Wildfire Damage Mitigation: Predictive Machine Learning Models for Wildfire Severity Classification

Jason Widjaja Djuandy, Nail Enikeev, Nassim Ali-Chaouche, Rohini Sidharth Kulkarni

Department of Applied Data Science, San Jose State University

DATA 270: Data Analytics Process

Dr. Ming-Hwa Wang

Abstract

This project presents a machine learning-based approach to predict the severity class of a wildfire using data such as weather and satellite data. The purpose of this project is to address the increasing occurrence of wildfires and their destructive impact on human lives and the environment. The project aims to develop a predictive model that can accurately forecast the severity extent of a wildfire, hopefully enabling better decision and resource management for first responders in the event of multiple wildfires therefore mitigating the damage done by wildfires, saving lives and the environment in the process. The project involves multiple tasks, spanning the collection and processing of various datasets such as historical wildfire incident data and data such as weather data, vegetation index, drought index, elevation and slope which may play a part in predicting the severity of a wildfire. Alongside these tasks, the project also involves designing and training machine learning models to identify trends and patterns and evaluating their performance on a test set using appropriate metrics, such as F1-score, Matthews Correlation Coefficient, and Macro AUC-ROC. Effective approaches such as hyperparameter tuning through stratified k-fold cross validation and grid search will be utilized to develop accurate and robust models for wildfire severity class prediction. The models being evaluated are k-NN, SVM, Random Forest, XGBoost, AdaBoost, and Artificial Neural Network (ANN). All of the models performed slightly better than a random classifier, with an Artificial Neural Network model having the best overall performance with a Matthews Correlation Coefficient of 0.08 and an AUC-ROC score of 0.57.

1. Introduction

1.1 Project Background and Executive Summary

OEHHA (California Office of Environmental Health Hazard Assessment) (2022) states that: “The area burned by wildfires and the number of large fires across the state have increased markedly in the last 20 years ... Wildfires in 2020 burned an unprecedented 4 million acres across California. In 2021, about 2.6 million acres burned, making it the second highest year, followed by 2018, with 1.5 million acres burned” (p. 1).

The damage caused by wildfires to the environment needs to be addressed immediately. Preventing wildfires may be the first logical thing to explore. Lightning is one of the natural causes of wildfires; however, not much can be done to avoid this genuine cause. According to the National Park Service (2022) article, “Nearly 85 percent of Wildland fires in the United States are caused by humans. Human-caused fires result from campfires left unattended, debris burning, equipment use and malfunctions, negligently discarded cigarettes, and intentional acts of arson.” Preventive actions can be taken to reduce the chances of wildfires happening, such as laws and fines. Even with laws, wildfires still occur, and immediate countermeasures must be taken.

In a wildfire, the top priority is to extinguish or stop it as soon as possible. The longer it takes to extinguish the fire, the more damage it will do to the environment and people around it. Coming up with a solution or tool that will help with damage mitigation caused by wildfires will, at the same time, save the environment and lives that could be affected by the fires. The team proposes a solution for the wildfire severity class prediction method, which will allow for better resource management for firefighters and first responders. In the event of multiple wildfires, the

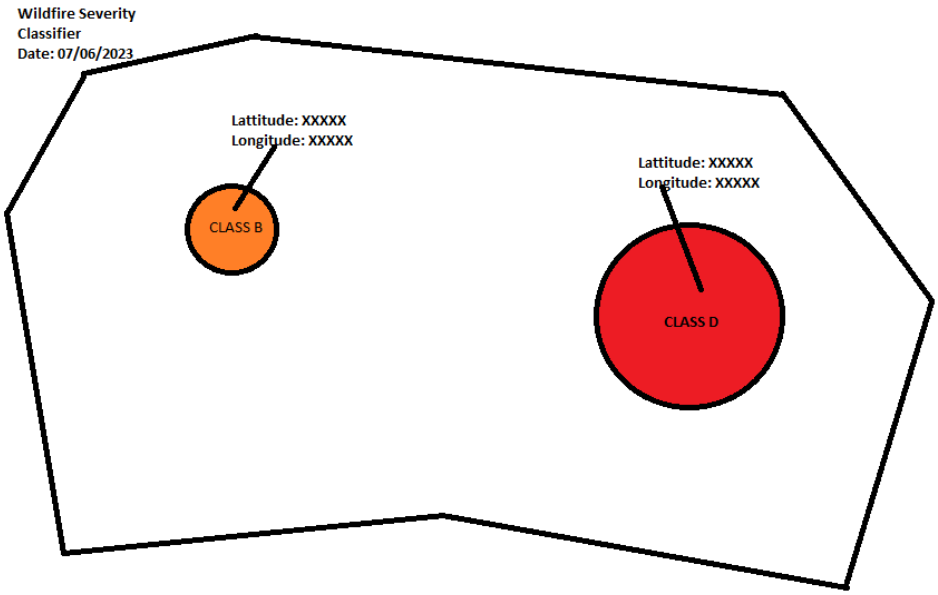
proposed solution will classify the severity class for each wildfire, first responders can then put in more resources into dealing with more severe wildfires compared to less severe ones.

Approach and method of this project would be to initially use available historical data on wildfires, then considering weather data, drought index, vegetation index, and elevation and slope during the wildfire event, key factors can be identified which will induce wildfire spread by comparing the different wildfire incident severity classifications. Different approaches can be achieved with different machine learning methods. Selecting the most accurate model will help create predictive wildfire severity scores to combat and mitigate wildfire damage.

The contributions and applications that this project will bear fruit will be useful for emergency responders such as firefighters and safety personnel which will help for better decision making and better handling of wildfires. The project hopes to be able to visualize the wildfire severity class, as illustrated in Figure 1 below.

Figure 1

Project Application Vision



1.2 Project Requirements

The functional requirements of the project include the ability to predict the wildfire severity class of a wildfire instance. Data will be collected from several sources, such as government websites, and APIs through sources such as Visual Crossing for weather data and Google Earth Engine for vegetation, elevation, and slope data. The data must cover several areas, including weather data such as wind speed, temperature, and humidity. Satellite and geographical data which includes the drought index, vegetation index, slope of the terrain, and elevation, will be included.

Data pre-processing will be required, such as handling missing values, feature normalization, and dimensionality reduction. Machine learning techniques, such as KNN, Random Forest and neural networks (such as ANN), will be trained and evaluated for the purposes of this project. Hyperparameter tuning and cross-validation methods will be used to generate a more accurate and robust model. The quality of the model's performance will be evaluated through appropriate metrics such as F1 score, Matthews Correlation Coefficient, and Macro AUC-ROC score. The predictions from the model should be easily interpretable to aid emergency responders in mitigating the effects of a wildfire.

1.3 Project Deliverables

The project result is the web-application "Wildfire Severity Score," which uses real-time data and the best performing machine learning model to predict the severity score of wildfires that are happening currently. The application is an interactive tool that helps firefighters and emergency responders mitigate wildfire damage. In the event of multiple wildfires happening concurrently, the tool will allow firefighters to gain some insights into which wildfires are likely to be more severe. This in turn will allow first responders to allocate resources more efficiently, especially

during the active wildfire season. A better allocation of resources into handling more severe wildfires will mitigate the damage done in the process as well.

Effort estimation is used to understand the amount of effort required to solve the research problem. It is done for various processes depending on team member availability and task complexity. The Gantt chart is used as a graphical representation of tasks, subtasks, dates, milestones, and resources allocated to each task/subtask in the form of bars. It provides a comprehensive view of all subtasks and how they are interrelated. The Work Breakdown Structure (WBS) is a hierarchical representation of project components, work packages, tasks, and deliverables that correspond to the phases of the CRISP methodology. The PERT chart organizes and schedules tasks in a project by breaking down individual tasks, mapping them, and visualizing the timeline. It helps estimate the minimum amount of time required to complete all tasks in a project.

The final report for the project describes how the team solved the research problem. The document explains the steps the team took to solve the problem, compares different machine learning models, and assesses the performance of each of them. The abstract will cover a summary of the research project as it will give a preview of the paper. The introduction will explain more into the problem and motivation as well as current solutions listed in the literature survey. Data management plan will go into the steps or approaches taken to collect the data required for the project. Project management plan explains the methodology used for the length of the project as well as a timeline for project deliverables. The Data Engineering chapter contains all the processes used to decide on data sources, steps to collect data, store, clean, and perform transformations if required, split available data for testing and training, and summarize the analysis results using visualization tools. Model development explains the model used,

justification for using those models, metrics used to evaluate the performance of the model as well as the results of each model.

