of trees.

One key issue with the ordinary bagging process is that if one or a few of the features turn out to be very strong predictors for the target, these features will be selected in many of the B trees, causing them to become correlated. In order to mitigate this, at each candidate split in the learning process, a random subset of the features are selected (generally $\sqrt{p}$ for p features), and this process is known as "feature bagging".

In this project, I have performed the grid-search along with K-fold cross validation over the hyperparameters: Number of estimators, Maximum depth of each tree, and have obtained the optimum model with validation accuracy of 91.11% and the parameters: Optimum number of estimators = 10, Optimum Max depth = None (here none denotes that nodes are expanded until all leaves (final nodes) are pure or until all leaves contain less than minimum number of split samples). The respective performance metrics on the training and test dataset are shown below:

Performance metrics on train set:
Accuracy on train set = 100%
F1 score = 1.0

True/Predicted	Class 0	Class 1
Class 0	69	0
Class 1	0	115

Table 7 : Confusion Matrix of Random forest on train dataset

Performance metrics on test set:
Accuracy on test set = 91.66666666666666%
F1 score = 0.8936170212765957

True/Predicted	Class 0	Class 1
Class 0	34	3
Class 1	2	21

Table 8 : Confusion Matrix of Random Forest on test dataset

5. K Nearest Neighbors:[11]

K Nearest Neighbors is a supervised classification model that outputs the class of an input datapoint x_i based on the plurality vote of its neighbors, with x_i being assigned to the class most common among its k nearest neighbors ($k \in N$). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Algorithm:

The training data consist of multidimensional vectors, each with a class label. During the training phase, the feature vectors and their respective classes are stored.

In the prediction phase, a testing datapoint x_i is classified to the class label with the highest frequency among its k nearest neighbors. One metric that is most frequently used for selecting the k nearest neighbors of a sample is the "Eucledian distance". Often, normalization of the data improves the acuracy of the KNN classifier due to the reduction in noise as a result of standardization of the variance.

In this project, I have performed the K-fold cross validation to find the optimum number of nearest neighbors and have obtained the optimum validation accuracy of 90% for optimum number of nearest neighbors = 5. The respective performance metrics on the training and test dataset are shown below:

Performance metrics on train set:
Accuracy on train set = 90.76086956521739%
F1 score = 0.9270386266094419

True/Predicted	Class 0	Class 1
Class 0	59	10
Class 1	7	108

Table 9 : Confusion Matrix of KNN on train dataset

Performance metrics on test set:

Accuracy on test set = 86.66666666666666%
F1 score = 0.8181818181818182

True/Predicted	Class 0	Class 1
Class 0	34	3
Class 1	5	18

Table 10 : Confusion Matrix of KNN on test dataset

6. Multi-layer Perceptron:[12]

A multilayer perceptron (MLP), a non-linear classifier, is a type of Artificial Neural Network that is fully connected. The input to the MLP is the original feature space of the dataset but the hidden layers perform non-linear mapping on it to a new feature space. In an MLP with L hidden layers, the first N layers perform feature extraction and the subsequent $L-N$ layers perform classification. The 'depth' of the network (i.e, the number of hidden layers) can be determined by model selection methods. The criterion function is given by:
$J = \sum_{n=1}^{N} \epsilon^{(n)} = \sum_{n=1}^{N} \frac{1}{2} \sum_{i=1}^{C} L\left(v_{i,L}^{(n)}, \bar{v}_{i,L}^{(n)}\right)$, where C is the total number of classes and L is a classification loss function. Some of them include cross-entropy loss, negative lod-likelihood loss, etc.
Then, this criterion function is minimized by gradient descent using an optimization algorithm such as ADAM, SGD etc. Various hyperparameters for tuning this model include learning rate, optimizer, loss function, number of layers. In this project, I have performed the grid-search along with K-fold cross validation to find the optimum parameters for this model and have acheived an optimum validation accuracy of 92.78% with the hyperparameters: Optimum number of hidden layers = 50, Optimum activation = tanh, Optimum solver/optimizer = adam, Optimum Learning Rate = 0.1. The respective performance metrics on the training and test dataset are shown below:

Performance metrics on train set:
Accuracy on train set = 100.0%
F1 score = 1.0

True/Predicted	Class 0	Class 1
Class 0	69	0
Class 1	0	115

Table 9 : Confusion Matrix of MLP on train dataset

Performance metrics on test set:
Accuracy on test set = 90%
F1 score = 0.8749999999999999

True/Predicted	Class 0	Class 1
Class 0	33	4
Class 1	2	21

Table 10 : Confusion Matrix of MLP on test dataset

One major caveat with the neural networks is the selection of number of hidden layers. It is so difficult to fine tune the model due to the enormous number of combinations of the hyperparmeters that can be selected. Moreover, the deeper networks require large datasets to overcome the problem of underfitting.

