The Effect of Conditioning of Trigonometric Transformations of Dates with Meteorological Data in Forest Fires Prediction : An Experimental Study

Data Set: Algerian forest fires

Sarthak Kumar Maharana
maharana@usc.edu

Shoumik Nandi
shoumikn@usc.edu

05/02/2022

Link to GitHub repository

1 Abstract

In this work, we study the effects of conditioning a trigonometric transformation of dates with meteorological data, that would aid in predicting the occurrence of forest fires in Algeria. We pose this as a binary classification problem. At first, we experiment by computing statistical parameters of 'Temperature', 'Wind Speed (Ws)', 'Humidity (RH)', and 'Rain', by sliding windows of lengths 2, 3, and 7 days. We also condition the "encoded" dates, especially the cosine of the days, with the best features obtained. Experimental results reveal that an optimal sliding window of size 3 days, is beneficial to obtain good predictions. We train various machine learning classifiers to study the effect of our hypothesis. Support Vector Machines (SVM), with a linear kernel and a regularisation parameter of value 5, without any conditioning, gave the best results with a test accuracy of 0.916 and F1 score of 0.893, and a relative improvement of 5.84% in accuracy and 7.2% in F1 score, over a nearest-means baseline classifier. The corresponding SVM model conditioned with the transformed dates gave a relative improvement of 3.92% in classification accuracy, over the baseline.

2 Introduction

Forest fires are disasters of nature which cause a lot of economical loss to wildlife as well as human life. Hence, it is very important to try to control them as much as possible. For this purpose, it might be useful to predict whether there would be a fire in a particular region on a particular day or not, to be able to control it or possibly even avoid it altogether. This work aims to predict if a forest fire will occur or not given different weather data observations such a Temperature, Rain, Relative Humidity, Wind Speed and Fire Weather Index (FWI) components such as Fine Fuel Moisture Components (FFMC), Duff Moisture Code (DMC), Drought Code Index (DC), Initial Spread Index (ISI) and Buildup Index (BUI). We are also given the dates of these observations. We are given data regarding all the above components from the beginning of June 2012 to the end of August 2012 and with the information of whether a fire has occurred or not on the given day, We are expected to predict the occurrence of a fire/no fire during the month of September 2012.

2.1 Problem Assessment and Goals

We work on this project, with an aim of understanding the importance of extraction of date features, namely the trigonometric transformation / encoding of a "date" , which would assist us in doing binary classification by classifying if there's a fire or not. We hypothesise that such an effect could be beneficial. Our experiments are two-fold. At first, we condition the extracted transformation of the dates, with the features of the 'un-processed' train set. Next, we analyse the same effects on a feature-engineered train set by conditioning the best features with the transformed dates and vary the sliding window size while calculating the past statistics. In this, we also have another objective of choosing an optimal window size. In essence, for a certain data point, we aim to see how much of "past" data is crucial in accurate binary classification.

And so, the rest of the report is divided as follows : We first discuss the usage of the datasets for the two sets of experiments, along with the 5-fold cross validation setup. We present few data analysis plots and trends we observed, followed by discussion of the preprocessing step and feature engineering. Next, we discuss about the feature dimensionality adjustment methods, training, model selection, and results.

2.2 Literature Review

Paulo Cortex et al., in [1], predict forest fires as fast as possible using a data mining approach. They used five different data mining techniques along with four distinct feature set ups (FWI components, spatial, temporal and spatial attributes) to carry out their predictions. The FWI components consisted of 'ISI', 'FFMC', 'DC', 'DMC', 'BUI'. The four weather components consisted of 'Rain', 'Temperature', 'Relative Humidity' and 'Wind'. They found out that for the FWI components, there was a massive correlation between 'BUI' and 'DMC', and 'BUI' and 'DC'. As a result, they decided to drop 'BUI' which yielded better results. They carried out experiments using four different setups : STFWI - spatial, temporal and the four FWI components, STM - spatial, temporal and four weather conditions, FWI - four FWI components and M - four weather components. The best results were yielded by the configuration that used a SVM and the four weather inputs. In [2], the aim was to see if there will be a fire or not, thus treating it as a binary classification problem. The dataset included FWI components and meteorological data such as rain, relative humidity, temperature and wind speed. The aim was to integrate a decision tree classifier as a part of the smart sensor node architecture that allows fire prediction in automated and intelligent ways without human interaction. On studying the features they concluded that rain did not have an effect on the prediction. They also concluded that the best features for classification are temperature, relative humidity and wind speed. The testing was done using simple Decision Tree (DT), Random Forest, Boosting (AdaBoostM1) and Bagging classifiers. The results they got showed that boosting of decision tree (AdaBoostM1) gave the best results in terms of recall, accuracy and precision. However, boosting DT needed significant resources and effort to translate to hardware implementation. As a result simple DT was chosen for hardware implementation.

3 Dataset Description

This work was carried out on a dataset that houses meteorological data and their related parameters of a region located in the northeast of Algeria. The dataset has been split into train and test sets where the train set has 184 data points and the test set has 60 data points. We've been provided with the following features for both the sets, namely, Date, Temperature, RH (Humidity), Ws (Wind Speed in km/h), Rain, FFMC, DMC, DC, ISI, BUI, and Classes. The crucial weather parameters include Temperature, RH , Ws, and Rain. FFMC, DMC, DC, ISI, and BUI are derived from the previous weather parameters. FFMC, in particular, was derived from the weather parameters (Temperature, RH , Ws, and Rain) using the Fire Weather Index Simulator [3]. The train set spans from the 1st of June, 2012 ('01/06/2012') up until the 31st of August, 2012 ('31/08/2012'). Similarly, the test set has data for the entire month of September 2012 i.e., from '01/09/2012' to '30/09/2012'. And so, for each date in 'Date', we have access to two data points with it's corresponding features. The 'Classes' column is a representation of 'Fire' and 'No Fire' encoded as 1 and 0, respectively, making this entire work as a binary classification problem.

From the 'Classes' column of the train set, we observe that there are 115 instances of 'Fire (1)' and 69 instances of 'No Fire (0)'. Similarly, for the test set, there are 23 and 37 instances of 'Fire (1)' and 'No Fire (0)', respectively. This makes the train and test sets 'slightly' imbalanced with an approximate 60-40 split of the classes. However, we don't address this issue in our work since we believe the dataset is not too imbalanced and our approach should work fine.

4 Approach and Implementation

4.1 Dataset Usage

As mentioned earlier in Section 2.1, we run two folds of experiments. The details of the dataset usage are given below:

4.1.1 Experiments with the original train set

At first, we employ a trigonometric transformation of the 'Date' column and condition it with the 'un-processed' train dataset. The number of training data points is equal to the length of the train set i.e., 184. We dropped the 'BUI' feature since it had a high Pearson correlation coefficient with 'DC', 'DMC', 'ISI', and 'FFMC'. The details behind this reasoning are described later. We also drop 'Date'. Hence, the feature matrix either had a shape of (184, 8) or (184, 9) depending on whether we add the date transformation column or not, to see its effects. The test set was untouched, and had a total of 60 data points.

4.1.2 Experiments with the feature-engineered train set

In this set of experiments, we vary the length of the sliding window by either 2 / 5 / 7 days i.e we collect statistics of the past data with the aim of analysing it's effect on the binary classification. Alongside this, we also experiment by conditioning the feature matrix with the trigonometric transformation of 'Date'. To avoid any leakage of the train data onto the test set, we remove data of the last few days. Table 1 summarises the lengths of the train set, depending on the length of the sliding window.

Window length	**Total data points**
2 days	180
3 days	178
7 days	170

Table 1: Variation of the length of the train set with respect to the sliding window size.

