EE 559 Final Project

Data Set: Algerian forest fires

Nisha Elizabeth Jacob, nejacob@usc.edu
Muskaan Parmar, muskaanp@usc.edu

May 4, 2022

1 Abstract

A forest fire is an unplanned and uncontrollable event and can be caused due to natural events such as ignition by the sun's heat or a lightning strike or due to human activities like unattended campfires and discarded cigarettes. It causes threat to human life and large amounts of economic and ecological damage. Climate change has increased the number of forest fires in the last decade, making them more common and severe. In this paper, we have implemented seven different models namely Trivial, Nearest Means, Support Vector Machine, Naïve Bayes Classifier, Logistic Regression, Decision Tree and Artificial Neural Networksto be able to find the best model that is correctly able to predict based on inputs about whether there would be a fire or not. We make use of features like the date, Temperature, Rolling Temperature, Fine Fuel Moisture Code, Duff Moisture Code, Drought Code, Initial Spread Index, and Build Up Index. The best performing model is the Decision Tree model which has an accuracy of 93% and an F1 score of 0.9.

2 Introduction

2.1 Problem Assessment and Goals

In this project, we present five systems for Algerian forest fire prediction. We use the dataset provided by UCI (Faroudja & Izeboudjen, 2020). The dataset provides a compilation of forest fire observations and data from two regions in Algeria – Bejaia and Sidi Bel-Abbes. It spans the months from June to September 2012. This forecast uses meteorological data for essential weather elements that influence the occurrence of forest fires, such as temperature, Build Up Index and Initial Spread Index. It contains 12 features with no missing data. The dataset is highly imbalanced with 37% points belonging to the no fire class and 63% belonging to the fire class. We aim to predict whether certain weather characteristics can help predict future forest fires in these regions using different classification techniques, and evaluate the accuracies of each model.

2.2 Literature Review

The author of [1] describes the implementation of Decision Trees for prediction of forest fires by integrating it into FPGA based smart sensor node. Binary Decision Tree algorithm is used for classification and an accuracy of 82.92% was achieved. From data analysis, it was found that Rain had no impact on the prediction of the classes whereas features like Temperature, RH and Wind Speed were having a high impact on the performance of the decision tree model. The paper also includes a comparison between different machine learning models like simple decision tree, random forest, boosting and bagging and states that the performance of the decision tree model is close to the performance of the boosting model. In [2], the authors make use of ANN to build a regression model that would predict the burned area of the forest. It describes a simple ANN model consisting of one input layer followed by one hidden layer and finally an output layer. It explains the importance

of other features like FFMC, DMC, DC and ISI for the prediction. The number of neurons for the input layer is decided based on the number of features present in the dataset. It makes use of backpropagation and trains the data on a large number of epochs to get a good accuracy. For [3], the author describes the usage of Bayes statistics for Naive Bayes classifier for forest fire detection and obtains a precision of 94%. This work was extended to detect fires in real time by making use of sensing nodes. In [4], the authors found that the best accuracy was obtained from the SVM model with the usage of temperature, relative humidity, rain and wind to predict the burned area of forest in Portugal.

3 Approach and Implementation

3.1 Dataset Usage

We have used a dataset that combines data from two regions in Algeria - Sidi Bel-Abbes located in the northwest of Algeria and Bejaia located in the northeast. The dataset contains 224 datapoints (122 points each from the two regions) for the days from June to September 2012. Of this, we use 184 points (June to August 2012) for training and validation. The remaining 30 days are used to test the accuracy of our model in predicting fires in the future. The dataset provided to us is highly imbalanced with 115 points from Class Fire and 69 points from Class No Fire. The distribution of the data points can be seen in the Figure 1.

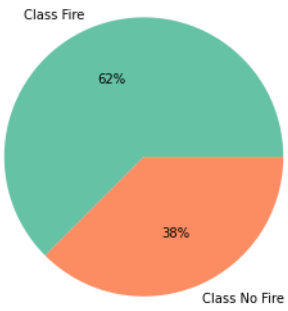

Figure 1: Pie chart showing the distribution of classes

As a part of feature engineering, we compute the rolling temperature average across 5 days as a new feature to help predict the accurate class label. In order to avoid past data having an undue influence in predicting future data, we remove the last 8 rows (4 days) from the training data since they would contain indirect information for the test set. Hence, our final training data contains 176 data points.

To reduce the number of features as compared to the number of datapoints, we only consider the 8 top features with highest correlation (obtained by computing Pearson's correlation coefficient for each feature). The 8 best features for our dataset were: Temperature, Rolling Temperature, Fine Fuel Moisture Code, Duff Moisture Code, Drought Code, Initial Spread Index, Build Up Index and the month. The remaining attributes do not really have an impact on the models.

We utilized cross-validation to ensure maximum usage of the available data points. We did not use any available libraries for this purpose and coded the functionality by ourselves. Cross-validation was performed for each model separately and is done after the stages of pre-processing and feature engineering are complete. We used 6 partitioning combinations, which means, we would have a total of 6 validation sets and 6 training sets. The principle of indirect information removal is used here as well. If the validation data contains moving average information for the training data, we

remove the last 8 rows from the validation set. The same is applied on the training data as well. For each division of the training and validation batches, we fit the model on our training data and evaluate the model on the validation set, for different hyper parameters. The F1 score is stored and is averaged across T, where T is the number of times we perform a new partitioning. The hyper parameter that gives the maximum average F1 score is selected for the final training. The model is finally trained on the entire training dataset with the optimal parameters and this model is used on the test set. Thus, the test set is used only once in each of our models and hence, we can be sure that the model is performing on data never seen before.