7. Trivial Regressor:

A trivial system outputs class assignments (S1, S2) at random with probability N1/N and N2/N, respectively; where, Ni is the population of data points with class label Si, and N is the total population of data points. This is all based on the training set.
I have acheived train accuracy of 62.5% and test accuracy of 38.33% on this trivial regressor.

4 Analysis: Comparison of Results, Interpretation

4.1 Effect of Standardization

The tables shown below demonstrate the comparisions of training and testing accuracies on un-standardized and standardized data:

Model	Train accuracy on un-normalized data	Train accuracy on normalized data
NMC	74.45652173913044%	88.04347826086956%
SVM	92.93478260869566%	94.02173913043478%
Logistic Regression	93.47826086956522%	94.02173913043478%
Random Forest	100%	100%
KNN	89.13043478260869%	90.76086956521739%
MLP	95.1086956521739%	100%

Table 13: Comparision of training accuracies before and after normalized on all models

Model	Test accuracy on un-normalized data	Test accuracy on normalized data
NMC	78.33333333333333%	81.66666666666667%
SVM	88.33333333333333%	90.0%
Logistic Regression	90.0%	91.66666666666667%
Random Forest	91.66666666666667%	91.66666666666667%
KNN	85.0%	85.66666666666667%
MLP	88.33333333333333%	90.0%

Table 14: Comparision of test accuracies before and after normalized on all models

As depicted in the above tables, normalization almost always results in an improvement in accuracies of the machine learning models. This is because algorithms like linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be normalized and therefore have a uniform step-size for all the features. This ensures that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features. The only model that doen't depend on normalization is, as mentioned earlier in the previous section, the Random forest classifier. This is because the Tree-based models do not care about the absolute value that a feature takes, but only care about the order of the values. Moreover, normalization improves the training speeds as the features after normalized follow the same scaling and thus converge faster, thereby drastically reducing the training time and improving the speed.

4.2 Effect of Feature Engineering

The tables shown below demonstrate the comparisions of training and testing accuracies on un-standardized and standardized data:

Model	Train accuracy on original data	Train accuracy on featured data
NMC	74.45652173913044%	71.27071823204419%
SVM	92.93478260869566%	92.81767955801105%
Logistic Regression	93.47826086956522%	93.37016574585635%
Random Forest	100%	92.26519337016575%
KNN	89.13043478260869%	89.50276243093923%
MLP	95.1086956521739%	92.81767955801105%

Table 15: Comparision of training accuracies before and after Feature engineering on all models

Model	Test accuracy on original data	Test accuracy on featured data
NMC	78.33333333333333%	73.33333333333333%
SVM	88.33333333333333%	91.66666666666666%
Logistic Regression	90.0%	91.66666666666667%
Random Forest	91.66666666666667%	88.33333333333333%
KNN	85.0%	80.0%
MLP	88.33333333333333%	85.0%

Table 16: Comparision of test accuracies before and after Feature engineering on all models

As we can observe from the above tables, the process of feature engineering does not always result in improvement of the accuracy of the model, . In fact, the only model that benefit from feature engineering are SVM and logistic regression. The reduction in accuracy of the other models is due to the underfitting of the dataset, though the engineered dataset follows the general rule of thumb i.e, Number of constraints = (3-10)* degrees of freedom, and this can be attributed to either small size training dataset or because of the presence of large number of features. This has also increased the training time for all the models by quite some margin because of the high feature size. The key take away from the feature engineering is that it is only useful if the dataset is quite large.

The key reason for not performing feature selection is because, the K-best features that best define the training set may not be the same K-best features for the test or validation sets. To verify my hypothesis, I have selected the 9-best features for the first 160 datapoints and the remaining 24 datapoints of the training dataset and 2 of the 9 features did not match in both the cases. To further bolster of my hypothesis, the actual validation and test accuracies have dropped by around 2.2% for the SVM, 1.5% for logistic regression, 3% for Random Forest and 4.7% for KNN models.