1.4 Technology and Solution Survey

Wildfires have become a significant global issue in recent years, severely harming infrastructure, human life, and natural ecosystems. A promising method for reducing the damage caused by these catastrophes is using predictive machine learning models to map wildfires. This study will examine existing technologies and solutions that could satisfy the project's needs, including features and applications, techniques, algorithms, and models. According to statistics generated by Salas (2023), California is one of the states in the US that experiences wildfires the most frequently, with a fair number of them happening every year. In California, where wildfires pose a severe threat to natural and urban areas, applying predictive machine learning models to predict wildfires can be very helpful.

Geographical information system technology is frequently utilized in California to map and analyze wildfire data, and it has significantly aided firefighting organizations' response times. In addition, drones and satellites that use remote sensing are increasingly used to track the spread of fires and monitor wildfire activity. Real-time weather data reports are particularly beneficial for gathering images of wildfires, which may be used to inform firefighting operations.

Based on historical data and real-time inputs like weather, vegetation cover, and topography, machine learning algorithms have been employed in California to predict the likelihood and spread of wildfires. These algorithms have successfully sent out early caution warnings and directed allocating resources for battling fires.

In California, AI systems are also being investigated for their potential to analyze vast volumes of data from multiple sources, including social media and news reports, to detect the beginning of wildfires and provide firefighters and other first responders with real-time information. As an example, the WIFIRE Lab at the University of California, San Diego, uses supervised machine learning on satellite imagery as well as landcover maps which are then reclassified into higher resolution for better fire prediction (WIFIRE, n.d.). Several machine learning algorithms and models, including decision trees, random forests, neural networks, support vector machines (SVMs), and Bayesian networks, can be used to map and predict wildfires. While random forests are an ensemble learning technique that mixes many decision trees to improve prediction accuracy, decision trees are a straightforward but efficient algorithm used for classification and prediction. On the other hand, a subset of deep learning algorithms called neural networks can discover patterns and connections between the properties of the input and the results of wildfires. Bayesian networks are probabilistic graphical models for prediction and decision-making, while SVMs are for classification and regression analysis. Depending on the amount and complexity of the dataset, the available computing resources, and the desired level of forecast accuracy, the machine learning algorithm or model to be used for wildfire mapping and prediction will be chosen.

1.5 Literature Survey of Existing Research

Liang et al. (2019) developed a model to predict the scale of a wildfire using solely meteorological data. The wildfires were classified as Levels 1, 2, 3, 4, or 5, with the scale corresponding to the area burned and the fire's duration. The total number of samples was around 72,000, which all came from the region of Alberta, Canada. The data was extracted from the Canada National Fire Database. They removed variables with a variance high inflation factor

and used Min-Max scaling for feature normalization. They assessed the performance of three different neural network models: Backpropagation Neural Network (BPNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM). Of the three, the LSTM model has the highest predictive accuracy at around 90.9%, with an AUC of 0.942. The authors state that the study demonstrates that it is feasible to predict the scale of forest wildfires using solely meteorological data and acknowledges that the model has limitations since the data all came from a single region and thus have similar types of topography features.

Jiang et al. (2021) developed models to predict wildfire risk based on environmental factors. They divided the state of California into a few thousand grids, with each grid being 0.1 by 0.1 degrees in terms of latitude and longitude and predicted the wildfire risk for each grid. The data on the wildfire incidents was retrieved through the CA Fire database. They developed static prediction models, which consider historical data, and dynamic prediction models, which included a time element. They predicted the wildfire risk for a particular year from 2013 to 2016. There was a class imbalance in the authors' dataset, and a SMOTE oversampling technique was used. They assessed the performance of Multi-Layer Neural Networks (MLNN), Logistic Regression, Support Vector Machine, and Random Forest models to create the static prediction model. MLNN has the best predictive accuracy of around 64.5%. Since the MLNN model performed best for the static prediction model, MLNN was used to create the dynamic prediction model. The dynamic prediction model generated wildfire risk predictions for another state, Washington. The impact of every environmental factor (altitude and drought severity index) was also assessed.

For predicting wildfire spread using features like topography, weather, vegetation, and population density, Huot et al. (2022) article "Next Day Wildfire Spread Prediction using Deep

Learning on Remotely Sensed Data" compare the performance of neural network models with statistical analysis and non-deep learning(libraries like NumPy, SciPy, and matplotlib, and provides a wide range of algorithms for classification, regression, clustering, and dimensionality reduction, among others) models like logistic regression and random forest. Convolutional autoencoder is employed as the deep learning model to use the spatial information in the input data. The neural networking approach using logistic regression feature analysis provided accuracy, analysis of the effect of different input features, model performance, insights into which variables contributed the most significant predictive capabilities, and a confusion matrix. They also conduct a random forest study, which demonstrates the tremendous potential of deep learning for wildfire prediction. Finally, they assess the impact of various input characteristics and model performance. They provide insights into which factors contribute the most significant predictive capacities using neural networking (logistic regression feature analysis). Before adopting data-driven techniques into wildfire warning and prediction technology, the article's conclusion highlights the limitations in data resolution and labeling that must be addressed.

Recent studies have shown that deep learning techniques hold promise for developing early wildfire detection systems. Mahdi and Mahmood (2022) evaluate various deep-learning methods for their effectiveness in early wildfire detection. The paper analyzes each technique's advantages and disadvantages and provides recommendations for future development. The paper discusses the challenges of early wildfire detection and introduces the concept of using deep learning techniques for this purpose. The study examines several deep learning techniques, including Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Deep Belief Networks (DBN), and evaluates their accuracy, speed, and reliability. The report also highlights the difficulties encountered while developing early wildfire detection systems using

deep learning techniques and suggests areas for future research. The research emphasizes the potential of deep learning techniques in creating early wildfire detection systems. However, some issues still need to be addressed to improve the precision and reliability of these systems. The authors of this paper recommend that future research focus on developing more effective and efficient deep-learning systems for wildfire detection. They also stress the importance of gathering and annotating data to evaluate these systems and propose the creation of big datasets exclusively for wildfire detection to enhance their performance. Overall, this report provides valuable insights into the potential of deep learning techniques for early wildfire detection and identifies areas for further study.

Zope et al. (2020) used the Wildfire Prediction System (WiPreSy) by employing machine learning to predict the intensity of a forest fire hopes to prevent huge losses due to wildfires. Iot sensors and deep neural networks are utilized to predict the intensities of these fires. Sensors are used to record the temperature, humidity, soil moisture, pressure, and altitude. Sudden changes in the following features are excellent identifiers of a wildfire. The sensors will upload data collected real-time into the cloud and at the same time a model is trained with historical forest fire data. The estimated is calculated by a measure of confidence with a 91 percent in classification performance for their employed deep neural network with a binary cross-cross entropy loss of 1.12. Intensities were graded in low, medium, high and very high. A main drawback of the current solution is that the sensors employed throughout the forest still need a working internet connection to update the data real time into the cloud for further processing. This is an issue as internet coverage around forest areas is very scarce.

Girtsou et al. (2021) creates a next day wildfire prediction in Greece with the help of machine learning models. Data on vegetation index NDVI/EVI, temperature, wind components,

precipitation, land use/cover, Dem (Dem, aspect, slope, curvature) are collected from different sources such as FIREHUB, NOA, and NASA. The models used were shallow Neural Networks (NN architectures of maximum 3 hidden layers), Random Forest, XGBoost and Logiboost. 10-Fold Cross validation was done for hyperparameter tuning and best performing models were chosen based on F-score, ROC-AUC, as well as hybrid measures HMRa and HMRb are taken into account. 96% and 97% Recall for RF and NN respectively was identified and a risk map was then generated using the NN showing different shades corresponding to different probabilities in Greece where a wildfire will occur.

2. Data and Project Management

2.1 Data Management Plan

The project will focus on wildfire instances in California due to the frequent occurrence of wildfires in the area. Multiple datasets from different sources have been selected for this project as each of them are identified to be possible factors which affect the rate and area of the wildfire spread. The historical wildfire data used is the incident data taken from the State of California's Department of Forestry and Fire Protection (CAL FIRE). For weather data, the data is obtained from Visual Crossing as it provides daily weather data such as wind speed, wind gust, precipitation etc. Drought index data is obtained from the gridMET dataset. Vegetation indices are provided from NASA in the MOD13A1.061 dataset at a 500 m resolution updated every 16 days. Elevation and slope information is provided by the Shuttle Radar Topography Mission (SRTM) at a 30 m resolution from NASA.

Google Earth Engine (GEE) is used to collect the vegetation index and elevation/slope data from the corresponding NASA datasets. The GEE API can collect data by exact geographic locations and dates (which in the case of this project, the state of California). The gridMET data which contains the drought index data is hosted by Climatology Lab in the form of a file download. All the datasets excluding weather are taken from public government/institutional agencies such as NASA which promote the use of the datasets provided by them to be used by the public for research purposes. The weather data is collected from the Visual Crossing API and does not allow for the data to be publicly shared, therefore limitations for weather data sharing will be noted. The Visual Crossing API provides free access and paid access for data retrieval. For instance, in the free tier, 1000 records per day can be obtained, and with the "Professional" Tier, 10,000,000 records can be obtained per month with no daily limits. The Google Earth

Engine API is completely free, however there are limitations such as a single query being limited to 10 MB, and 6000 requests are allowed to be made per minute.