3.2 Preprocessing

Data preprocessing is an important step to clean and prepare the dataset for building our machine learning models. The dataset was standardized to make sure that feature values lie in a similar range. Standardization ensures that the features of the dataset have the same importance if they are measured on different scales. The use of StandardScaler from the sklearn library ensures that the mean and variance for each feature of the dataset is set to 0 and 1 respectively. After standardizing the train data, we apply the same scaling to the test data rather than standardizing both the datasets separately. We also perform one hot encoding on the newly formed days column. This generates 7 additional columns, one for each weekday. From the distribution of the features in the dataset in

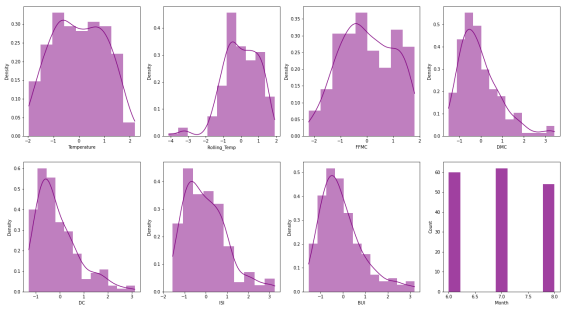

Figure 2: Distribution of features

Figure 1, we can see that the features have a distribution which is approximately Gaussian.

3.3 Feature engineering and dimensionality adjustment

A feature statistic that we used as an additional feature is the rolling temperature mean across 5 days (past 4 days and the current day). This was implemented using the .rolling() feature from the Pandas library. Since the dataset contains 2 data points for each day, the moving average remains the same for both the points because we consider the same days for each. In addition to this, we also extracted the day, month and year from the Pandas 'Date' column in the dataset. The year is discarded since it remains the same for the entire dataset. The month is coded using ordinal encoding, as it has a natural ordering. For the features related to the days, we use one-hot encoding for the 7 days of the week from Monday to Sunday. We cannot use ordinal encoding here as the data has no natural ordering in this case.

Feature dimensionality reduction was performed to obtain the important features from the entire dataset as the performance of the model can be poor when we have too many features. Dimensionality reduction is performed prior to modeling. We have performed dimensionality reduction on the test set as well.

First, we used Pearson correlation coefficient test to understand the relationship between the features. The value of Pearson correlation coefficient lies between -1 to 1. The heatmap in Figure 3 shows the correlation between all the features of our dataset.

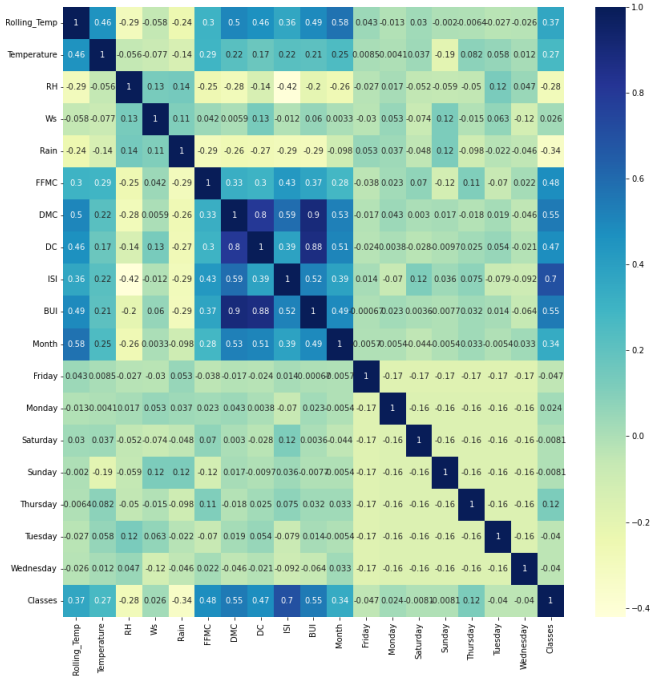

Figure 3: Heatmap showing the Pearson Correlation Coefficients

The coefficients can be interpreted as follows:

- A positive correlation value between two features indicates a positive correlation between the two features i.e. feature A increases as feature B increases
- A negative correlation value between two features indicates negative correlation between the two features i.e. feature A decreases as feature B decreases
- A correlation value of zero between two features indicates that there is no correlation between them.

From the heatmap, we can see that features like Rolling_Temp, Temperature, FFMC, DMC, DC, ISI, BUI and Month show a strong positive correlation which is indicated by the color blue. The other original features like RH, WS and Rain show a negative correlation which is indicated by the color yellow/green.

The second method used for dimensionality reduction was Principal Component Analysis (PCA). PCA is a technique used when the dataset has a lot of features. PCA makes use of a new coordinate system and transforms the data in such a way that the largest variance with some scalar projection of the data lies on the first coordinate axis and it follows the same method for the second largest variance and so on. Variance is a measure of the degree to which every data point differs from the mean. A high variance indicates more information.

To determine the total number of features we require for training our model, we have generated a plot shown in Figure 4. From the plot we can see that we only need 8 of the 19 principal components (features) to explain approximately 99% of the data. It means that when we reduce the number of features from 19 to 8 we are still able to retain 99% of information from the original data.

Thus, we discard features with negative correlation coefficient and keep the following features: Rolling_Temp, Temperature, FFMC, DMC, DC, ISI, BUI and Month which is in accordance with

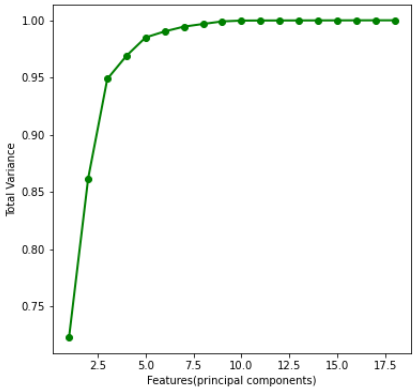

Figure 4: Plot of PCA

our observation from the plot of PCA.