4.2 k-fold Cross Validation (k = 5)

For all of our experiments mentioned earlier, we employed a 5-fold cross-validation (CV) system to estimate the performances of our classifiers on unseen data and abet us in model selection. Here, we made sure the length of the train data was an integer multiple of five, and so the data was divided into five equal subsets. As an example, for a length of 184, we discarded the last 4 data points to make the length equal to 180 with a subset-length of 36. This recipe was followed for the other lengths as well. For a clear explanation, since we're dealing with a time-series data and predicting the future, it's crucial for us to not use the conventional k-fold CV strategy [4]. However, since the dataset is of low-resource, employing such a strategy worked out fine. The setup was inclusive of splitting up of the "new" length of the train data into five equal subsets [1, 2, 3, 4, 5]. Table 2 demonstrates the chaining for each fold accordingly. We made sure that there was no data leakage between the train and validation sets. In essence, if a validation set followed a train set, data of the last "x" days (2/3/7) from the train set was removed. The same method was used if a train set followed a validation set. Our target was to make sure that the validation set was of equal length across all the folds, and that the model generalised well to "unseen" data when doing cross-validation. We coded up a single function that would accept, as an argument, the train set, perform the steps mentioned earlier, and save the corresponding new train and validation data as .csv files per fold. This ensured that there was no overlap between the sets. We then call the respective train/val .csv files as and when required.

Fold number	Train	Validation
1	[2, 3, 4, 5]	[1]
2	[1, 3, 4, 5]	[2]
3	[1, 2, 4, 5]	[3]
4	[1, 2, 3, 5]	[4]
5	[1, 2, 3, 4]	[5]

Table 2: 5-fold cross validation strategy

On similar lines, we added the same set of feature preprocessing steps (discussed later), like that on the train set, to the test set. However, the length of the test was untouched and was fixated to 60. We observed the average validation accuracy, across all the 5-folds, and fine-tuned the model parameters until there was a reasonable performance change. After performing model selection based on the strategy mentioned above, the model, with the best hyperparameters, was fit on the entire training data and was evaluated on the test set to obtain the evaluation metrics (F1-score, accuracy, and confusion matrix) and analyse the model performance.

4.3 Exploratory Data Analysis

In this subsection, we analyse and present the data analysis techniques we used on the original training dataset. This is one of the first steps and sanity checks that must be done when solving a machine learning problem. It helps us in analysing the trends and patterns that are present in the training data. This way, the required feature preprocessing steps are taken. And, since, here, we're dealing with time-series data, it is important to collect statistical data (mean / maximum / minimum / median) over a set of past data points, for a current data point, that would help in better classification. It helps in drawing conclusions about data changes over time.

4.3.1 Features vs Classes

We present the results of plotting the real-valued features against the classes (0/1). We aim at understanding the variation of features and their influences on the occurrence of fire.

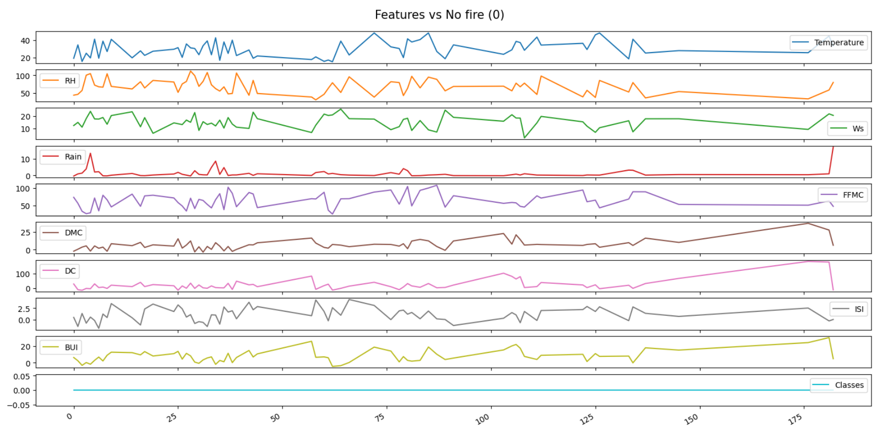


Figure 1: Features vs No fire occurrence

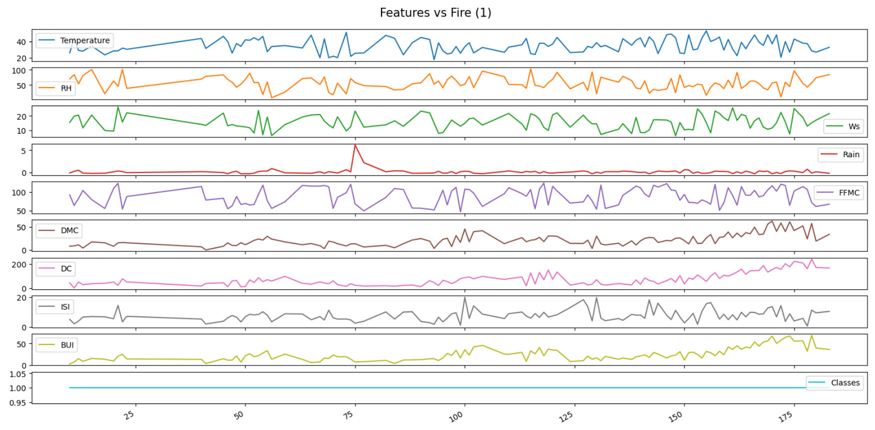


Figure 2: Features vs Fire occurrence

Figures 1 and 2 illustrate the changes of features for each class i.e when there's no fire and when there is, respectively. We observe that over the passage of time that there are abrupt changes of data as time passes by, for each feature. It is also important to observe that there are appearances of continuous spikes for about 140 data points and then there's a very steady curve after, in Figure 1. This means that data points begin to smoothly transition towards the end of August. And so, an important statistical property to be exploited is the maximum value of the features, for a given window length. In Figure 2, we continue to see rapid changes in data. As an example, for both the figures, we observe that for 'Rain', changes hover over zero over time but with steady spikes at uniform intervals.

4.3.2 Histograms of features

Figure 3 illustrates the normalised histograms of all the features with an estimated probability density function (pdf) of the continuous data variables, generated using the seaborn library in Python. Here, along with the existing real-valued variables, we treat the 'Date' feature as such and plot it too to understand the trend it follows over time. In all the subplots, data points with class 0 are at a higher scale. We observe that 'DMC', 'DC', 'ISI', and 'FFMC' are slightly left -skewed with 'Rain' being heavily skewed to the left.

4.3.3 Scatter Plot

Figure 4 illustrates the scatter plots of all the features with respect to the two classes i.e., No fire (0) and Fire (1). We observe that there is a linear relationship between the data of each of the classes per feature.

4.4 Preprocessing

As a first step, we address the issue of finding a suitable transformation / encoding of the dates of the train and test sets. The format of the data was given as "DD/MM/YY", where DD is the day, MM is the month, and YY is the year (fixed at 2012). So, it was important to represent the day and month. Hsu et al. in [5] suggest to perform a *1-of-C* encoding. However, we make use of a continuous trigonometric transformation [6] to preserve the cyclical properties and model seasonal trends. The motivation is that, for days in a month, 31 comes just before 1, and so they're placed very close to each other. Similarly, the months appear cyclically. January follows December. Hence, it is important to preserve this cyclical property and so such a transformation is done to project all the days and months onto a circle. We use the following mathematical expressions to transform