Weather, vegetation, drought, elevation, and slope data will be retrieved from multiple sources for each of the wildfire incidents and stored in CSV file format. The different features will be merged into one file and then prepared for data modeling. For folder structure, file name conventions and versioning, refer to the table below.

Table 1

File and folder structure

Purpose	Folder name	File name convention and versioning
Historical Wildfire Data	Datasets	<dataset_name>_<version_num>.csv
Weather Data	Datasets	weatherdata_<version_num>.csv
Vegetation Data	Datasets	vegetationdata_<version_num>.csv
Drought Data	Datasets	droughtdata_<version_num>.csv
Elevation and Slope Data	Datasets	elevation_slopedata_<version_num>.csv
Merged Data	Merged Data	merged_data_<version_num>.csv
Cleaned and Preprocessed Data	Transformed Data	transformed_data_<version_num>.csv

Data collected for the project must be protected and its storage environment must be secure from any form of tampering. A quick survey was conducted on the possible data sources for the purposes of this project, and it was identified that for the current solution, a cloud data repository such as Amazon S3 would not be required at this point in time. The data can be stored in a private Google Drive to be shared only to the project members. If in the future of this project, additional storage space and security is required, other solutions such as Amazon S3 and Amazon's Identity and Access Management (IAM) feature can be utilized to ensure that appropriate read and write permissions are provided only to the members working on the project. If any change in permission access arises in the future, then the IAM role can be easily updated to do so.

The processed data collected for the purposes of the project will be published in Kaggle if the data is legally allowed to be shared to the public for public use. This naturally will include the metadata for each of the sources given there is one. The code as well as the methods used to retrieve the data will be shared in the GitHub repository where future use and method replication can be done easily. This means the code will include documentation and steps on using the Visual Crossing API as well as the Google Earth Engine API along with the code for the models' training, testing etc. After the completion of the project, data which is legally allowed to be published will be available on GitHub.

Distribution of responsibilities and resource allocation for the different phases of the Data Management Plan (DMP) can be referred to in the table below.

Table 2

Responsibilities for each phase of the DMP

DMP Phase	Resource	Responsibility

Data Collection	Jason	GEE API: Vegetation data
Documentation and Metadata	Nail	CAL FIRE Incident Dataset (Historical Wildfire Dataset)
	Nassim	Visual Crossing API: Weather data GEE API: Topography (Elevation and Slope Data)
	Rohini	Drought Data
Ethics and Legal Compliance	Jason	Ensure that the team meets and complies with the software and data acquired licenses throughout the lifetime of the project
Storage and Backup	Nail	File versioning as well as back up creation
Data Sharing	Rohini	Update GitHub Repository and Kaggle Datasets with datasets which are allowed to be published
Data Preservation	Nassim	Decision of which data to retain and preserve as well as storage cost management if applicable

2.2 Project Development Methodology

The CRISP-DM project methodology will be used for the project. Benefits of the CRISP-DM methodology include flexibility for iterative approaches, a more pronounced focus on business goals, and adaptability of objectives based on new findings. The CRISP-DM methodology consists of the following 6 steps: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Deployment.

The danger and impact of wildfires on society and the environment is extensive. Efficient mitigation efforts are key to limiting the damage of wildfires. The project hopes to produce a model that can accurately predict the severity of a wildfire based on initial meteorological and environmental conditions. The aim of this project is to produce a well-performing model and deploy it into production to aid first responders in resource management during the event of multiple wildfires occurring. Doing so will in turn help in wildfire damage mitigation.

The data used in the model will come from several sources such as CAL FIRE, NASA, gridMET, and Visual Crossing. The data originates from sources such as weather stations and satellites, which may be updated at different time intervals. The data includes historical wildfire, weather, topography, vegetation, and drought. The data included should include factors which contribute to the rate of wildfire spreading, such as wind speed, humidity, and slope of the terrain.

All the data will be filtered based on the dates and locations of when the wildfires happened. The data will be cleaned, for instance by removing outliers or imputing missing data. The datasets will be merged into one unified dataset based on date and location (latitude and longitude coordinates) of each wildfire instance. The features will be normalized, for instance through min-max scaling, to have all the features be on the same scale.

Once the unified dataset is ready for analysis, multiple models will be built and assessed. These include more standard machine learning models such as KNN, Random Forest (RF), SVM, XGBoost AdaBoost, along with an Artificial Neural Network. Further optimization of the models can be done by using k-fold cross validation as well as employing grid search cross validation for hyperparameter tuning. Each model will be judged based on predefined metrics such as the Matthews Correlation Coefficient (MCC), macro precision/recall/F1-score, and the AUC-ROC score. The best performing model will be deployed.

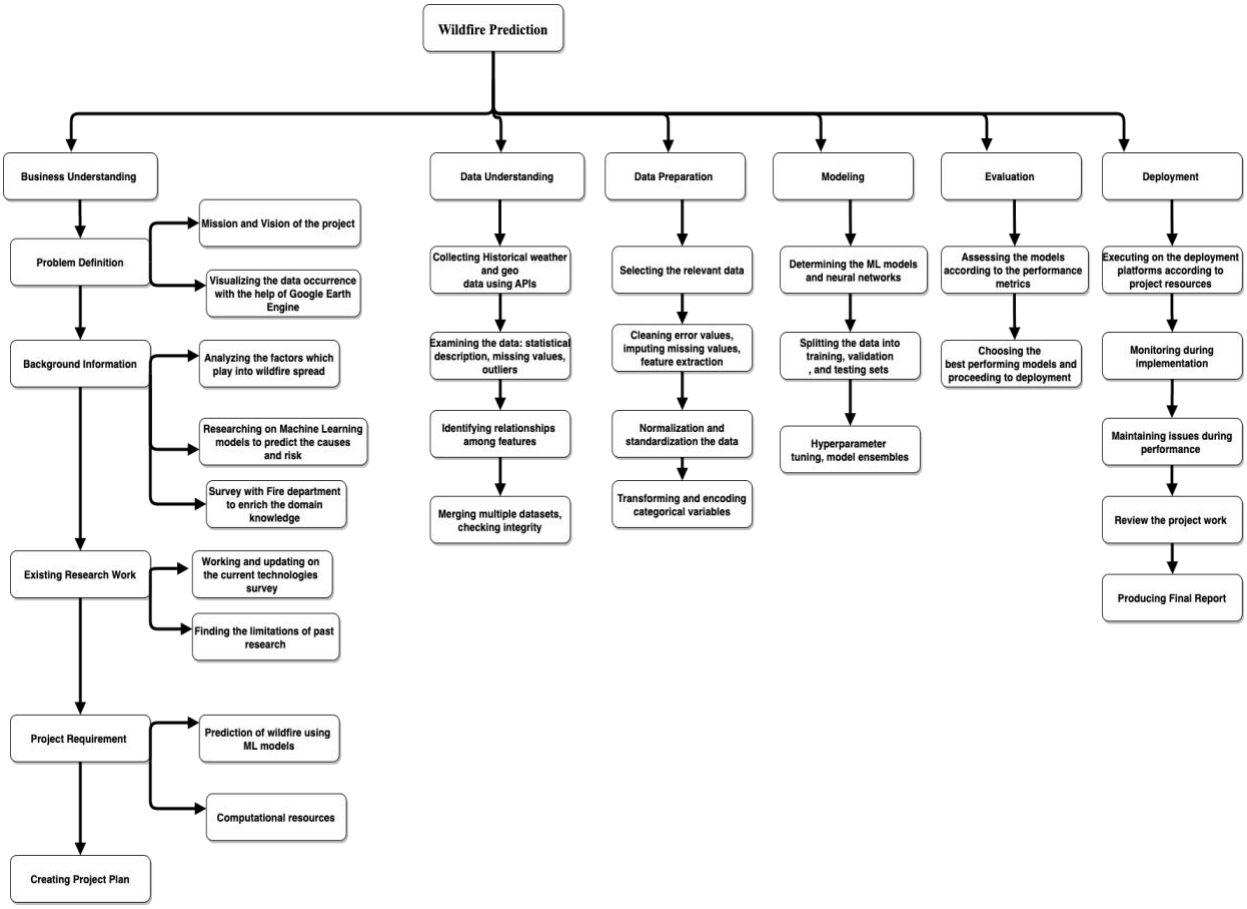
The deployment of the final model will be done through AWS. The model will be implemented in a production environment in order to make predictions on real-time wildfire data. The quality of the data and the model's performance will be continuously monitored. We also hope to deploy a web-based application through which users can obtain predictions of the severity of currently occurring wildfires.