3.4 Training, Classification or Regression, and Model Selection

3.4.1 Trivial Model

This model outputs class labels 0 and 1 at random with probability N_1/N and N_2/N respectively, where N_i is the population of datapoints with class label S_i and N is the total population of data points. This is all based on the training set. This model gives us an accuracy of 53% on the test dataset and an F1 score of 0.416. The confusion matrix for test set can be seen in Figure 5. Here,

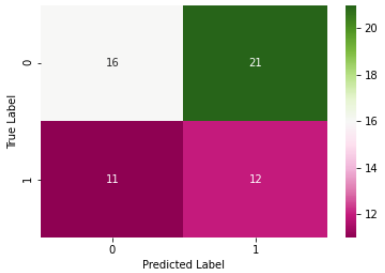

Figure 5: Confusion Matrix for Test Set on Trivial Model

the train dataset is divided into a train/validation dataset with an 80/20 split, for additional testing purposes. Since the outputs are generated at random, we see that this model provides very low accuracy.

3.4.2 Baseline Model

We use the nearest means classifier as the baseline model. It assigns datapoints the class label whose mean (derived from the training samples) is closest to the point. This is the simplest classification algorithm in Machine Learning. Its algorithm involves three simple steps:

- The centroid for each target class is computed while training
- After training, given any point, say 'X'. The distances (Euclidean) between the point X and each class' centroid is calculated.

- Out of all the calculated distances, the minimum distance is picked. The centroid to which the given point's distance is minimum, it's class is assigned to the given point.

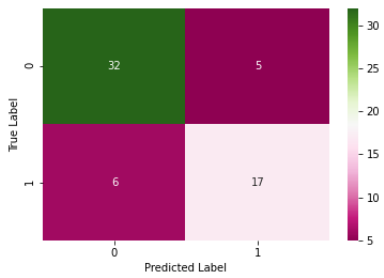

Figure 6: Confusion Matrix for Test Set on Baseline Model

This model gives us an accuracy of 81% on the test dataset and an F1 score of 0.75. The confusion matrix for test set can be seen in Figure 6.

Here, the train dataset is divided into a train/validation dataset with an 80/20 split, for additional testing purposes. The accuracy is surprisingly high eventhough not much learning is involved here. 6 days are detected as False Negatives.

3.4.3 Support Vector Machine Classifier

The SVM algorithm aims to find a hyperplane in an N-dimensional space (N–number of data points) that distinctly classifies the datapoints. This hyperplane should have the maximum margin (maximum distance between datapoints of classes). This is so that future data points can be classified with confidence. It can work on sparse data, including under-determined systems.

Support vectors are data points that lie closer to the hyperplane and have an influence on the hyperplane's position and orientation. The classifier margin is maximized using these support vectors. Lagrange optimization is used to minimize the loss function represented by the below equation

$$f(w) = \frac{1}{2}||w||^2$$

Support vector machines can also be used to classify data that are not linearly separable using the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to an expanded feature space. This makes the data easily differentiable. SVM supports four major kernels : Linear, Polynomial and RBF kernel.

1. The linear kernel is defined as: $K(x_i, x_j) = x_i * x_j$. This is used when the data is linearly separable in the original feature space and no extra transformation is required.

2. The polynomial kernel is defined as: $K(x_i, x_j) = (x_i * x_j + c)d$. This kernel contains two parameters: a constant c and a degree of freedom d. A d value of 1 is the same as the linear kernel. A larger value of d will make the decision boundary more complex and may result in overfitting.

3. The RBF kernel is defined as $K(x_i, x_j) = e^{-\gamma * ||x_i x_j||^2}$. The RBF (Radial Basis Function) kernel is also called the Gaussian kernel. It will result in a more complex decision boundary. The RBF kernel contains parameter γ. A small value of makes the model similar to a linear SVM. A large value of γ will make the model heavily impacted by the support vectors.

4. The sigmoid kernel is defined as a similarity measure, like the others. The hyperbolic tangent value is directly proportional to the value of the inner product of x_i and x_j.

For our dataset, since there is a large imbalance between the data of the majority and minority classes, we added the 'balanced' class weight mode which automatically adjusts the weights inversely proportional to the class frequencies in the input data. The model parameters used for hyper-parameter tuning and model selection are:

- C, the regularization parameter
- Kernel type, the algorithm
- γ, the kernel coefficient for RBF, polynomial and sigmoid kernels

The range of C values used were: 0.01, 0.5, 0.1, 1, 2, 5, 10, 50, 100, 500 and 1000. Small values of C will result in a wider margin at the cost of some misclassifications while large values of C behaves like a Hard Margin classifier and tolerates zero constraint violation. The types of kernel functions used include linear, polynomial, RBF and sigmoid kernels. The range of γ values employed were: 0.001, 0.1, 0.2, 2 and 10.

The optimal values of the parameters were chosen through the process of cross-validation. We used 6 equally sized subsets. Each of these subsets was used as a validation set for 6 different iterations and the remaining subsets comprised the training data for that iteration. Prior to fitting the model on the training data, we remove the last 8 days from the train/validation subset if they influence any data in the other subsets. After fitting, the model is evaluated on the validation subset and the F1 score is stored. The parameter combination that gives the highest F1 score across all iterations are chosen to be the optimal values of C, kernel and γ. In our case, the optimal value of C obtained through model selection was 10, the optimal kernel was the linear kernel and the optimal value of γ was 0.001.

We have selected 8 features for 176 datapoints. The degrees of freedom to number of constraints ratio lies well within the general rule of thumb ($N_c = (3-10) * d.o.f$).

The accuracy obtained on the test set with this model is 91.67% and the F1 score is 0.89. The confusion matrix for train and test data can be seen in Figure 7 and Figure 8 respectively. We can

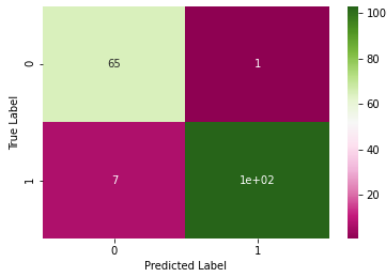

Figure 7: Confusion Matrix for Train Data on SVM Model

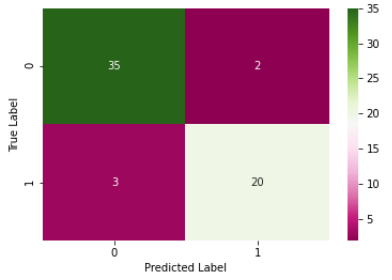

Figure 8: Confusion Matrix for Test Data on SVM Model

see that 3 cases are False negatives.