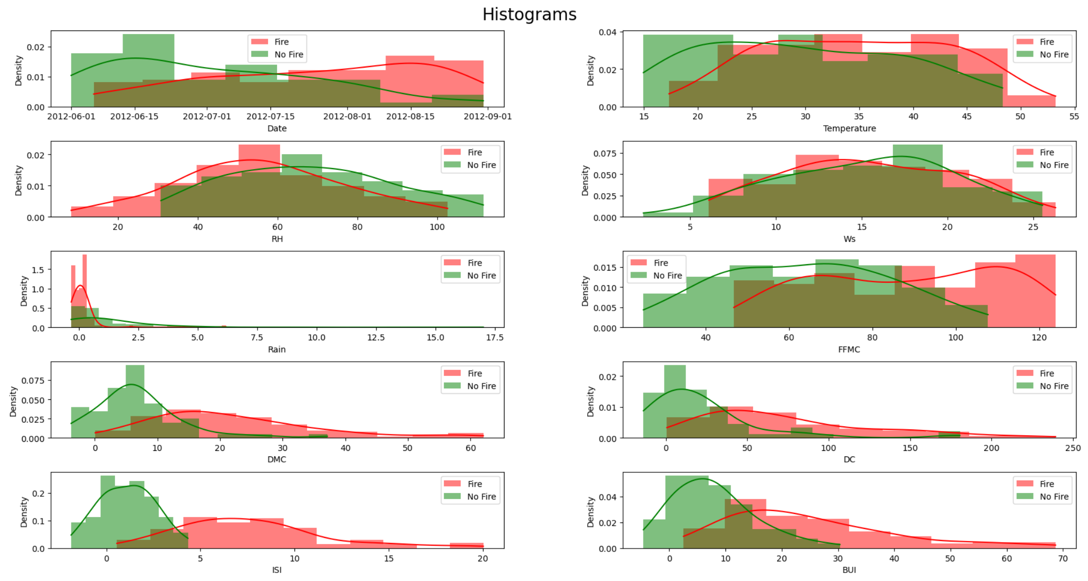


Figure 3: Histograms of features

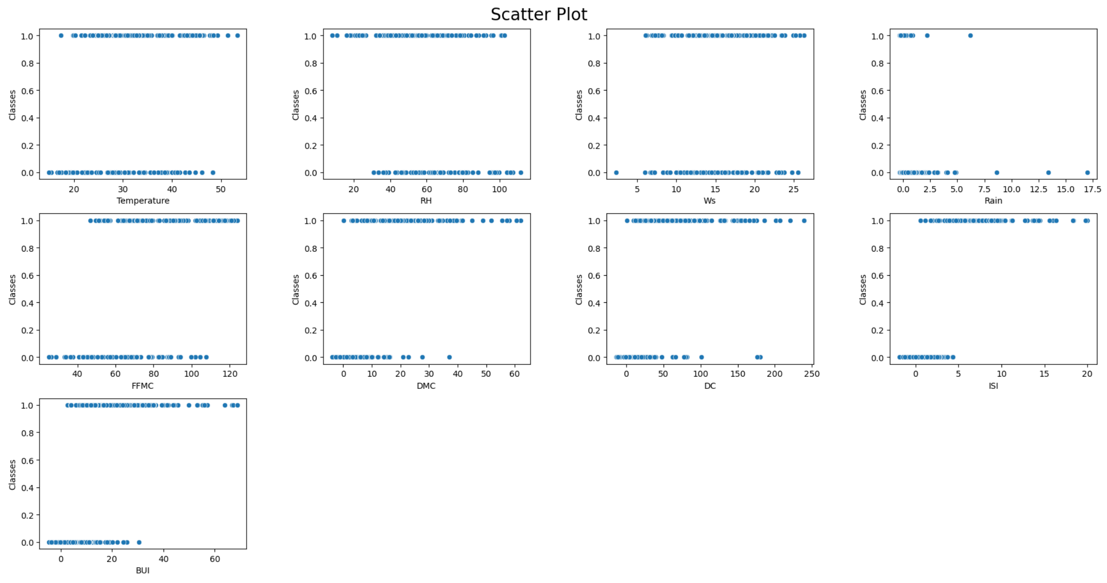


Figure 4: Class-wise data distribution (Scatter Plot)

each day and month from the 'Date' column:

$$sin(2\pi * x[idx]/max(x)) \tag{1}$$
$$cos(2\pi * x[idx]/max(x)) \tag{2}$$

, where x could be a day/month vector and idx is the pointer. Day ranges from 1 to 31 and month ranges from 6 (June) to 9 (September) for the train and test sets. Hence, we obtain two vectors of all the days, representing the sine (day-sin) and cosine (day-cos) transformed values. The same applies to the months as well (month-sine, month-cos). Clearly, there's no variability in the values for the "encoded" months.

As it can be seen from Figure 5, the scatter plot of the sine and cosine transformation of all the days follows a cyclical property. The same figure applies to the months as well. In this way,

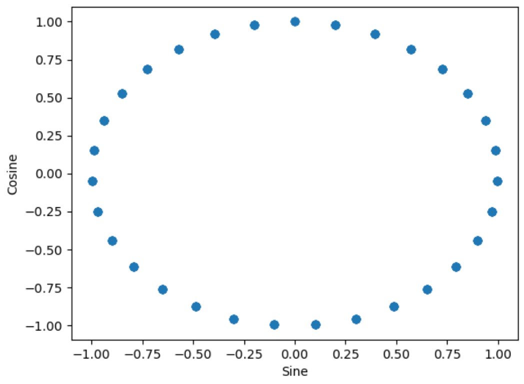


Figure 5: Scatter plot of the sine and cosine transformation of all the days in the 'Date' column of the train set.

we project all the values between -1 and +1, and obtain two time-series features each, for days and months. Our experimental setup is such that we plan on conditioning these newly obtained time-series feature to the features of the train set, and study model performances for the binary classification task.

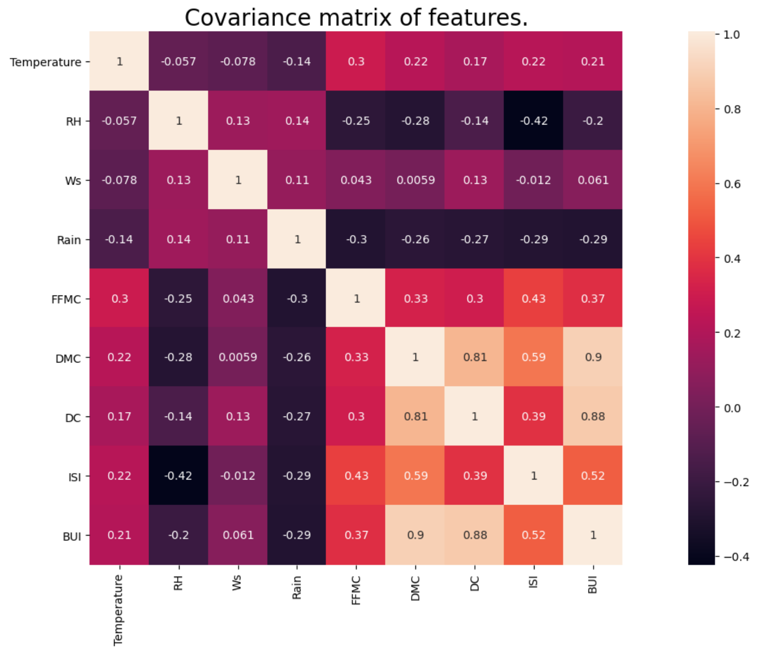
4.5 Feature engineering

In this section, we address the methodology we used to develop new relevant statistical features. To clearly explain, we compute the mean, maximum, minimum, and median of each of the four weather variables namely 'Temperature', 'RH', 'Ws', and 'Rain'. We also define a variable-sized sliding window to aid in computing the statistical parameters mentioned. We experiment with a window of 2, 3, and 7 days. To compute each of the parameters, we use a sliding window technique. The main objective of such a computation is to model the influence of the past data on the current data point's class. We aim to find an optimal value of the number of past days' data that is required for efficient prediction.

At first, we concatenate the given train and test sets before we begin computation. For example, to compute the mean of 'Temperature' for a window of 3 days, for the first two data points of a certain date, we assign 0. For the next day, we compute the mean of the previous day i.e, mean of the last two data points of the previous day. For the second day, we compute the average of the last four data points. For the third day onwards, we employ the window of 3 days, slide it, and compute the mean as we traverse through the entire feature. We make sure that for each date, which has two data points, the mean is the same. This entire line of thought is extended to calculating other statistical parameters of the other weather variables. We later remove data of the last "x" days from the train set since the last "x" days will be used for the statistical computation of the first data point of the test set. We do so by keeping a pointer at the start index of the test set, and then remove the data accordingly. After all the possible combinations, we add up 16 new statistical features.