2.3 Project Organization Plan

The WBS consists of six main tasks that align with the six stages of the CRISP-DM. Each step has its own set of associated tasks and deliverables, which helps us manage resources and time efficiently by dividing the project requirements into smaller, more manageable parts. The WBS is shown in Figure 2.

Figure 2

Work Breakdown Structure



The first step in any project is understanding the business problem. The problem and motivation must be well defined to be able to comprehend the objective and vision of the project. Background information will be gathered, variables assessed, and a review of current research papers conducted to understand related work better. Next, research into data sources and computing resources to collect information on machine learning algorithms, software, tools, and relevant expertise in similar research challenges to ensure all the necessary information needed is acquired. Finally, a comprehensive proposal that includes details on the project's timeframe, context, goals, and deliverables will be made.

Data understanding will go through collecting historical data and other factors using APIs such as the Google Earth Engine API. Examining the data will create statistical descriptions

which will help in identifying relationships between features. The data collected from multiple sources will be merged into a single dataset for the next phase.

The data preparation step will conduct data cleaning, feature selection, normalization of the features, and other data transformations such as encoding categorical variables. These steps will handle issues in the dataset to obtain the best-fit dataset for our machine-learning model and neural networks.

In terms of the Modeling and Evaluation phases, the dataset will be divided into training and testing sets, and k-fold cross validation and grid search will be employed on the training set in order to find the best performing set of hyperparameters. The best performing model will then be evaluated on the test set and the model will be evaluated using various metrics. For each machine learning model, the metrics will be compared, and a final determination will be made as to the best model. Finally, the best performing machine learning model will be deployed using Amazon Web Services.

2.4 Project Resource Requirements and Plan

Information on the purpose and estimated cost for every hardware and software resource, as well as for every tool/license, are elaborated on in Tables 3, 4, and 5.

Table 3

Hardware Requirements

Resource	Purpose	Estimated Cost
Local Machine	Navigating different tools efficiently, storage	No additional cost

Table 4

Software Requirements

Resource	Purpose	Estimated Cost
Python and associated libraries such as NumPy, pandas, seaborn, scikit-learn	Data Preprocessing, Data Visualization, Model Development	Free
Jupyter Notebook	Documenting and sharing code	Free
Git	Sharing code, version control	Free

Table 5

Tools and Licenses

Resource	Purpose	Estimated Cost
Tableau	Data Visualization	Free with student license
Diagrams.net	Creating the WBS	Free tier
Onlinegantt.com	Creating the Gantt Chart	Free
LucidChart	Creating the PERT chart	$9.99
Trello / ClickUp	Project Management	Free tier
Visual Crossing Data API	Historical weather data in a convenient format	$70

Google Earth Engine API	Geographical data	Free

A variety of different resources will be utilized in this project. Exploratory data analysis, data preprocessing, and model development will be performed on a local machine with a Jupyter notebook. To run the full end-to-end machine learning process with the entire data, a cloud solution such as AWS Sagemaker can be integrated if needed. Several tools, such as Tableau for data visualization and Trello for project management, either have a free license for students or a free tier option. To obtain historical weather data, the team will use Visual Crossing which has a Professional Tier option for $35/ month, and with the duration of the project being two months, will amount to $70. Overall, considering aspects such as the duration of the project, the size of the data, and the costs of various resources, we expect the total costs to complete the project to be around $100.

2.5 Project Schedule

Gantt Chart

A Gantt chart is a tool that displays a project's schedule, activities, subtasks, start dates, end dates, and resources. It uses bars to show how more minor activities are connected, giving a detailed view of the project. Our Gantt chart is divided into sections corresponding to CRISP-DM stages, making it easy to see the progress of the project. At the end of each step, a deliverable is produced according to the project plan, which runs horizontally across each phase. When necessary, the team may go back to a previous horizontal level, but the project culminates in one primary deliverable at the end of the final step. Each week, a deliverable is due, which helps the team to divide each phase into weekly sprints. Activities and subtasks are assigned to different team members for each sprint. The critical path is shown in the red box.

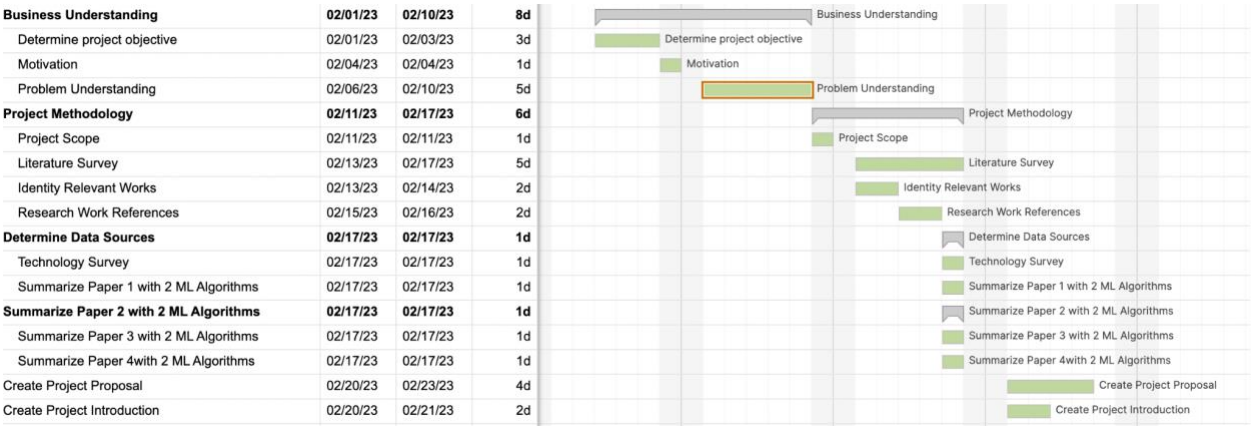
When estimating effort:

- There will not be any assignments due between March 28, 2022, and April 1, 2022, as that period is for spring break.
- Subtasks are assigned to a single resource based on availability, with a few exceptions where the whole team collaborates to complete a specific subtask. Each subtask is given a maximum of two days to be completed.

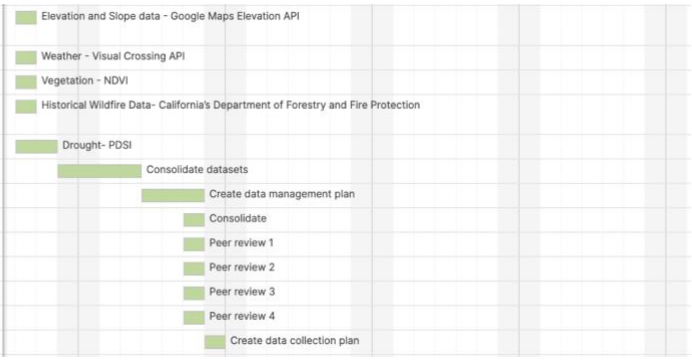
The Gantt Chart of the project is shown below in Figure 3.

Figure 3

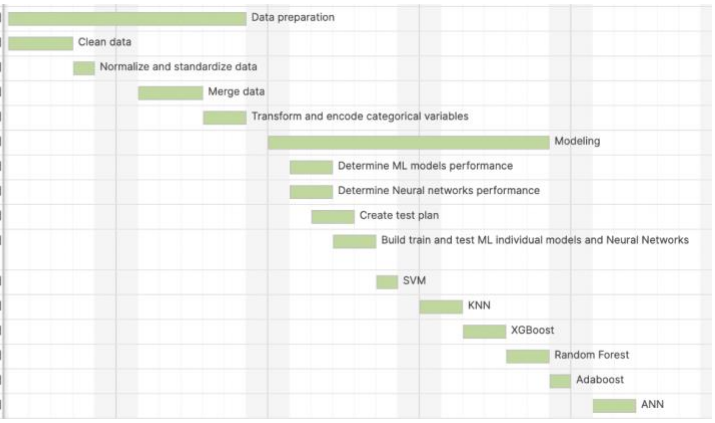
Gantt Chart



Elevation and Slope data - Google Maps Elevation API	03/16/23	03/16/23	1d
Weather - Visual Crossing API	03/16/23	03/16/23	1d
Vegetation - NDVI	03/16/23	03/16/23	1d
Historical Wildfire Data- California's Department of Forestry and Fire Protection	03/16/23	03/16/23	1d
Drought- PDSI	03/16/23	03/17/23	2d
Consolidate datasets	03/18/23	03/21/23	3d
Create data management plan	03/22/23	03/24/23	3d
Consolidate	03/24/23	03/24/23	1d
Peer review 1	03/24/23	03/24/23	1d
Peer review 2	03/24/23	03/24/23	1d
Peer review 3	03/24/23	03/24/23	1d
Peer review 4	03/24/23	03/24/23	1d
Create data collection plan	03/25/23	03/25/23	1d