3.4.4 Naïve Bayes' Classifier

Naïve Bayes' classifier is based on applying Bayes' theorem with strong (naive) independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve high accuracy levels. Naive Bayes classifiers are highly scalable, requiring a number of parameters, linear in the number of features in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers. Bayes' Theorem can be written as :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B}$$

This can be rewritten as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

By substituting X as a combination of its individual features and applying chain rule, we see that the denominator term remain static. Hence, a proportionality can be introduced and we obtain the classification criterion as:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

There are three main types of Naïve Bayes' classifiers: Multinomial Naïve Bayes (used when predictors are categorical variables), Bernoulli Naïve Bayes (used when the predictors are boolean variables) and Gaussian Naïve Bayes (used when the predictors are continuous valued and not discrete).

For our dataset, since there is a large imbalance between the data of the majority and minority classes, we use *ComplementNB* provided by sklearn, which is suited for imbalanced datasets. The weights are provided per sample and is inversely related to the class size. The model parameter used for hyper-parameter tuning and model selection is: α, the additive smoothing parameter. The range of α values used were: 0.005, 0.01, 0.1, 0.5, 1 and 10.

The optimal values of the parameters were chosen through the process of cross-validation. We used 6 equally sized subsets. Each of these subsets was used as a validation set for 6 different iterations and the remaining subsets comprised the training data for that iteration. Prior to fitting the model on the training data, we remove the last 8 days from the train/validation subset if they influence any data in the other subsets. After fitting, the model is evaluated on the validation subset and the F1 score is stored. The parameter that gives the highest F1 score across all iterations is chosen to be the optimal values of alpha. In our case, the optimal value of α chosen was 0.005.

We have selected 8 features for 176 data points. The degrees of freedom to number of constraints ratio lies well within the general rule of thumb ($N_c = (3 - 10) * d.o.f$).

The accuracy obtained on the test set with this model is 81.67% and the F1 score is 0.717. The confusion matrix for the training and test set data can be seen in Figure 9 and 10 respectively. We

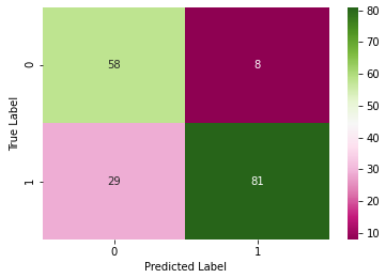

Figure 9: Confusion Matrix for Train Data on Naïve Bayes' Classifier

can see that 9 cases are False negatives. This is a very high number, indicating that many days are classified as no fire when there indeed was a fire, which is not a good sign.

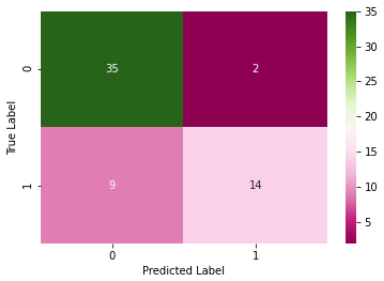

Figure 10: Confusion Matrix for Test Data on Naïve Bayes' Classifier

3.4.5 Logistic Regression

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. It is a simple and efficient method for binary and linear classification problems and achieves good performance with linearly separable classes.

Logistic model (or logit model) is a statistical model that models the probability of one event (out of two alternatives) occurring by having the log-odds (the logarithm of the odds) for the occasion be a linear combination of one or more independent variables ("predictors"). Logistic regression estimates the parameters of a logistic model. The logistic function is also commonly used in very complex neural networks as the activation function of output layer.

The main distinction between linear and logistic regression is that the range of logistic regression is limited to 0 and 1. Furthermore, logistic regression does not require a linear relationship between input and output variables, unlike linear regression. This is due to applying a nonlinear log transformation to the odds ratio.

$$Logistic function = \frac{1}{1 + e^{-x}}$$

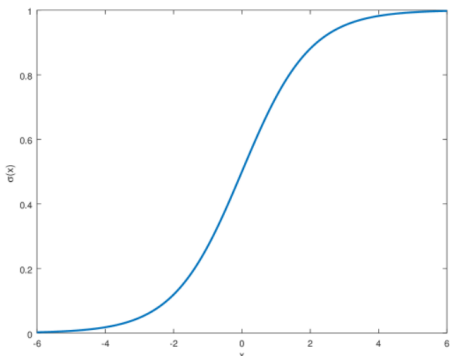

Figure 11: Sigmoid function

The loss function used here is related to "maximum likelihood estimation (MLE)" which is a conditional probability. We use log loss with a hard threshold for binary classification cases. Log Loss is the negative average of the log of corrected predicted probabilities for each instance. If the probability is greater than 0.5, the prediction will be classified as class 0. Else, class 1 will be assigned. Logit function is defined as the natural log of the odds. A probability of 0.5 corresponds to a logit of 0, probabilities smaller than 0.5 correspond to negative logit values, and probabilities greater than 0.5 correspond to positive logit values. Logistic function ranges between 0 and 1 $P[0, 1]$ while logit function can be any real number from minus infinity to positive infinity $P[-\infty, \infty]$.

$$LogLoss = -\frac{1}{N}\Sigma_{i=1}^{N} log(P_i)$$

For our dataset, since there is a large imbalance between the data of the majority and minority classes, we use the 'balanced' class weight mode which automatically adjusts the weights inversely proportional to the class frequencies in the input data. The model parameters used for hyper-parameter tuning and model selection are:

- C, inverse of regularization parameter
- Solver, the algorithm used in the optimization problem.