We know that, while solving a machine learning problem, it is important to have linear independence / less correlation between the features, depending on the problem statement at hand. If not, this would lead to redundancy and lower performance of the model. And hence, we did a quick sanity check by plotting the normalized covariance matrix of all the real-valued features.

Figure 6 is an illustration of the covariance heatmap of all the features. The diagonal entries represent the covariance and the off-diagonal elements represent the variance between two features. Since there is a direct relationship of the covariance and the correlation, between two features X and Y, according to the formula given below,



	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI
Temperature	1	-0.057	-0.078	-0.14	0.3	0.22	0.17	0.22	0.21
RH	-0.057	1	0.13	0.14	-0.25	-0.28	-0.14	-0.42	-0.2
Ws	-0.078	0.13	1	0.11	0.043	0.0059	0.13	-0.012	0.061
Rain	-0.14	0.14	0.11	1	-0.3	-0.26	-0.27	-0.29	-0.29
FFMC	0.3	-0.25	0.043	-0.3	1	0.33	0.3	0.43	0.37
DMC	0.22	-0.28	0.0059	-0.26	0.33	1	0.81	0.59	0.9
DC	0.17	-0.14	0.13	-0.27	0.3	0.81	1	0.39	0.88
ISI	0.22	-0.42	-0.012	-0.29	0.43	0.59	0.39	1	0.52
BUI	0.21	-0.2	0.061	-0.29	0.37	0.9	0.88	0.52	1

Figure 6: Covariance heatmap of the original train set.

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3}$$

where the covariance (cov(X, Y)) is computed as:

$$cov(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \mu_x)(Y_i - \mu_y) \tag{4}$$

where ρ_{XY} is the correlation between X and Y, σ_X and σ_Y are the standard deviations of X and Y respectively, μ_x and μ_y are the sample means of X and Y, N is the total data points, we observe that 'BUI', in the last row, has a high correlation with 'DMC', 'DC' and 'ISI', and so we decided to drop 'BUI' out of all our experiments [1].

As mentioned earlier, we run two sets of experiments. As a preliminary experiment, we wish to understand the effects of the "encoded" days and months on the original train and test set. Here, we don't employ any sliding window method i.e the window size is 0. Later, after feature engineering and reducing the total number of real-valued dimensions $D(9)$ to $D^{`}$ to avoid the curse of dimensionality [7], where $D^{`} < D$, we experiment with conditioning the "encoded" days and months with the feature matrix, and also study the amount of past data required for computation of the statistics of the current data by varying the number of "x" (2/3/7) days.

We also perform a min-max normalisation on the obtained train features after feature engineering. The reason behind adopting such a normalisation method was to scale the values of all the features, between 0 and 1, by preserving the information and data distribution since we found a wide range of values between the features, and so, comparing them on a similar scale is better. We use the following formula for normalising a feature "A":

$$A_{norm} = \frac{A - min(A)}{max(A) - min(A)} \tag{5}$$

However, a major caveat of such a normalisation scheme is that, it will encounter an out-of-bounds problem if a certain normalised test data point falls outside the range of A_{norm}. However, the values in the test set were similar to the train set, and so we didn't encounter any issue.

4.6 Feature dimensionality adjustment

Feature dimensionality adjustment is done to remove all irrelevant and redundant features that would affect model performances. It is also crucial for the total number of data points (N) > (3-10)*(d.o.f), where d.o.f refers to the degrees of freedom i.e., the independent parameters that are learned while a model is trained. If that's not the case, it would lead to overfitting of a model i.e the model would learn the data patterns and noise of the train set so well that it would fail to generalise when evaluated on an "unseen" set.

After normalisation, we employ the following feature selection / dimensionality adjustment methods to select the best features from the available real-valued 8 features (after dropping 'Date' and 'BUI'). We report the results of each of the methods, after giving a brief introduction.

4.6.1 Pearson Correlation Coefficient (PCC)

PCC is a statistical test which measures the association and relationship between two variables. It works by measuring the relationship between the two chosen variables based on covariance (Refer Equation 3). It gives us information of the extent of the relationship using a magnitude and the direction of the linear relationship: positive being if one increases then the other one increases and negative being if one increases the other one decreases. The coefficient values of PCC range from +1 to -1 . If the value is near +1 or -1 it is said to be a perfect correlation. However, if the value is 0, then it is said to have no correlation. Correlation of coefficients are symmetric between two variables i.e. the relation of A and B or B and A will be the same. The correlation does not depend on the unit of measurement. And so, for feature selection, it is equally important to consider the positively and negatively correlated features with the target variable [8]. We coded our own implementation of the PCC.

4.6.2 Principal Component Analysis (PCA)

PCA is used to reduce the dimensionality of the feature space and for feature extraction while maintaining the most relevant information. It finds the direction of maximum variance in high dimensional data and usually projects the data onto a new subspace of lower dimensionality [10]. This way it searches for the components that preserve the maximum amount of variance and information without any loss of generality. To compute the PCA, we perform the following steps:

1. Construct the covariance matrix (Equation 4) and decompose it to find the corresponding eigenvalues and eigenvectors. The eigen values are found by solving (X - λI) = 0, for λ and then resubstituting back to obtain the eigen vectors.

2. Sort eigenvalues in decreasing order along with their corresponding eigenvectors.

3. Select k eigenvectors depending on the required dimensionality k which is less than the original dimensionality of the data.

4. Construct a projection matrix from the k selected eigenvectors.

5. Transform the pre-processed features into k dimensional features using the projection matrix.

In addition, we store all the sorted variances of each principal component (transformed features) and plot them against the total number of features. This gives us a rough idea of how many features are actually required for a good model performance.

Figure 7 illustrates the contribution of maximum variance by each principal component (after PCA) against the total number of features. We observe that the first principal component contributes about 25.96% of the total variance, followed by a contribution of 17.26% of the second, and so on. We

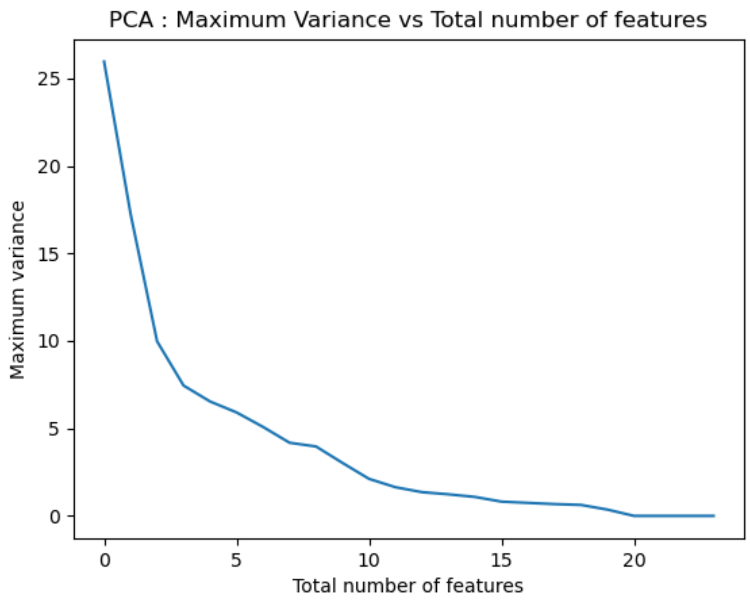


Figure 7: Maximum variance of principal components vs total features

keep a threshold of 6 features as the best number of features, where the 6^{th} component contributes 5.91% of the variance. There's minimal contribution after this. We also keep an addition of 2 features, when we run experiments of conditioning the "encoded" days and months to the feature matrix. We coded our own implementation of the PCA.