In conclusion, among all the models I have implemented, the SVM, Logistic regression and Random forests have reported the highest test accuracy of 91.67%. The model that gave the best test accuracy and F-1 score is the random forest classifier that achieved test accuracy of 91.67% along with the highest F-1 score of 0.8936.

5 Libraries used and what you coded yourself

In this project, I have designed, implemented, and evaluated 7 models namely, Nearest Means Classifier (Baseline system), Support Vector Machines, Logistic Regression, Random Forest, KNN, Multi-layer perceptron, and a trivial classifier on 3 types of datasets in total i.e, the original dataset, the standardized dataset, and the dataset with feature engineering. I used K-fold cross-validation where I have split my training data into train and validation data to train my model on the train data and validate it on validation data, along with grid search for model selection, and selected the model with optimum average validation accuracy across all folds, to evaluate it on the test dataset. I have reported the performance metrics: Accuracy, Confusion matrix, and F1 score on all the training and testing datasets mentioned above. The model that gave the best test accuracy and F-1 score is the random forest classifier that achieved test accuracy of 91.67% along with the highest F-1 score of 0.8936.

I have coded the following functions myself:

1. $addndayavg()$: Computes the rolling averages of the previous $ndays$ of a feature from the dataset and removes the last n_days of data and adds this feature to the dataset. I have coded this from scratch and have obtained the featured dataset with 18 fetures excluding Date and Classes.
2. $evaluate()$: It takes the model, train dataset and test dataset, and performs evaluation of the model and returns the accuracy and predicted classes of the test set.
3. K-fold cross validation and grid-search functions: I wrote the codes for K-fold cross-validation and grid-search myself and separately for each model.
4. $Normalize()$: Standardizes the train and test datasets to zero mean and unit variance. I have used StandardScaler() from sklearn to fit the datasets but have coded the processing section by myself.

The libraries I have used are: numpy, pandas, math, sklearn, seaborn (to plot the heatmap of confusion matrix).

I have used the following models from sklearn library : NearestCentroid, KNeighborsClassifier, LogisticRegression, RandomForestClassifier, MLPClassifier, DummyClassifier. Apart from this, I have used other sklearn functions to compute confusion matrix, accuracy scores. However, as mentioned earlier, I have coded the loops for grid-search and K-fold cross validation by myself and have used these for model selection.

6 Summary and conclusions

Among all the models implemented in this project, the random forest classifier has achieved highest test accuracy of 91.67% along with the highest F-1 score of 0.8936. However, I believe that with better stratergies like Bayesian Optimization techniques for model selection, the models can be fine-tuned further, and thereby giving higher accuracies. However, one caveat of using Bayesian Optimization techniques is the high training times for optimum selection. So, I believe that pursuing this method would be interesting and useful for obtaining better performance from the model.

References

[1] *Cumulative area burned by wildfires in Algeria from June 2021 to August 2021, with average for 2008 to 2021*, available at `https://www.statista.com/statistics/1281357/algeria-area-burned-by-wildfire-per-weak/`

[2] *Algeria suffers from devastating wildfires, but faces big challenges in addressing them*, available at `https://theconversation.com/algeria-suffers-from-devastating-wildfires-but-faces-big-challenges-in-addressing-them-166944`

[3] Faroudja Abid, Nouma Izeboudjen, *Predicting Forest Fire in Algeria Using Data Mining Techniques: Case Study of the Decision Tree Algorithm*, in Advances in Intelligent Systems and Computing book series (AISC,volume 1105).

[4] *Nearest centroid classifier*, available at https://en.wikipedia.org/wiki/Nearest_centroid_classifier/

[5] *Platt scaling*, available at https://en.wikipedia.org/wiki/Platt_scaling

[6] *Support vector machine*, available at https://en.wikipedia.org/wiki/Support-vector_machine

[7] *Hesse normal form*, available at https://en.wikipedia.org/wiki/Hesse_normal_form

[8] *Logistic regression*, available at https://en.wikipedia.org/wiki/Logistic_regression

[9] *Random forest*, available at https://en.wikipedia.org/wiki/Random_forest

[10] *K-nearest neighbors algorithm*, available at https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[11] *Decision tree learning*, available at https://en.wikipedia.org/wiki/Decision_tree_learning

[12] *Bootstrap aggregating*, available at https://en.wikipedia.org/wiki/Bootstrap_aggregating