The range of C values used were: 100, 10, 1.0, 0.1 and 0.01.The types of solver functions used include newton-cg, lbfgs and liblinear. The "lbfgs" is an optimization algorithm that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm 8, which belongs to quasi-Newton methods. The solver "liblinear" uses a coordinate descent (CD) algorithm. The optimization problem is decomposed in a "one-vs-rest" fashion so separate binary classifiers are trained for all classes. The "newton-cg" solver calculates Hessian explicitly and can be computationally expensive for high dimensional data. The optimal values of the parameters were chosen through the process of cross-validation. We used

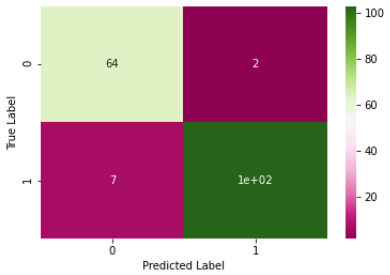

Figure 12: Confusion Matrix for Train Data on Logistic Regression Model

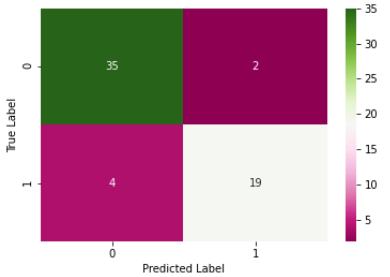

Figure 13: Confusion Matrix for Test Data on Logistic Regression Model

7 equally sized subsets. Each of these subsets was used as a validation set for 7 different iterations and the remaining subsets comprised the training data for that iteration. Prior to fitting the model on the training data, we remove the last 8 days from the train/validation subset if they influence any data in the other subsets. After fitting, the model is evaluated on the validation subset and the F1 score is stored. The parameter combination that gives the highest F1 score across all iterations are chosen to be the optimal values of c and solver algorithm. In our case, the optimal value of C chosen was 100 and the optimal solver was liblinear. We have selected 8 features for 176 datapoints. The degrees of freedom to number of constraints ratio lies well within the general rule of thumb $N_c = (3 - 10) * d.o.f)$. The accuracy obtained on the test set with this model is 90% and the F1 score is 0.86. The confusion matrix for the training and test set data can be seen in Figure 12 and 13 respectively. We can see that 4 cases are False negatives.

3.4.6 Decision Tree Classifier

Decision Tree Classifier is a method of Supervised learning by splitting the data into partitions iteratively and generating a tree-like structure for making decisions by using a set of rules. They are useful for solving cases where we have decision related problems.

A decision tree consists of nodes, edges and leaf nodes. The nodes represent the test feature. The edges are used to connect the node to another node/leaf. The root node contains the whole population of the dataset. The leaf nodes are the nodes that do not split further and are used to show the predicted outcome. As the forest fires dataset has only two outcomes: fire and no fire,

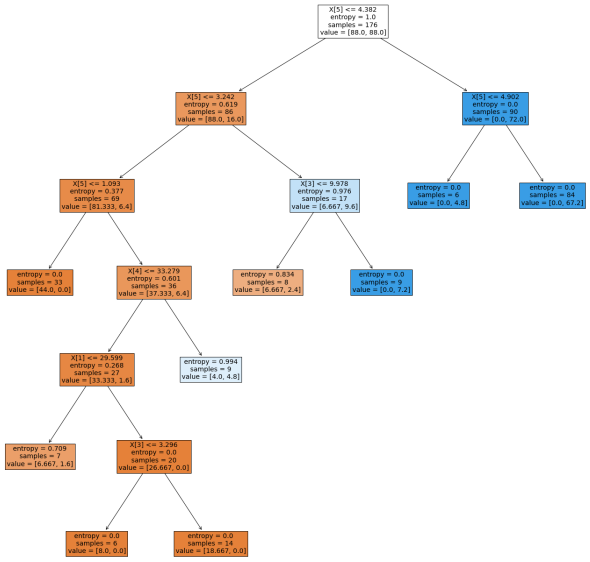

Figure 14: Structure of Decision Tree

there would only be two leaf nodes in the decision tree. This is called a binary decision tree where there are only two possible outcomes. From Figure 14, we can see that the decision tree model for our case also has only two leaf nodes.

A measure called Gini impurity is used to decide the optimal split from the root node. It can perform binary splits and would help us in generating a decision tree.

$$Gini = 1 - \Sigma_{i=1}^{C}(p_i)^2$$

where p_i stands for the probability

The gini impurity for a node lies in the range of 0 to 0.5. A gini impurity value of zero means that the node is a pure node i.e. a leaf node. Usually a node's gini impurity is less than its parent node's gini impurity. A low gini impurity indicates a high homogeneity of the node. So, gini impurities help us in finding out the root node, intermediate nodes, leaf nodes and the structure of the decision tree. Based on the gini impurities a decision tree was created to predict the forest fires.

For building our decision tree model,we make use of *DecisionTreeClassifier* from the sklearn library. We performed hyper parameter tuning for the following two parameters:

- min_samples_leaf, the minimum number of samples required to be at a leaf node

- min_samples_split, the minimum number of samples required to split an internal node

As our dataset is highly imbalanced, we have set the parameter class_weight as 'balanced'.This automatically adjusts the weights inversely proportional to the class frequencies in the input data.

The range of values for min_samples_leaf is 2, 3, 4, 5, 6 and 7 and the range of min_samples_split is 2, 3, 4, 5, 6, 7, 8 and 9. The optimal values of the parameters were chosen through the process of cross-validation. We used 6 equally sized subsets. Each of these subsets was used as a validation set for 6 different iterations and the remaining subsets comprised the training data for that iteration. Prior to fitting the model on the training data, we remove the last 8 days from the train/validation subset if they influence any data in the other subsets. After fitting, the model is evaluated on the validation subset and the F1 score is stored. The parameter combination that gives the highest F1 score across all iterations is chosen to be the optimal values of min_samples_leaf and min_samples_split.