4.6.3 Mutual Information (MI) (Non-EE559)

Another important feature selection method to understand the relationship between features and the target is by the computation of mutual information. Here, we compute the drop in the uncertainty of a variable provided we know the value of the other variable [9]. The equation of the mutual information is:

$$I(X|Y) = H(X) - H(X|Y) \tag{6}$$

where, I(X|Y) is the mutual information between a feature X and the classes Y, H(X) is the entropy for X and H(X|Y) is the conditional entropy. To calculate the entropy, the following formula is used:

$$H(X) = -\sum_{i=1}^{N} P(X(i))log_2(P(X(i))) \tag{7}$$

where, P(X(i)) is the probability of element i in X. The values obtained for mutual information range from 0 to 1, where anything close to 1 represents high amount of dependence and mutual information exchange between the feature and target, and a value close to 0 relates to independence between the two. We use sklearn's feature_selection.mutual_info_classif to get the mutual information score. We then plot and print out the sorted scores based on the column headers.

4.6.4 Recursive Feature Elimination (RFE)

In RFE, we set a target for the desired number of features. It considers all features in the training set and removes features until the desired number of features remain. This is achieved by training a selected simple model with all the features, ranking the features by importance, removing the least important feature and re-fitting the model. The process is repeated till the targeted number of desired features remain [11]. In our work, we chose a Support Vector Machine (SVM) as the simple model. Here, we set the maximum desired number of features as 7. We make use of sklearn's feature_selection.RFE for our work and passed sklearn's SVM model and set n_features_to_select = 7.

4.6.5 Sequential Feature Selection (SFS)

SFS adds the best features based on the performance of a model. We set the desired number of features. In this method we use the performance of the model and not the feature importance [12]. Here too, we use an SVM as the linear model with the desired number of features being 7. There are two variants of SFS, namely the 'forward' and 'backward'. SFS perform a greedy search over all the features and hence, is time consuming. We employ the 'forward' variant and list out the steps as follows:

1. Choose a simple model. Here, we use the SVM classifier.
2. For each feature,
   (a) Add the feature.
   (b) Fit the model.
   (c) Find the performance.
3. Add the feature that scores the best metric in 2(c).
4. Repeat 2-4 until the desired number of features is reached.

We use sklearn's feature_selection.SequentialFeatureSelector and passed sklearn's SVM model, set n_features_to_select = 7, and chose the 'forward' mapping .

4.6.6 Analysis of Variance (ANOVA)(Non-EE559)

ANOVA is a statistical method, used to check the means of two or more groups that are significantly different from each other [13, 16]. It assumes the hypothesis as

1. Null hypothesis H_0 : Means of all the groups are equal.
2. Not null hypothesis H_1 : At least one of the means of the groups is different.

When the null hypothesis is rejected it means that variance exists between group and thus the feature has an impact. Here, we calculate the F value and using the F value and the F distribution, we perform the F tests. The F value is calculated as the ratio of the between-groups variance to the within-group variance.

The distribution curve of F distribution displays the likelihood of the F values [15]. Using a significance level which is usually 0.05 i.e ($p < 0.05$), we can reject the null hypothesis. The significance level is a measure of the strength of the evidence that must be present in your sample before you will reject the null hypothesis and conclude that the effect is statistically significant [14]. We use sklearn's feature_selection.SelectKBest and set k = 7 (to get the best 7 features).

In the next few subsections, we exploit fitting a couple of classical machine learning classifiers on the train set that have proven well to deciding a good predictive performance. Through this, we can decide upon how much each feature is contributing to the performance and that will aid in choosing the best features for our work.

4.6.7 Decision Trees (DT)(Non-EE559)

Decision Tree is a supervised learning algorithm and can be used for both regression and classification. For our work, we use it for the purpose of classification [17]. A DT creates a model that predicts the class of the target variable by learning decision rules with the help of the training data. The root node of the tree represents the entire set of data. This data gets divided into homogeneous sets. Nodes after which further splitting does not take place are called leaf nodes. The decision tree sorts data as it goes down the tree from the root node to the leaf node. The classification of the datapoint is given at the root node. Each node acts as a test case for some attribute and each edge descending to its subnodes corresponds to the possible answers of the test. This process goes on recursively until a leaf node is reached and the datapoint is classified [18].

To decide which feature to select for a particular node to perform a test on the data, there are attribute selection methods. The two famous criterion functions are entropy and gini impurity. We used the gini impurity in our work.

Gini impurity is calculated based on the gini index which can be viewed as a cost function which is calculated by subtracting the square of probability of each class from one, and favors larger partitions [18].

$$gini = 1 - \sum_{i=1}^{C} P_i^2 \tag{8}$$

Lower the value of gini index , better the criteria for splitting and higher the homogeneity. The depth of the decision can be predefined if we want to. Based on this the tree will stop splitting the data and will give us the classification once the predefined depth is reached [18]. We directly use sklearn's implementation of DecisionTreeClassifier and set the criterion function as 'gini'. We then use the feature_importances_ method to obtain the best features [17]. The code description has been provided.

4.6.8 Random Forests (RF)(Non-EE559)

Random forest is a supervised learning algorithm which is used for both regression and classification. In this work, we are using it for classification. Random forest builds decision trees using different data and takes a majority vote for classification. We can predefine the number of decision trees we want the RF model to use to carry out the classification. It uses an ensemble technique called bagging. Ensemble means combining multiple classifiers and using this collection to carry out prediction. Bagging chooses random datapoints from the dataset with replacement. Each tree is trained independently and results are generated. The final classification is done based on majority voting after combining the results of all the classifiers [20]. The RF model in our work uses the gini impurity for feature selection for all the decision trees in the model. Following are the steps :

1. k random points are taken from all the training dataset.
2. Individual decision trees are constructed from each of the k selected random points.
3. Each decision tree will generate an output.
4. Final class assigned based on majority voting

In our work, we directly used sklearn's ensemble.RandomForestClassifier module and set n_estimators = 10 and the criterion function as 'gini' [19]. The other arguments were left as it is. We then fit the model to the train data, and use the feature_importances_ argument to receive our best features. The code description has been provided.

4.6.9 Choosing of the best features

Now, we present the results of using the feature dimensionality adjustment techniques (except for PCA) on the train data to choose our best set of features. As mentioned earlier, since we are calculating the statistical parameters and optionally conditioning the feature matrix with the ”encoded” days and months, we present three tables by varying the sizes of the sliding window (2/3/7 days). To get the best features, we do a majority voting of the frequently occurring features. As mentioned earlier, in Section 4.5, we added 16 new features alongside the original features (after dropping ’Date’ and ’BUI’). In total, we have 24 features. The results in the tables report the best 7 features for each of the techniques, mentioned earlier.

Feature Dim Reduction Method	**Features (w/ a sliding window of 2 days)**						
PCC	ISI	DMC	DC	FFMC	Rain-mean	Rain-max	Rain-median
MI	ISI	DMC	RH-max	DC	Temperature-max	Rain-min	Rain-max
RFE	Rain-median	Rain-min	ISI	DC	DMC	FFMC	Rain
SFS	Temperature-min	RH	DMC	DC	ISI	Ws	Rain
ANOVA	Rain-median	Rain-max	Rain-mean	ISI	DC	DMC	FFMC
DT	ISI	DC	DMC	Rain-median	Temperature-mean	RH-min	FFMC
RF	ISI	DMC	FFMC	DC	Rain	Temperature	Rain-max

Table 3: Best features obtained from the feature dimensionality adjustment techniques discussed for a window size of 2 days.

Feature Dim Reduction Method	**Features (w/ a sliding window of 3 days)**						
PCC	ISI	DMC	DC	FFMC	Temperature-mean	Temperature-max	Temperature-median
MI	ISI	DMC	RH-min	DC	RH-max	Ws-min	Rain
RFE	Rain-median	Temperature-min	Temperature-max	ISI	DMC	DC	FFMC
SFS	DC	ISI	Temperature-mean	Temperature-min	Ws-max	Rain	Ws
ANOVA	Rain-median	Rain-mean	Temperature-mean	ISI	DC	DMC	FFMC
DT	ISI	DMC	Temperature-max	Rain-min	Ws-mean	RH-min	Temp-median
RF	ISI	DMC	DC	Temperature	FFMC	Rain	Temp-mean

Table 4: Best features obtained from the feature dimensionality adjustment techniques discussed for a window size of 3 days.