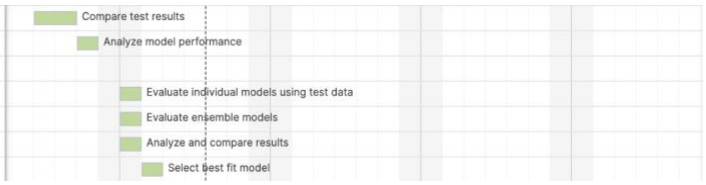


Data preparation	04/04/23	04/14/23	9d
Clean data	04/04/23	04/06/23	3d
Normalize and standardize data	04/07/23	04/07/23	1d
Merge data	04/10/23	04/12/23	3d
Transform and encode categorical variables	04/13/23	04/14/23	2d
Modeling	04/16/23	04/28/23	11d
Determine ML models performance	04/17/23	04/18/23	2d
Determine Neural networks performance	04/17/23	04/18/23	2d
Create test plan	04/18/23	04/19/23	2d
Build train and test ML individual models and Neural Networks	04/19/23	04/20/23	2d
SVM	04/21/23	04/21/23	1d
KNN	04/23/23	04/24/23	2d
XGBoost	04/25/23	04/26/23	2d
Random Forest	04/27/23	04/28/23	2d
Adaboost	04/29/23	04/29/23	1d
ANN	05/01/23	05/02/23	2d



Compare test results	05/03/23	05/04/23	2d
Analyze model performance	05/05/23	05/05/23	1d
Evaluation	05/07/23	10-May-22	1d
Evaluate individual models using test data	05/07/23	05/07/23	1d
Evaluate ensemble models	05/07/23	05/07/23	1d
Analyze and compare results	05/07/23	05/07/23	1d
Select best fit model	05/08/23	05/08/23	1d
Deployment	05/09/23	05/15/23	5d
Develop deployment plan	05/09/23	05/09/23	1d
Deploy ML models and neural networks	05/10/23	05/11/23	2d
Monitor model performance	05/12/23	05/12/23	1d
Maintenance issues during performance	05/13/23	05/13/23	1d
Review the project work	05/14/23	05/15/23	2d
Producing Final Report	05/15/23	05/15/23	1d

Compare test results	05/03/23	05/04/23	2d
Analyze model performance	05/05/23	05/05/23	1d
Evaluation	05/07/23	10-May-22	1d
Evaluate individual models using test data	05/07/23	05/07/23	1d
Evaluate ensemble models	05/07/23	05/07/23	1d
Analyze and compare results	05/07/23	05/07/23	1d
Select best fit model	05/08/23	05/08/23	1d



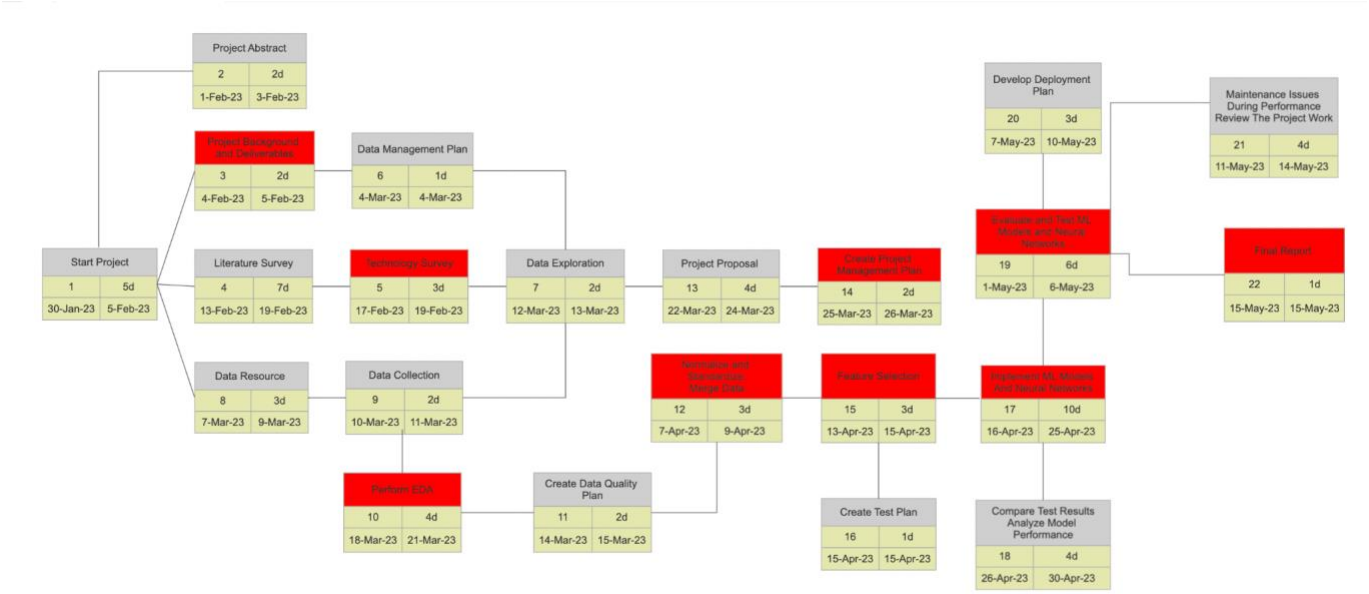
PERT Chart

The PERT chart helps to determine the scheduled time needed to finish all tasks by identifying the critical path and analyzing each individual task of the project. Red arrows indicate the critical path and red boxes mark the essential tasks. The chart will be used to

establish the timeline to cover the scope of the project. The PERT Chart of the project is shown below in Figure 4.

Figure 4

PERT Chart



3. Data Engineering

3.1 Data Process

To create a predictive model for wildfire severity classification, data engineering needs to be done to collect and prepare data for modeling purposes. Data collection is initially done to collect data from different sources relevant to the project. Data preprocessing is next done on the raw datasets for cleaning and validation. Data transformation and preparation are done to prepare the processed data for modeling. Data statistics is also done to summarize the data preparation, including raw, pre-processed, transformed, and prepared datasets, and statistically presenting the results in visualization formats. More on each of the steps will be explained in the upcoming sections of this chapter.

Data collection includes the selection of a historical wildfire dataset, in this case, the Incident Data collected from California's Department of Forestry and Fire Protection, which contains information such as the exact time, location, etc., for fire incidents that have happened in the state of California. The Incident data is initially filtered to narrow down the relevant incidents for severity classification. Unique features to each wildfire incident, such as the time and location of each wildfire, will then be fed through different API's and collection methods and be used as criteria for various data sources to collect wildfire factor data which in this case will include weather, vegetation index, drought index and elevation and slope data.

After the data is collected from multiple sources, data pre-processing is done to remove any noise or null values in the data which will affect the model's performance. This process can include removing irrelevant data from the collected datasets, such as the 'URL' column in the Incident Dataset, which contains a URL leading to more information on the selected fire incident. In the case of this project, the 'URL' column contains no relevant data in severity

classification, and therefore the column will be removed. Outliers are identified and dealt with whenever possible to reduce noise. Renaming columns is done to ease the data merging process between multiple datasets. Removing duplicates is also performed to reduce bias in the data.

With the data cleaned and preprocessed, data transformation and preparation will be done to the aggregated data to prepare the data for the model development and evaluation phase. This step will include creating a severity metric where each wildfire can be ranked by severity allowing a quantifiable metric to determine which wildfires are more ‘severe’ than the others. Normalization will make all the attributes on the same scale so that no attribute has an artificial influence on the model's predictions. Finally, the data will be prepared by splitting the data for training, testing, and validation purposes. The resulting dataset will be saved in CSV, where it will be ready for the subsequent phases.

3.2 Data Collection

As stated previously, the data required for wildfire severity classification will be collected from different sources, which contain information on historical wildfire data as well as on factors that affect wildfire spread which in turn affects wildfire severity. The California Fire Incident Dataset is taken from California’s Department of Forestry and Fire Protection. The dataset contained 2062 rows of fires from May 2009 to October 2022 in California. The published data was downloaded as a CSV and loaded into pandas, where certain key features from the dataset will be extracted for further use. Figures 5 and 6 show the features as well as the raw dataset of the incident data.