After obtaining the optimal values for hyper parameters, we train the model on data comprising both training and validation using the optimal values of the hyper parameters. The accuracy obtained on the test set with this model is 93.3% and the F1 score is 0.9. The confusion matrix for the training and test set data can be seen in Figure 15 and 16 respectively. We can see that 3 cases

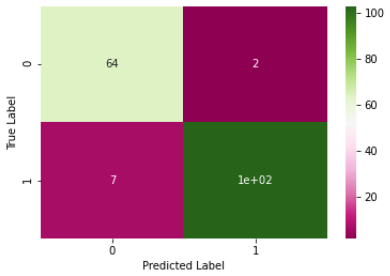

Figure 15: Confusion Matrix for Train Data on Decision Tree Model

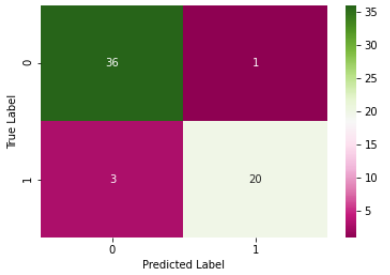

Figure 16: Confusion Matrix for Test Data on Decision Tree Model

are False negatives for the test set. This shows that our model is able to successfully predict most of the cases correctly.

3.4.7 Artificial Neural Networks

Artificial Neural Network (ANN) also called Neural Networks mimic the behavior of biological neurons present in our brain which help in decision making. The neural network created by us has three basic layers:

1. Input layer: The number of neurons in this layer is equal to the number of features present in the dataset i.e. 8 neurons are used in this layer since we have 8 features in our dataset.

2. Hidden layer: The first hidden layer has 6 inputs and ReLU as its activation function. This is followed by a second hidden layer which has 3 neurons and ReLU for adding non linearity.

3. Output layer: The last layer makes use of sigmoid as its activation function to generate the output of binary type i.e. 0 or 1 where 0 indicates no fire and 1 indicates fire.

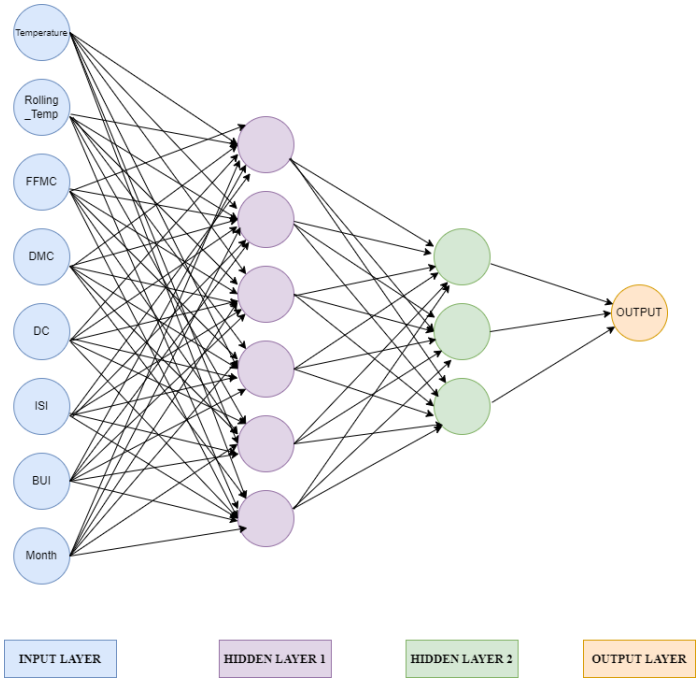

Figure 17: Structure of ANN model

Non linear activation functions:

- ReLU

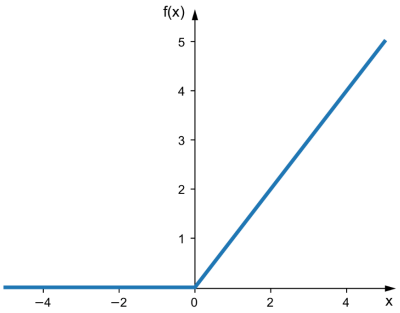

Figure 18: ReLU function

$$f(x) = max(0, x)$$

ReLU is one of the most used activation functions to add non linearity in the network structure since it is faster to compute it as compared to other activation functions. The ReLU function as shown in Figure 18 takes a value of 0 for negative inputs and is linear for positive inputs. We used ReLU as an activation function for both the hidden layers.

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function generates output in the range of 0 to 1. This property is helpful as we can convert the output into a probability. We use the sigmoid function for the final layer of our network.

We make use of the PyTorch framework which is based on the Torch library. We define the model under the class ANN. Here, we declare the structure of our network under the *init* function. A sequential container is used to prepare the network. We make use of a custom loss function so as to take the imbalance of the dataset into consideration. The loss function is similar to the binary cross entropy loss except this custom function would take class weights into consideration. The equation is given by:

$$Loss = -\frac{1}{M}\Sigma_{i=1}^{M}(w1 * y * log(a) + w0 * (1-y)log(1-a))$$

where $w0$ and $w1$ are the weights for classes 0 and 1 respectively and a is the output from the last layer of the ANN.