Feature Dim Reduction Method	**Features (w/ a sliding window of 7 days)**						
PCC	ISI	DMC	DC	FFMC	Temperature-mean	Temperature-max	Temperature-median
MI	ISI	DMC	DC	Temperature-max	Ws-max	RH-max	Temperature-mean
RFE	Temperature max	ISI	DC	DMC	FFMC	ISI	Rain
SFS	Ws	Rain	DC	ISI	Ws-mean	Ws-min	Rain-median
ANOVA	Temperature-median	Temperature-max	Temperature-mean	ISI	DC	DMC	FFMC
DT	ISI	Temperature-median	DMC	RH-min	Rain-min	Ws-mean	RH
RF	ISI	DMC	DC	Rain	Temperature-max	FFMC	Temperature

Table 5: Best features obtained from the feature dimensionality adjustment techniques discussed for a window size of 7 days.

Tables 3, 4, and 5 report the best seven features that the feature dimensionality adjustment techniques give for a sliding window of sizes 2, 3, and 7 days. Based on a majority voting scheme [21] from the tables, we observe that ’ISI’, ’DMC’, ’DC’, ’FFMC’, ’Temperature-max’, and ’Rain-Max’, occur the most. We limited ourselves to selecting these top 6 features as indicated by Figure 7. The choice of ’Temperature-max’, and ’Rain-Max’ also corroborates with Figures 2 and 1. The plots gave way an important indication that local maximas are crucial in predicting if a certain data point is classified as fire/no fire. With respect to the ”encoded” day and month, we decided to use the ’day-cos’ feature since it had the highest correlation with the target labels.

4.7 Training, Classification, and Model Selection

In the next few subsections, we discuss and give an outline about the machine learning classifiers that we have used in this work.

4.7.1 ML classifiers used

1. Nearest Means (Baseline system) : In the nearest means model, two means are calculated. One for class 1 using the points classified into class 1 and the other for class 0 using the points classified into class 0. The data points are classified based on the Euclidean distance of the "new" data point from the mean. The data point is assigned to the class of the mean with the shortest distance from it.

2. Trivial system : In this model, an "unseen" point is randomly classified into one of the two classes based on the probability of N_0/N and N_1/N respectively, where N_0 and N_1 are the total points of class 0 and 1 in the training dataset and N is the total number of datapoints in the training dataset i.e ($N = N_0 + N_1$).

3. Perceptron : The perceptron algorithm is the simplest type of classifier which is a binary classification machine learning algorithm. The perceptron consists of - one input layer, weights and bias, net sum and activation function. An input vector x is multiplied by corresponding weight vector w and a bias b is added to it in the perceptron. This result is passed through an activation function. The weights indicate the influence of a specific node on the output. The bias helps shift the curve of the activation function up or down [24]. The activation function maps the input between required values. The perceptron produces a binary output. A weighted sum is calculated. This weighted sum is compared with a threshold value, or bias. If the weighted sum is equal to or below the threshold, or bias, the output is 0. If the output is greater than the threshold, or bias, the output is 1 [24].

4. K-Nearest Numbers (KNN) : K-Nearest Neighbors model is a supervised machine learning algorithm which assumes that similar things exist in close proximity to each other. KNN stores all available cases and classifies new cases based on a similarity measure. The algorithm looks for the 'k' closest points in proximity of the point under consideration and then takes a majority vote to decide the class of the point. We used the Euclidean distance to calculate proximity. On finding the 'k' nearest points based on the distance from the point under consideration, we count the number of points belonging to class 0 and class 1. The one with a higher frequency is then assigned to the label under consideration.

5. Gaussian Naive Bayes (NB)(Non-EE559) : The Gaussian Naive Bayes algorithm is a supervised algorithm based on Bayes theorem. However, in this algorithm a naive assumption is made where a condition independence is assumed between every pair of features given the value of the class variable [22]. On using the naive assumption of conditional independence between all the pairs of features and the target variable,

$$P(x_i|y, x_1, ..., x_{i-1}, x_i, ..., x_n) = P(x_i|y) \tag{9}$$

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ...x_n)} \tag{10}$$

$$\tag{11}$$

where, $P(y \mid x_1, ..., x_n)$ is defined as the posterior probability, P(y) is the likelihood, and $P(x_i|y)$ is the prior. For a given input, the denominator can be considered as a constant, and as a result we can further simplify the equation and use it for classification [22].

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \tag{12}$$

or,

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{13}$$

We calculate the prior probability as N$_i$ / N where N$_i$ is the number of data points in class i and N is the total number of data points. And so,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}) \tag{14}$$

where, σ and μ are the standard deviation and mean, respectively.

6. Decision Tree (DT)(Non-EE559) : We also make use of a decision tree as a model. The description of it has been discussed in Section 4.6.7.

7. Random Forest (RF)(Non-EE559) : Random Forests have been discussed in Section 4.6.8.

8. Support Vector Machines (SVM) : The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors [23]. In particular, we were interested in understanding the effect of two kernels of SVM i.e., linear and radial basis function (RBF). With a linear kernel, a single straight line can classify the two classes. However, an RBF kernel makes use of a Gaussian similarity function, with an aim of modelling the similarity between a test point and the training point in a Gaussian function as [23],

$$K(x_1, x_2) = exp(\frac{\|x_1 - x_2\|^2}{2\sigma^2}) \tag{15}$$

After performing pre-processing (Section4.4), feature engineering (Section4.5), and feature selection (Section4.6) for both the experiments, we obtain the new feature matrix and it's labels. We perform a 5-fold cross validation, explained in Section 4.2, on the feature-engineered train set.

4.7.2 Experimental Setup

As proposed earlier, we work on two sets of experiments, named as Experiment 1 and 2. In Experiment 1, we use the original dataset (with no 'Date' and 'BUI') and run experiments to see the effect of conditioning the features with the 'day-cos' feature. In particular, we make use of the cosine of the days since it had higher correlation with the 'Classes'. In Experiment 2, we make use of the best features (as mentioned earlier) after feature dimension adjustment, and experiment with the change in window size and by conditioning the new features with the obtained 'day-cos' feature. We do not shuffle the data in any sense, since it's a time-series dataset.

The following subsection report the average validation accuracy and F1 score per model, across all the 5 folds, that would help in good model selection. The accuracy is calculated as the mean of the number of correct predictions with respect to the target. On the other hand, the F1 score is calculated as $\frac{2*precision*recall}{precision+recall}$, where precision = $\frac{truepositive}{truepositive+falsepositive}$ and recall = $\frac{truepositive}{truepositive+falsenegative}$.

4.7.3 Model Selection

1. Experiment 1: We first present the average validation accuracy and F1 score of this experiment **without** the conditioning of 'day-cos'.

Nearest Means	**Average validation accuracy**	**Average F1 score**
Baseline	**0.85**	**0.811**

Trivial System	**Average validation accuracy**	**Average F1 score**
	0.5611	**0.639**

Perceptron	**Average validation accuracy**	**Average F1 score**
learning rate = **0.01**	**0.728**	**0.7914**
learning rate = 0.1	0.727	0.791
learning rate = 1	0.716	0.785

KNN	**Average validation accuracy**	**Average F1 score**
k =1	0.82	0.824
k = 2	0.844	0.8296
k = 3	0.833	0.8413
k = 4	0.866	0.8615
k = 10	**0.866**	**0.877**

Naive Bayes	**Average validation accuracy**	**Average F1 score**
var_smoothing = 10	0.627	0.748
var_smoothing = 1	0.9	0.906
var_smoothing = **1e-1**	**0.911**	**0.906**

Decision Tree	**Average validation accuracy**	**Average F1 score**
criterion = 'gini' max_depth = 1	0.905	0.907
criterion = 'gini' max_depth = 2	0.905	0.907
criterion = **'gini'** max_depth = **3**	**0.905**	**0.907**

Random Forest	**Average validation accuracy**	**Average F1 score**
n_estimators = 300, max_depth = 1	0.8611	0.871
n_estimators = 300, max_depth = 10	0.905	0.9166
n_estimators = **300**, max_depth = **15**	**0.9055**	**0.9166**

We experiment with different sets of hyperparameters by tuning them heuristically until we reach a good amount of average validation accuracy and F1 score. The best set of hyperparameters have been marked in bold and are used to run all the sub-experiments of Experiment 1. The baseline and trivial classifiers are not tuned since they don't have any hyperparameters. However, we report the average validation accuracy and F1 score.