Figure 5

Wildfire Incident Data Features

```
df_calfire.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2062 entries, 0 to 2061
Data columns (total 23 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   incident_name                   2062 non-null   object
 1   incident_is_final               2062 non-null   object
 2   incident_date_last_update       2062 non-null   object
 3   incident_date_created           2062 non-null   object
 4   incident_administrative_unit    2050 non-null   object
 5   incident_administrative_unit_url  0 non-null    float64
 6   incident_county                 2052 non-null   object
 7   incident_location               2062 non-null   object
 8   incident_acres_burned           2023 non-null   float64
 9   incident_containment            2025 non-null   float64
 10  incident_control                123 non-null    object
 11  incident_cooperating_agencies   1585 non-null   object
 12  incident_longitude              2062 non-null   float64
 13  incident_latitude               2062 non-null   float64
 14  incident_type                   828 non-null    object
 15  incident_id                     2062 non-null   object
 16  incident_url                    2062 non-null   object
 17  incident_date_extinguished      1869 non-null   object
 18  incident_dateonly_extinguished  1869 non-null   object
 19  incident_dateonly_created       2062 non-null   object
 20  is_active                       2062 non-null   object
 21  calfire_incident                2062 non-null   bool
 22  notification_desired            2062 non-null   bool
dtypes: bool(2), float64(5), object(16)
```

Note. Historical wildfire data retrieved from California's Department of Forestry and Fire Protection.

Figure 6

Wildfire Incident Sample Data

```
df_calfire.head()
```

	incident_name	incident_is_final	incident_date_last_update	incident_date_created	incident_administrative_unit	incident_administrative_unit_url	incident_county
0	Bridge Fire	Y	2018-01-09T13:46:00Z	2017-10-31T11:22:00Z	Shasta-Trinity National Forest	NaN	Shasta
1	Pala Fire	Y	2020-09-16T14:07:35Z	2009-05-24T14:56:00Z	CAL FIRE San Diego Unit	NaN	San Diego
2	River Fire	Y	2022-10-24T11:39:23Z	2013-02-24T08:16:00Z	CAL FIRE San Bernardino Unit	NaN	Inyo
3	Fawnskin Fire	Y	2013-04-22T09:00:00Z	2013-04-20T17:30:00Z	San Bernardino National Forest	NaN	San Bernardino
4	Gold Fire	Y	2013-05-01T07:00:00Z	2013-04-30T12:59:00Z	CAL FIRE Madera-Mariposa-Merced Unit	NaN	Madera

incident_location	incident_acres_burned	incident_containment	...	incident_latitude	incident_type	incident_id	incident_url
I-5 and Turntable Bay, 7 miles NE of Shasta Lake	37.0	100.0	...	40.774000	NaN	2ca11d45-8139-4c16-8af0-880d99b21e82	https://www.fire.ca.gov/incidents/2017/10/31/b...
Hwy 76 and Pala Temecula, northwest of Pala	122.0	100.0	...	1.000000	Wildfire	8f61f461-552d-4538-b186-35ab030da416	https://www.fire.ca.gov/incidents/2009/5/24/pa...
south of Narrow Gauge Rd & north of Hwy 136, e...	407.0	100.0	...	36.602575	NaN	094719ba-a47b-4abb-9ec5-a506b2b9fd23	https://www.fire.ca.gov/incidents/2013/2/24/ri...
west of Delamar Mountain, north of the communi...	30.0	100.0	...	34.288877	NaN	58f89ff8-bd3e-4355-b1c0-8fa05c747d3f	https://www.fire.ca.gov/incidents/2013/4/20/fa...
Between Road 210 and Road 200 near Fine Gold C...	274.0	100.0	...	37.116295	NaN	357ffc13-bef9-48eb-810f-c5de851972eb	https://www.fire.ca.gov/incidents/2013/4/30/go...

incident_date_extinguished	incident_dateonly_extinguished	incident_dateonly_created	is_active	calfire_incident	notification_desired
2018-01-09T13:46:00Z	2018-01-09	2017-10-31	N	False	False
2009-05-25T00:00:00Z	2009-05-25	2009-05-24	N	True	False
2013-02-28T20:00:00Z	2013-02-28	2013-02-24	N	True	False
2013-04-22T09:00:00Z	2013-04-22	2013-04-20	N	False	False
2013-05-01T07:00:00Z	2013-05-01	2013-04-30	N	True	False

Note. Sample of data of historical wildfire dataset.

The initial raw dataset contains 2062 rows and 23 columns. There are vital features to be highlighted in this dataset. The ‘incident_name’ column contains the names of the fires which has happened; the ‘incident_is_final’ column indicates whether the fire incident is resolved or not, ‘incident_date_created’ contains the incident date creation, ‘incident_location’ has a description of the location of the incident, ‘incident_acres_burned’ which shows how many of acres is damaged by the fire, ‘incident_longitude’ and ‘incident_latitude’ indicates the longitude and latitude of the incident, ‘incident_type’ identifies each incident type as fire or wildfire, ‘incident_date_extinguished’ provides the date and time at which the fire has been completely extinguished. In this case, the raw dataset is narrowed to records with ‘incident_is_final’ as ‘Y’ for yes and ‘incident_type’ as ‘Wildfire.’ It will narrow down to records where wildfire is confirmed and wholly extinguished. The severity of the wildfire is measured through the incident and must be finalized and not ongoing. With these filters, the records are narrowed down to 756 wildfire incidents.

In order to collect data on factors of wildfire spread, such as weather, key features must be selected which will help data retrieval. The selected set of features unique to every wildfire incident and quickly fed into API’s or other data collection methods for data retrieval would be the location and date of each wildfire incident. This data would be represented in the columns ‘incident_longitude,’ ‘incident_latitude,’ and ‘incident_dateonly_created’ columns, similar to the ‘incident_date_created’ column only containing the date of the incident instead of the date and time.

Besides historical wildfires, data from wildfire spread factors need to be collected to see if any of these factors will affect the severity of the wildfire. This project will consider the

weather, vegetation index (NDVI), drought index (PDSI), elevation, and slope as factors of wildfire severity. “The effect of climate change on wildfire severity will depend on pre-suppression activities, fire suppression strategies, human settlement patterns, the degree of climate change, and how these affect vegetation type and fuel loading” (Fried, J.S.,2004; Torn, M.S. & Mills, E., 2004, p.188).

Wildfire is the spread of fire; hot weather conditions or wind conditions will be able to affect the rate and direction of wildfire spread. The weather data is collected from the Visual Crossing API, where the API is fed with the latitude, longitude, and start date of the wildfire incident using a Python script. The API will return the location's weather information for that specific date. Figures 7 - 9 show the fetched weather data given the time and place of the wildfire as well as statistical descriptions of the data with the sample data.

Figure 7

Weather Data Features

```
df_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 798 entries, 0 to 797
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Latitude             798 non-null    float64
 1   Longitude            798 non-null    float64
 2   Date                 798 non-null    object 
 3   Maximum Temperature  792 non-null    float64
 4   Minimum Temperature  792 non-null    float64
 5   Temperature          792 non-null    float64
 6   Dew Point            791 non-null    float64
 7   Wind Chill           76 non-null     float64
 8   Heat Index           596 non-null    float64
 9   Precipitation        798 non-null    float64
 10  Precipitation Cover  792 non-null    float64
 11  Snow                 652 non-null    float64
 12  Snow Depth           652 non-null    float64
 13  Wind Speed           792 non-null    float64
 14  Wind Gust            612 non-null    float64
 15  Wind Direction       792 non-null    float64
 16  Visibility           791 non-null    float64
 17  Cloud Cover          791 non-null    float64
 18  Relative Humidity    791 non-null    float64
 19  Sea Level Pressure   765 non-null    float64
 20  Solar Radiation      777 non-null    float64
 21  Solar Energy         777 non-null    float64
 22  Weather Type         364 non-null    object 
 23  Info                 6 non-null      object 
 24  Conditions           792 non-null    object 
dtypes: float64(21), object(4)
```

Note. Data retrieved from the Visual Crossing API.

Figure 8

Weather Data Statistical Descriptions

```
df_weather.describe()
```

	Latitude	Longitude	Maximum Temperature	Minimum Temperature	Temperature	Dew Point	Wind Chill	Heat Index	Precipitation	Precipitation Cover
count	798.000000	798.000000	792.000000	792.000000	792.000000	791.000000	76.000000	596.000000	798.000000	792.000000
mean	37.397804	-120.261336	87.398359	58.554924	72.483586	43.983818	38.192105	89.694463	0.004173	0.615568
std	2.602920	2.726204	11.807131	9.992981	10.264877	12.086273	9.993034	6.832880	0.090475	4.094303
min	32.557546	-124.199540	50.800000	13.400000	30.800000	-8.600000	7.200000	78.300000	0.000000	0.000000
25%	34.864043	-121.913804	80.175000	52.700000	66.300000	38.400000	32.850000	83.975000	0.000000	0.000000
50%	37.821198	-120.815586	89.000000	58.500000	73.000000	46.800000	41.950000	89.300000	0.000000	0.000000
75%	39.347725	-118.796281	96.250000	65.200000	79.900000	52.500000	45.575000	94.500000	0.000000	0.000000
max	44.734000	-80.491630	116.000000	86.800000	101.800000	65.900000	48.100000	110.600000	2.550000	91.670000

Snow Depth	Wind Speed	Wind Gust	Wind Direction	Visibility	Cloud Cover	Relative Humidity	Sea Level Pressure	Solar Radiation	Solar Energy
652.000000	792.000000	612.000000	792.000000	791.000000	791.000000	791.000000	765.000000	777.000000	777.000000
0.013451	14.875253	26.408987	207.322702	9.512389	11.639317	42.937901	1013.140000	405.149550	23.853668
0.207422	5.431155	8.160138	63.411265	1.204873	17.756636	16.798429	3.673026	103.121403	6.993188
0.000000	3.300000	11.400000	40.000000	3.100000	0.000000	6.290000	999.300000	35.500000	1.500000
0.000000	11.100000	20.800000	161.667500	9.500000	0.000000	30.290000	1010.900000	337.700000	18.900000
0.000000	14.000000	25.150000	205.595000	9.900000	3.200000	40.830000	1012.900000	421.100000	24.500000
0.000000	17.525000	31.100000	256.565000	9.900000	15.700000	53.635000	1015.200000	484.600000	28.500000
3.940000	42.500000	68.400000	340.130000	21.200000	99.800000	97.600000	1028.500000	634.300000	52.300000

Note. Statistical description of the weather dataset.