An optimizer is used to help find the global minima of the loss function. We make use of the Adam optimizer due to its faster computation property.We performed hyper parameter tuning for the following two parameters:

- lr, the learning rate, the step size taken at each iteration to move to the minimum of the loss function
- weight_decay, the L2 penalty

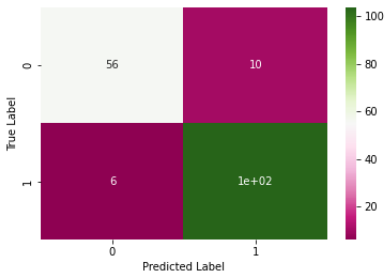

Figure 19: Confusion Matrix for Train Data on ANN Model

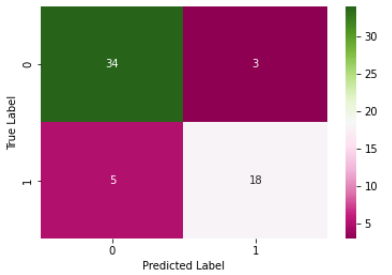

Figure 20: Confusion Matrix for Test Data on ANN Model

The range of values for lr is 0.0001, 0.001, 0.01, 0.1 and 0.5 and the range of weight_decay is 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1 and 0.5. The optimal values of the parameters were chosen through the process of cross-validation. We trained our model on 5000 epochs as it is a sufficiently large number for our model to learn the features of the training model. After every 10 epochs, we have calculated the accuracy for both training and validation. We make use of backpropagation to ensure that the model is able to learn weights based on the previous epoch.

After completion of the cross validation process, we obtain the optimal learning rate as 0.0001 and the optimal weight decay as 0.0001. Now, we train our model on these optimal parameter values obtained from hyper tuning and then find the accuracy on the testing data. The accuracy on test data was 86.6% and the F1 score was 0.81. The confusion matrix for the training and test set can be seen in Figure 19 and 20 respectively. We can see that there are 5 data points under the False Negatives section of the confusion matrix. A readme.txt file has been provided along with this which explains in detail about the libraries used and the libraries required to run this code.

4 Analysis: Comparison of Results, Interpretation

4.0.1 Summary of Baseline and Trivial models

Below are the final performance measures for the train, validation and test sets for the trivial and nearest means classifier models. We divide the training data into train and validation data with an 80/20 split (along with removal of last 8 days to avoid predicting future information)

	Training Accuracy	**Training F1**	**Validation Accuracy**	**Validation F1**	**Test Accuracy**	**Test F1**
Trivial Model	42%	0.375	41%	0.58	53%	0.416
Baseline Model	78%	0.78	96%	0.98	81%	0.75

4.0.2 Summary of ML methods we designed

We have highlighted the training and validation performance measures seen during cross validation (on average) below. Using the optimal parameters obtained during model selection, we fit our model to the entire training data one last time and this is indicated as the 'Final Training' measure below. Finally, we also have listed the performance measures on the Test data.

	SVM	**Naive Bayes**	**Logistic Regression**	**Decision Tree**	**ANN**
Cross-val Training Accuracy	82%	77.3%	94.1%	96.3%	87%
Cross-val Training F1	0.76	0.794	0.951	0.969	0.82
Cross-val Validation Accuracy	70%	76.3%	90.4%	96%	84%
Cross-val Validation F1	0.63	0.692	0.894	0.861	0.753
Final Training Accuracy	95%	95.4%	94.8%	95.4%	90.9%
Final Training F1	0.96	0.962	0.958	0.963	0.924
Test Accuracy	91.67%	81.67%	0.9%	**93.3%**	86.67%
Test F1	0.88	0.72	0.863	0.9	0.82

From the tables above, we can conclude that Decision Tree Classifier performs the best on the given dataset with a high accuracy of 93.3% and an F1 score of 0.9 on the test dataset. Only 3 days

are detected as false negatives. In order to achieve this, rolling mean of 5 days (past 4 and current) is included as an extra feature and any influence on future dates is removed while calculating performance on the train/test set.

The set of features included in the final dataset are: Rolling Temp, Temperature, FFMC, DMC, DC, ISI, BUI and Month.

The set of parameters to achieve this accuracy/F1 score include: min_samples_split - 6, min_samples_leaf - 6, criterion - gini, class_weight = 'balanced'

These results indicate that the data could very well be linearly separable with a couple of outliers for each class. This is because of the high accuracy obtained by the SVM linear kernel as compared to the RBF kernel which works very well at separating non-linearly separable data. The nearest means model is also able to give a good accuracy of around 81% on the test data which means that the data of both classes are clustered together and not far apart causing the centroids to be located at the centre. This leads to more points being correctly classified by this method. There are however, 6 days classified as no fire when a fire indeed happens, which is not a good sign. Our models aim to reduce the number of false negatives while keeping the accuracy high. We use F1 score as the determining factor in model selection. F1 score combines the precision and recall of a classifier. It is given by:

$$\frac{2.precision.recall}{precision + recall}$$

Decision trees are susceptible to any changes in the training data, but through model selection and cross-validation, it has been given exposure to different models through varying divisions of the train and val sets. This helps it achieve maximum performance. We are able to keep the number of false negatives to a minimum here, only 3 days are classified as no fire when a fire actually occurs.

Naive Bayes' classifier is only able to achieve an accuracy as much as our baseline system. This could be because of the underlying multinomial distribution with which the model is not able to accurately identify which class a certain point belongs to. *ComplementNB* from sklearn was used in order to incorporate the weights of the datapoints since the classes are highly imbalanced in the dataset. Logistic Regression performs fairly well with a regularization coefficient of 100. With hard thresholding performed on the sigmoid output, we achieve a high accuracy of 90%. 4 false negatives are predicted through this method.

We also see that for almost all models, our data is able to achieve high metrics while predicting the final training data with optimized parameters. This means that our model has been able to fit well to the network without much overfitting. Thus, we have maintained our constraint to degree of freedom ratio.

Since there are very few datapoints in the train and test set here, we need to be careful about the number of batches we divide the dataset into while performing cross validation. Too few batches and there won't be a lot of variety in exposure to the model at hand. Too many batches and there will be very little data in each batch for the model to learn from. Through trial and error, we found 6 batches to be the optimal number which contained sufficient data for the model to absorb.