2. Experiment 2: Here, we present the average validation accuracy and F1 score after considering the best features, with a window size of 2 days, **without** conditioning of 'day-cos'. The same

SVM	**Average validation accuracy**	**Average F1 score**
kernel = 'linear', C = 1	0.911	0.914
kernel = **'linear'**, C = **5**	**0.9277**	**0.9342**
kernel = 'rbf', C = 1	0.844	0.856
kernel = 'rbf', C = 5	0.911	0.9155

set of best hyperparameters are considered when running experiments for 3 and 7 days as well.

Nearest Means	**Average validation accuracy**	**Average F1 score**
Baseline	**0.8611**	**0.8363**

Trivial System	**Average validation accuracy**	**Average F1 score**
	0.577	**0.6563**

Perceptron	**Average validation accuracy**	**Average F1 score**
learning rate = **0.01**	**0.738**	**0.805**
learning rate = 0.1	0.711	0.787
learning rate = 1	0.722	0.793

KNN	**Average validation accuracy**	**Average F1 score**
k = 1	0.883	0.891
k = 2	0.905	0.8912
k = 3	0.888	0.892
k = 4	**0.905**	**0.9**

Naive Bayes	**Average validation accuracy**	**Average F1 score**
var_smoothing = 10	0.627	0.748
var_smoothing = 1	0.91	0.918
var_smoothing = **1e-1**	**0.911**	**0.913**

Decision Tree	**Average validation accuracy**	**Average F1 score**
criterion = 'gini' max_depth = 1	0.905	0.907
criterion = 'gini' max_depth = 2	0.844	0.785
criterion = **'gini'** max_depth = **3**	**0.905**	**0.912**

This subsection's main objective was to choose the best hyperparameters of all the classifiers to conduct Experiment 2. We heuristically perform this until we achieve a good amount of

Random Forest	Average validation accuracy	Average F1 score
n_estimators = 100, max_depth = 1	0.877	0.8857
n_estimators = 100, max_depth = 2	0.9222	0.9186
n_estimators = **100**, max_depth = **5**	**0.9222**	**0.9297**

SVM	Average validation accuracy	Average F1 score
kernel = 'linear', C = 1	0.911	0.9129
kernel = **'linear'**, C = **5**	**0.9388**	**0.9356**
kernel = 'rbf', C = 1	0.872	0.88
kernel = 'rbf', C = 5	0.922	0.9212

average validation and F1 score. The baseline and trivial classifiers are not tuned since they don't have any hyperparameters. However, we report the average validation accuracy and F1 score.

5 Analysis: Comparison of Results, Interpretation

In the next two subsections, we present the results of Experiments 1 and 2 by reporting the accuracy, F1 score and the confusion matrix on the **test set** after training on the whole train set (train + validation), based on the respective best model hyperparameters obtained in the previous section.

5.1 Results of Experiment 1

Classifiers	**Without 'day-cos' conditioning**			**With 'day-cos' conditioning**		
	Accuracy	**F-1 score**	**Confusion matrix**	**Accuracy**	**F-1 score**	**Confusion matrix**
Nearest-means (baseline)	0.833	0.8214	[[27 10] [0 23]]	0.85	0.8235	[[30 7] [2 21]]
Trivial System	0.566	0.566	[[17 20] [6 17]]	0.566	0.566	[[17 20] [6 17]]
Perceptron	0.683	0.707	[[18 19] [0 23]]	0.766	0.766	[[23 14] [0 23]]
KNN	0.8166	0.8	[[27 10] [1 22]]	0.7833	0.7719	[[25 12] [1 22]]
Naive Bayes	0.8166	0.807	[[26 11] [0 23]]	0.85	0.836	[[28 9] [0 23]]
Decision Tree	0.85	0.8235	[[30 7] [2 21]]	0.85	0.823	[[30 7] [2 21]]
Random Forest	0.7833	0.7797	[[24 13] [0 23]]	0.8	0.793	[[25 12] [0 23]]
SVM	**0.85**	**0.856**	[[**28 9**] [**0 23**]]	**0.86**	**0.852**	[[**29 8**] [**0 23**]]

Table 6: Test accuracy, F1 score, and confusion matrices of all the classifiers used for Experiment 1.

Table 6 reports the test accuracy, F1 score, and confusion matrices on all the classifiers used to run Experiment 1. In this, we had the aim of understanding the effect of conditioning the "encoded" day-month representation, especially the 'day-cos' feature (since it had higher correlation with the target), to the original input features (after removing 'Date' and 'BUI') that would lead to better predictions of fire/no fire. Without the conditioning, we observe that SVM gives the best results with an improvement of 2.04% over the baseline, and about 50.17% over the trivial . We also observe that the Gaussian Decision Tree gives better results over the nearest-means (baseline). The Random Forest (RF) model was expected to give a lower score because it is a data-driven model. However, on the other hand, on conditioning the features with 'day-cos', we do see improvements / no changes over respective model performances, except for KNN. Hence, our hypothesis of conditioning the features with important 'day' feature is giving shape and yielding slightly better results. SVM achieved an improvement of 1.17% in terms of accuracy, over the vanilla set of features.

5.2 Results of Experiment 2

Classifiers	**Without 'day-cos' conditioning (2 days)**			**With 'day-cos' conditioning (2 days)**		
	Accuracy	**F-1 score**	**Confusion matrix**	**Accuracy**	**F-1 score**	**Confusion matrix**
Nearest-means (baseline)	0.866	0.826	[[33 4] [4 19]]	0.866	0.826	[[33 4] [4 19]]
Trivial System	0.566	0.566	[[17 20] [6 17]]	0.566	0.566	[[17 20] [6 17]]
Perceptron	0.6	0.65	[[13 24] [0 23]]	0.55	0.63	[[10 27] [0 23]]
KNN	0.8833	0.851	[[33 4] [3 20]]	0.9	0.88	[[32 5] [1 22]]
Naive Bayes	0.866	0.8461	[[36 1] [7 16]]	0.866	0.8461	[[36 1] [7 16]]
Decision Tree	0.866	0.8461	[[30 7] [1 22]]	0.85	0.823	[[30 7] [2 21]]
Random Forest	0.7833	0.7797	[[24 13] [0 23]]	0.7833	0.7796	[[24 13] [0 23]]
SVM	**0.9**	**0.888**	[[**32 5**] [**1 22**]]	**0.9**	**0.88**	[[**32 5**] [**1 22**]]

Table 7: Test accuracy, F1 score, and confusion matrices of all the classifiers (for a window size of 2 days) used for Experiment 2.

Classifiers	**Without 'day-cos' conditioning (3 days)**			**With 'day-cos' conditioning (3 days)**		
	Accuracy	**F-1 score**	**Confusion matrix**	**Accuracy**	**F-1 score**	**Confusion matrix**
Nearest-means (baseline)	0.866	0.833	[[32 5] [3 20]]	0.866	0.826	[[33 4] [4 19]]
Trivial System	0.566	0.566	[[17 20] [6 17]]	0.566	0.566	[[17 20] [6 17]]
Perceptron	0.466	0.589	[[5 32] [0 23]]	0.68	0.707	[[18 19] [0 23]]
KNN	0.85	0.80	[[32 5] [4 19]]	0.85	0.80	[[32 5] [4 19]]
Naive Bayes	0.9	0.8571	[[36 1] [5 18]]	0.9	0.8571	[[36 1] [5 18]]
Decision Tree	0.866	0.8461	[[30 7] [1 22]]	0.866	0.8461	[[30 7] [2 21]]
Random Forest	0.8	0.793	[[24 13] [0 23]]	0.7833	0.7796	[[25 12] [0 23]]
SVM	**0.9166**	**0.893**	[[**34 3**] [**2 21**]]	**0.9**	**0.8571**	[[**36 1**] [**5 18**]]

Table 8: Test accuracy, F1 score, and confusion matrices of all the classifiers (for a window size of 3 days) used for Experiment 2.