Figure 9

Fetched Weather Sample Data

```
df_weather.head()
```

	Latitude	Longitude	Date	Maximum Temperature	Minimum Temperature	Temperature	Dew Point	Wind Chill	Heat Index	Precipitation
0	38.56910	-120.78190	2014-07-25	92.5	69.5	80.8	40.0	NaN	88.1	0.00
1	38.49940	-122.11450	2015-07-22	82.2	61.5	70.0	53.0	NaN	81.3	0.00
2	38.32974	-120.70418	2015-09-09	99.2	69.1	83.9	37.1	NaN	94.0	0.00
3	38.39206	-122.24367	2017-10-09	82.1	56.8	70.3	24.2	NaN	79.7	0.01
4	38.60895	-122.62879	2017-10-08	85.6	48.3	66.6	32.2	46.5	82.5	0.00

Wind Direction	Visibility	Cloud Cover	Relative Humidity	Sea Level Pressure	Solar Radiation	Solar Energy	Weather Type	Info	Conditions
169.83	9.9	12.3	25.44	NaN	NaN	NaN	NaN	NaN	Clear
217.50	9.9	3.8	56.91	1008.0	NaN	NaN	NaN	NaN	Clear
186.33	9.7	18.7	20.57	1009.6	NaN	NaN	Smoke Or Haze	NaN	Clear
135.04	7.8	36.5	19.90	1011.8	NaN	NaN	Light Rain, Smoke Or Haze	NaN	Rain, Partially cloudy
157.17	9.8	2.5	40.06	1008.2	NaN	NaN	NaN	NaN	Clear

Note. Sample of the weather dataset.

Since the API is fed through incident time and locations for 756 records, the API fetched weather data for 756 records. Twenty-two weather variables are returned Maximum Temperature, Minimum Temperature, Temperature, Dew Point, Wind Chill, Heat Index, Precipitation, Precipitation Cover, Snow, Snow Depth, Wind Speed, Wind Gust, Wind Direction,

Visibility, Cloud Cover, Relative Humidity, Sea Level Pressure, Solar Radiation, Solar Energy, Weather Type, Info, and Conditions. The Data Preprocessing section will explain different feature selections and column removal of these variables. These data are retrieved with imperial units. The vegetation index is taken from the MOD13A1.061 Terra Vegetation Indices 16-Day Global 500m dataset. This dataset is provided by NASA which contains the Normalized Difference Vegetation Index (NDVI) values at a per pixel basis at 500-meter spatial resolution and updated every 16 Days." The NDVI quantifies vegetation by measuring the difference between near-infrared (which vegetation strongly reflects) and red light (which vegetation absorbs" (GISGeography, May 30, 2022).

This dataset is accessible by the Google Earth Engine API, where latitude, longitude, and date are fed through the API. The API will return NDVI values of the nearest location and the latest data before the wildfire incident within the 16-day data update lifecycle. Figures 10 - 13 show the fetched data features, statistical description and sample data through the Google Earth Engine API.

Figure 10

Vegetation Index Data Features

```
df_vegetation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 798 entries, 0 to 797
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Latitude   798 non-null    float64
 1   Longitude  798 non-null    float64
 2   Date       798 non-null    object
 3   NDVI       794 non-null    float64
dtypes: float64(3), object(1)
```

Note. Vegetation index data fetched from the NASA Earthdata API.

Figure 11

Vegetation Index Data Statistical Descriptions

```
df_vegetation.describe()
```

	Latitude	Longitude	NDVI
count	798.000000	798.000000	794.000000
mean	37.397804	-120.261336	3935.763224
std	2.602920	2.726204	1635.737992
min	32.557546	-124.199540	626.000000
25%	34.864043	-121.913804	2661.250000
50%	37.821198	-120.815586	3540.500000
75%	39.347725	-118.796281	4953.750000
max	44.734000	-80.491630	8982.000000

Note. Statistical description of the vegetation index data.

Figure 12

Vegetation Index Sample Data

```
df_vegetation.head()
```

	Latitude	Longitude	Date	NDVI
0	38.56910	-120.78190	2014-07-25	4653.0
1	38.49940	-122.11450	2015-07-22	4277.0
2	38.32974	-120.70418	2015-09-09	5661.0
3	38.39206	-122.24367	2017-10-09	5227.0
4	38.60895	-122.62879	2017-10-08	5951.0

Note. Sample of the vegetation index data.

According to the U.S. Department of Commerce’s National Oceanic and Atmospheric Administration, “Climate change, including increased heat, extended drought, and a thirsty atmosphere, has been a key driver in increasing the risk and extent of wildfires in the western United States during the last two decades" (https://www.noaa.gov/noaa-wildfire/wildfire-climate-

connection). This project will consider the drought index data during the wildfire incident to see potential correlations in the severity. The drought index is taken from the gridMET dataset. The gridMET dataset contains daily surface meteorological data from the contiguous United States, with a 4 km resolution. The specific drought index obtained is the Palmer Drought Severity Index, or PDSI, updated every ten days. It is a standardized index that generally spans from -10 (dry) to +10 (wet) and is calculated based on temperature and precipitation data, along with the local Available Water Content (AWC) of the soil. The drought index data is not obtained through an API. Instead, it is obtained from a file courtesy of climatologylab.org, which hosts gridMET data files. The data files are in a NetCDF format, and the specific file used to get the data was "pdsi.nc". The "xarray" Python package was utilized to be able to read the file. A brief overview of the file's contents is shown in Figure 13.

Figure 13

Overview of the contents of the "pdsi.nc" file

```
import xarray as xr
ds = xr.open_dataset('pdsi.nc')
print(ds)
```

```
<xarray.Dataset>
Dimensions:   (lon: 1386, lat: 585, day: 3157, crs: 1)
Coordinates:
  * lon       (lon) float64 -124.8 -124.7 -124.7 -124.6 ... -67.14 -67.1 -67.06
  * lat       (lat) float64 49.4 49.36 49.32 49.28 ... 25.19 25.15 25.11 25.07
  * day       (day) datetime64[ns] 1980-01-05 1980-01-10 ... 2023-03-31
  * crs       (crs) uint16 3
Data variables:
    pdsi      (day, lat, lon) float32 ...
    category  (day, lat, lon) float32 ...
Attributes: (12/20)
    geospatial_bounds_crs:      EPSG:4326
    Conventions:                CF-1.6
    geospatial_bounds:          POLYGON((-124.7666666333333 49.40000000000000...
    geospatial_lat_min:         25.066666666666666
    geospatial_lat_max:         49.40000000000000
    geospatial_lon_min:         -124.7666666333333
```

Note. Overview of the contents of gridMET drought index file.

After the file was read, a custom function was written to obtain PDSI values for certain latitude, longitude, and date combinations. To obtain PDSI values for every record in the wildfire dataset, the latitude and longitude values in the wildfire dataset were rounded to the nearest

values that were present in the data file. As for the date, the date of the wildfire incident was rounded to the nearest date in the file that occurred before the wildfire. For instance, if a wildfire occurred on 1/14/18, and the data file contained PDSI values for 1/10/18 and 1/15/18, the corresponding PDSI value for 1/10/18 will be extracted, considering the nearest latitude and longitude coordinates. After the PDSI value is obtained for each wildfire record, the results are exported to a CSV file. Figures 14 - 16 show the resulting data taken from gridMET.

Figure 14

Drought Index Data Features

```
df_drought.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Latitude   800 non-null    float64
 1   Longitude  800 non-null    float64
 2   Date       800 non-null    object 
 3   PDSI       797 non-null    float64
dtypes: float64(3), object(1)
memory usage: 25.1+ KB
```

Note. Overview of features of PDSI dataset.