5 Libraries used and what you coded yourself

The following libraries from Python were used for coding:

1. **Pandas**: We used the Pandas library for feature extraction from our data frame. We worked on extracting features like weekday and month from the 'Date' column present in the dataset.This was done by first converting the 'Date' column to Pandas datetime format and then making use of datetime library functions. Also, we used one hot encoding provided by Pandas to split the weekdays i.e. a separate column for each weekday. After the completion of Feature Engineering, we used the drop function of this library to discard the columns which we were no longer required after the analysis.To create a column with a moving average of temperature from the temperature column of the data frame, we use the function called .rolling() to find the rolling mean of temperature for the past 5 days. We use the .corr() function to find the

pairwise correlation of the features in the data frame. This was performed using the Pearson method.

2. **Datetime:** Datetime is a standard Python library for managing time related information in the dataset. It was used to extract the month and the weekday from the 'Date' column of the data frame. This helped us in creating new features.

3. **Numpy:** Numpy library was used in performing mathematical and matrix/vector computations. It makes the computations faster for the case of matrices and makes use of less memory. Some of the functions we make use of are .concatenate(), .delete(), .zeros(), etc.

4. **Scikit learn:** The scikit learn library provides built-in functions for creating machine learning models for prediction.We used it for standardization of our dataset under the preprocessing section. For PCA, we make use of the PCA function provided under sklearn.decomposition. We make use of the SVC algorithm provided by the sklearn.svm module to create our SVM model for classification. For our Naive Bayes Classifier, we make use of the ComplementNB algorithm under the sklearn.naive_bayes module. The sklearn.linear_model module provides a method called LogisticRegression which was used to build our Logistic Regression model for classification. For our Naive Bayes Classifier, we make use of the ComplementNB algorithm under the sklearn.naive_bayes module. To build our Binary Decision Tree classification model, we make use of the DecisionTreeClassifier algorithm provided by the sklearn.tree module.

5. **Matplotlib and Seaborn:** We used the Matplotlib and Seaborn library for generating data visualizations and graphics for easy understanding. We used it to show the heatmap of Pearson correlation coefficient, PCA plot, confusion matrices for each model, etc.

6. **PyTorch:** We used PyTorch for creating our ANN model. The functions provided by PyTorch make the work easy and efficient to create a neural network. We make use of the Sequential container to create the framework of our neural network. From the torch.optim package we make use of Adam algorithm. Please refer to the README.txt file for more information on installation of PyTorch and steps to run the code for the ANN model.

7. **Self-coded:** We implemented cross-validation ourselves without using any libraries available for this purpose. The dataset was divided into partitions of approximately the same size and using the concept of cross-validation, each partition was assigned as the validation set for different folds. Cross-validation was performed for each model separately after the stages of pre-processing and feature engineering are complete. We also executed the idea of eliminating the last x days while dividing the datasets so that no past information can influence future decisions.

6 Contributions of each team member

- Nisha Jacob - Responsible for feature engineering (computing Rolling Temperature), developing the trivial and baseline models and developing the SVM, Naive Bayes and Decision Tree models

- Muskaan Parmar - Responsible for data preprocessing (extracting date information and standardizing) and dimensionality reduction (extracting necessary features) and developing the ANN and Logistic Regression models.

Each section of the report was a combined effort with equal inputs from both sides.

7 Summary and conclusions

We have implemented seven different machine learning models for classification of forest fires namely Trivial, Baseline, Support Vector Machine, Naïve Bayes Classifier, Logistic Regression, Decision Tree

and Artificial Neural Networks. We found that the best performing model from all of them was the Binary Tree Classifier which had an accuracy of 93.3% and F1 score of 0.9 on the test set. Since our dataset is highly imbalanced, the f1 score plays an important role. The train data contains 62.5% of Class Fire label i.e. label 1 and 37.5% of Class No Fire label i.e. label 0. The F1 score is the harmonic mean of precision and recall.

The F1 score can range from 0 to 1. A score close to 0 indicates that the model is unable to classify points correctly and a score close to 1 indicates that the model is able to classify the points correctly. So, the F1 score of 0.9 shows that our Decision Tree model is able to classify Fire and No Fire cases with a low recall number. Only 3 days have been classified as False Negatives, as can be seen from the confusion matrix in Section 3.4.6. Further work can be done to bring this number down to zero, which is the ideal case.

We plan to work on extending our project by working on the problem of forest fires in California since each year California faces the problem of more than a dozen forest fires. We plan to collect data related to wind, region and climate from the past years which would be helpful in creating a model that can predict fires in this region. We can also extend the work to building an application that can provide real time prediction of forest fires.

Some of the key points we learned through this project are:

- Importance of pre-processing and feature engineering on datasets to be able to obtain features that have good correlation with the output.
- Performing dimensionality reduction using PCA/correlation to avoid underfitting in datasets that do not have a large number of datapoints.
- How to deal with datasets that have a high imbalance between the class labels and how different ML methods provide simple ways of learning from them.
- New models outside EE 559 such as Decision Tree and how they can be used as classifiers.

References

[1] Faroudja Abid; Nouma Izeboudjen, *Predicting Forest Fire in Algeria Using Data Mining Techniques: Case Study of the Decision Tree Algorithm*, in book: Advanced Intelligent Systems for Sustainable Development

[2] Aladdin Khaled Al-Zebda; Mutasim Mahmoud Al-Kahlout; Ahmed Mahmoud Abu Ghaly; Donia Zaher Mudawah *Predicting Forest Fires using Meteorological Data: an ANN Approach*

[3] Massinissa Saoudi; Ahcène Bounceur; Reinhardt Euler; Tahar Kechadi *Data Mining Techniques Applied to Wireless Sensor Networks for Early Forest Fire Detection*, in Proceedings of the International Conference on Internet of things and Cloud Computing, 2016

[4] Paulo Cortez; A. Morais*A Data Mining Approach to Predict Forest Fires using Meteorological Data*

[5] Thomas W. Edgar; David O. Manz*Exploratory study of Logistic Regression* in Research Methods for Cyber Security, 2017

[6] https://scikit-learn.org

[7] https://numpy.org

[8] https://pytorch.org/