Classifiers	**Without 'day-cos' conditioning (7 days)**			**With 'day-cos' conditioning (7 days)**		
	Accuracy	**F-1 score**	**Confusion matrix**	**Accuracy**	**F-1 score**	**Confusion matrix**
Nearest-means (baseline)	0.833	0.792	[[31 6] [4 19]]	0.85	0.809	[[32 5] [4 19]]
Trivial System	0.5833	0.573	[[18 19] [6 17]]	0.5833	0.573	[[18 19] [6 17]]
Perceptron	0.533	0.622	[[9 28] [0 23]]	0.7	0.719	[[19 18] [0 23]]
KNN	0.866	0.833	[[32 5] [3 20]]	0.8	0.769	[[29 9] [3 20]]
Naive Bayes	0.833	0.782	[[32 5] [5 18]]	0.816	0.755	[[32 5] [6 17]]
Decision Tree	0.75	0.727	[[25 12] [3 20]]	0.75	0.727	[[25 12] [3 20]]
Random Forest	0.766	0.766	[[23 14] [0 23]]	0.7833	0.7796	[[24 13] [0 23]]
SVM	**0.85**	**0.83**	[[**29 8**] [**1 22**]]	**0.85**	**0.83**	[[**29 8**] [**1 22**]]

Table 9: Test accuracy, F1 score, and confusion matrices of all the classifiers (for a window size of 7 days) used for Experiment 2.

Tables 7, 8, and 9 report the test accuracy, F1 score, and confusion matrices on all the classifiers used to run Experiment 2. We experiment with varying the window sizes by 2, 3, and 7 days to calculate statistical features for feature engineering, and to understand the effect of how much of past data helps in accurate binary classification, alongside conditioning the features with the 'day-cos' feature.

We observe that, in Table 7, the best performing model is the SVM with a classification accuracy of 0.9 and F1 score of 0.888. Except for the Random Forest, all the classifiers achieved an accuracy more than the baseline. In short, the best model i.e., SVM achieved an improvement of 3.9% over the baseline, and about 59.01% over the trivial. On conditioning the features with the 'day-cos' feature, we don't see any drastic improvements in results. In Table 8, however, we see improvements in the results. The best performing model is the SVM with a classification accuracy of 0.9166 and F1 score of 0.893. It achieved improvements of 5.84% over the baseline, and 61.94% over the trivial. The other classifiers also achieved slightly/no better performances over the results in Table 7. We, however, notice a drop in results in Table 8. This was expected since the window size of 7 is too large. And since, we also drop out data of the last 7 days while doing the train-test split in the feature engineering step, the number of training data points (170) go down from the initial 184. There's a substantial loss of data. In all the three sub-experiments, the Random Forest classifiers achieved a very low performance metric with the SVM topping the charts. An optimal window size of 3 days is, hence, ideal. A very small window doesn't gather enough past data while calculating the statistics. Our hypothesis of conditioning the input features with "encoded" day features did give slightly/no better results for most of the systems. For a couple of systems, the performance however dropped, for reasons unknown and yet to be explored.

And hence, the best performing system and results were by SVM (window-size = 3 days, no 'day-cos' conditioning) that achieved a classification accuracy of 0.9166, an F1-score of 0.893, and the confusion matrix is $\begin{bmatrix} 34 & 3 \\ 2 & 21 \end{bmatrix}$**. The best model hyperparameters was a 'linear' kernel with the regularisation constant C = 5. It achieved an improvement of 5.84% over the baseline, and 61.94% over the trivial in terms of classification accuracy. For the F1 score, we observed an improvement of 7.2% over the baseline and about 61.94% over the trivial. As stated earlier, we make use of min-max normalisation, with the best features being 'DMC', 'DC', 'ISI', 'FFMC', 'Temperature-max', 'Rain-max'.**

6 Libraries used and what you coded yourself

The following is a summary of the Python libraries along with the modules that were coded by the team.

Libraries used	**Modules Coded**
os	Feature engineering (from scratch)
math	Pre-processing (from scratch)
numpy	Cross-Validation setup (from scratch)
pandas	KNN (from scratch)
sklearn	Nearest-means (from scratch)
matplotlib.pyplot	Trivial System (from scratch)
argparse	PCA (from scratch)
shutil	Data Analysis Plots
random	
seaborn	

7 Contributions of each team member

Work	**Contributors**
Literature Survey	Sarthak and Shoumik
Pre-processing and Feature Engineering	Sarthak
5-fold CV setup	Sarthak
KNN, Naive Bayes, Performance Metrics	Shoumik
Nearest Means, Trivial System, Perceptron	Sarthak
Decision Tree, Random Forest, Anova	Shoumik
PCC, PCA, Covariance, SFS, RFE, Mutual Information	Sarthak
Data Analysis	Sarthak and Shoumik
Report Writing	Sarthak and Shoumik
Other work	Sarthak and Shoumik

8 Summary and conclusions

In this work, we study the effects of conditioning trigonometric transformations of dates with the meteorological data to help in accurate predictions of fire/no fire in Algeria. We also compute statistical parameters (min/max/mean/median) of these features with a sliding window approach. For model selection, we heuristically tune hyperparameters of all the ML classifiers used until we achieved a satisfactory performance on the validation set. Results reveal that SVM trained on the best features (window size = 3 days) with a linear kernel and regularisation parameter of 5, gives the best results. However, conditioning of such ”encoded” date information does help in better predictions for most of the classifiers, of varying sizes. For an optimal solution, window size of 3 days gave the best results. For reasons unknown, we’re not sure as to why such conditioning doesn’t translate to the best model performances. In the future, we plan on exploring the extraction of better time-related features that would generalise well to unseen future data for the prediction of fire/no fire.

References

[1] Paulo Cortex, et al., ”A data mining approach to predict forest fires using meteorological data.”, (2007).

[2] Abid, Faroudja, and Nouma Izeboudjen. ”Predicting forest fire in algeria using data mining techniques: Case study of the decision tree algorithm.” in *International Conference on Advanced Intelligent Systems for Sustainable Development*. Springer, Cham, 2019. Cortez, Paulo, and Aníbal de Jesus Raimundo Morais. ”A data mining approach to predict forest fires using meteorological data.” (2007).

[3] https://fire.synopticlabs.org/tools/fwi/

[4] Bengio, Y. and Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5(Sep):1089–1105.

[5] C. Hsu, C. Chang, and C. Lin. A Practical Guide to Support Vector Classification. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/ guide.pdf, July, Dep. of Comp. Science and Information Eng., National Taiwan University, 2003.

[6] http://blog.davidkaleko.com/feature-engineering-cyclical-features.html

[7] https://en.wikipedia.org/wiki/Curse_of_dimensionality

[8] https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/pearsons-correlation-coefficient/

[9] https://machinelearningmastery.com/information-gain-and-mutual-information/

[10] https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

[11] https://machinelearningmastery.com/rfe-feature-selection-in-python/

[12] http://rasbt.github.io/mlxtend/user$_guide/feature_selection/SequentialFeatureSelector/$

[13] https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

[14] https://scikit-learn.org/stable/modules/generated/sklearn.feature$_selection.f_classif.html$

[15] https://statisticsbyjim.com/anova/f-tests-anova/

[16] https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476

[17] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htmlsklearn.tree.DecisionTree

[18] https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[19] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[20] https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f

[21] https://www.sciencedirect.com/topics/computer-science/majority-voting

[22] https://scikit-learn.org/stable/modules/naive$_{b}ayes.html$

[23] https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

[24] https://sdsclub.com/the-complete-guide-to-perceptron-algorithm-in-python/