Figure 15

Drought Index Data Statistical Descriptions

```
df_drought.describe()
```

	Latitude	Longitude	PDSI
count	800.000000	8.000000e+02	797.000000
mean	37.199917	-1.489388e+06	-1.391455
std	6.093403	4.212286e+07	2.779766
min	-118.461150	-1.191415e+09	-8.520000
25%	34.841023	-1.219165e+02	-3.420000
50%	37.817219	-1.208156e+02	-1.930000
75%	39.333215	-1.187938e+02	1.340000
max	44.734000	-8.049163e+01	6.629999

Note. Statistical description of PDSI dataset.

Figure 16

Drought Index Sample Data

```
df_drought.head()
```

	Latitude	Longitude	Date	PDSI
0	38.56910	-120.78190	2014-07-25	-3.190000
1	38.49940	-122.11450	2015-07-22	-3.450000
2	38.32974	-120.70418	2015-09-09	-2.640000
3	38.39206	-122.24367	2017-10-09	1.180000
4	38.60895	-122.62879	2017-10-08	1.450001

Note. Sample records of PDSI dataset.

According to the North West Fire Science Consortium, which published the 'FIREFACTS Topography,' "Fire usually spreads faster uphill than downhill because fuels are more efficiently preheated by uphill spreading of heat and flames. The steeper the slope, the faster the fire can burn." It is also stated that "Elevation can influence fire behavior in several ways: the amount and timing of precipitation, heat, wind exposure, and context to the surrounding land" (https://www.nwfirescience.org/). With this information in mind, the slope and elevation are determined to be wildfire spread factors data that need to be collected. Using the Google Earth Engine API, the latitude and longitude are fed through, and the elevation response at the location is given in meters along with the slope which is given in degrees. Figure 17 - 19 shows the data collected from the API and script.

Figure 17

Elevation and Slope Data Features

```
df_elevation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 798 entries, 0 to 797
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Latitude         798 non-null    float64
 1   Longitude        798 non-null    float64
 2   Elevation (m)    798 non-null    float64
 3   Slope (degrees)  798 non-null    float64
dtypes: float64(4)
```

Note. Elevation and slope data fetched from GEE API.

Figure 18

Elevation and Slope Data Statistical Descriptions

```
df_elevation.describe()
```

	Latitude	Longitude	Elevation (m)	Slope (degrees)
count	798.000000	798.000000	798.000000	798.000000
mean	37.397804	-120.261336	569.014247	8.693132
std	2.602920	2.726204	550.406110	9.573419
min	32.557546	-124.199540	-65.955147	0.000000
25%	34.864043	-121.913804	164.444031	2.073451
50%	37.821198	-120.815586	401.568802	4.573274
75%	39.347725	-118.796281	792.445740	12.123724
max	44.734000	-80.491630	2910.233887	58.204086

Note. Statistical description of elevation and slope dataset.

Figure 19

Elevation Data and Sample of data

```
df_elevation.head()
```

	Latitude	Longitude	Elevation (m)	Slope (degrees)
0	38.56910	-120.78190	408.909271	7.969696
1	38.49940	-122.11450	283.758820	33.376866
2	38.32974	-120.70418	537.675354	5.580828
3	38.39206	-122.24367	418.207397	3.464834
4	38.60895	-122.62879	200.451569	15.971034

Note. Sample of data of elevation and slope dataset.

The data collected from different sources are merged based on the wildfire incident's latitude, longitude, and date. The aggregated dataset will then be preprocessed for further use; more on this is in the next section.

3.3 Data Pre-Processing

The aggregated data needs to be preprocessed and cleaned to remove any possible null values, irrelevant data, or duplicates. The cleaned data will maximize performance when used for model evaluation. Preprocessing will start with the removal of irrelevant columns from the aggregated dataset. In this case, the removed columns will be irrelevant features in the wildfire and weather data.

The following are the dropped features deemed irrelevant to wildfire spread for the machine learning models. For the wildfire incident data, the following columns are dropped as they are determined to be irrelevant for wildfire severity classification: 'incident_is_final,' 'incident_date_last_update,' 'incident_administrative_unit,' 'incident_administrative_unit_url,' 'incident_location,' 'incident_containment,' 'incident_control,' 'incident_cooperating_agencies, "incident_type,' 'incident_id,' 'incident_url,' 'incident_dateonly_extinguished,' 'is_active,' 'calfire_incident,' and 'notification_desired.' For the weather data, the following columns are dropped, 'Wind Chill,' 'Heat Index,' 'Precipitation Cover,' 'Snow,' 'Snow Depth,' 'Visibility', 'Sea Level Pressure,' 'Weather Type', 'Info' and 'Conditions.' The mentioned weather and wildfire columns are dropped as they do not pose as large factors in wildfire spread. The existence of duplicates would mean that the training and testing may be contaminated with each other. Duplicates are checked and dropped. Figure 20 shows the columns left after dropping irrelevant features.

Figure 20

Dataset Columns after Dropping Irrelevant Features.

```
merged_data_final2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 772 entries, 0 to 805
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Name                 772 non-null    object
 1   County               771 non-null    object
 2   Date                 772 non-null    object
 3   Start Date and Time  772 non-null    datetime64[ns, UTC]
 4   End Date and Time    611 non-null    datetime64[ns, UTC]
 5   Latitude             772 non-null    float64
 6   Longitude            772 non-null    float64
 7   Acres Burned         756 non-null    float64
 8   Duration             611 non-null    float64
 9   Maximum Temperature  766 non-null    float64
 10  Minimum Temperature  766 non-null    float64
 11  Temperature          766 non-null    float64
 12  Dew Point            765 non-null    float64
 13  Precipitation        772 non-null    float64
 14  Wind Speed           766 non-null    float64
 15  Wind Gust            589 non-null    float64
 16  Wind Direction       766 non-null    float64
 17  Cloud Cover          765 non-null    float64
 18  Relative Humidity    765 non-null    float64
 19  Solar Radiation      751 non-null    float64
 20  Solar Energy         751 non-null    float64
 21  PDSI                 771 non-null    float64
 22  NDVI                 768 non-null    float64
 23  Elevation            772 non-null    float64
 24  Slope                772 non-null    float64
dtypes: datetime64[ns, UTC](2), float64(20), object(3)
memory usage: 156.8+ KB
```

Note. Columns after dropping irrelevant features.

With the irrelevant features and duplicates removed, the next step is dealing with missing or null values in the aggregated dataset. These missing values can create bias in the dataset and affect the model's accuracy later in the model development phase. Since the project aims to create a severity metric for wildfires, the key feature for this severity classification is the ‘Acres Burned’ attribute. More on the severity metric is to be explained in the Data Transformation stage; however, records with missing values need to be removed as there would not be any way to measure how severe the wildfire is without using the ‘Acres Burned’ feature. Figure 21 shows records with missing values in the ‘Acres Burned’ field.

Figure 21

Records with missing values in acres burned.

```
missing_values_count = df['Acres Burned'].isna().sum()
print(f"Number of missing values in the 'Acres Burned' column: {missing_values_count}")
```

```
Number of missing values in the 'Acres Burned' column: 16
```

Note. Count of missing values in ‘Acres Burned’ feature.

Next, other fields with missing values are searched throughout the dataset. The following fields have missing values in the dataset. Figure 22 shows the number of missing values per field. Due to the relatively small number of records containing the number of non-missing values, the records with missing values for the columns ‘Dew Point,' 'Cloud Cover,' 'Relative Humidity,' 'Solar Radiation,' 'Solar Energy,' 'PDSI,' 'NDVI' are then dropped along with records with missing ‘Acres Burned.’

Figure 22

Number of rows with non-missing values for each column

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 766 entries, 0 to 765
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Name                 766 non-null    object
 1   County               765 non-null    object
 2   Date                 766 non-null    object
 3   Start Date and Time  766 non-null    object
 4   End Date and Time    607 non-null    object
 5   Latitude             766 non-null    float64
 6   Longitude            766 non-null    float64
 7   Acres Burned         750 non-null    float64
 8   Duration             607 non-null    float64
 9   Maximum Temperature  766 non-null    float64
 10  Minimum Temperature  766 non-null    float64
 11  Temperature          766 non-null    float64
 12  Dew Point            765 non-null    float64
 13  Precipitation        766 non-null    float64
 14  Wind Speed           766 non-null    float64
 15  Wind Gust            589 non-null    float64
 16  Wind Direction       766 non-null    float64
 17  Cloud Cover          765 non-null    float64
 18  Relative Humidity    765 non-null    float64
 19  Solar Radiation      751 non-null    float64
 20  Solar Energy         751 non-null    float64
 21  PDSI                 765 non-null    float64
 22  NDVI                 763 non-null    float64
 23  Elevation            766 non-null    float64
 24  Slope                766 non-null    float64
dtypes: float64(20), object(5)
memory usage: 149.7+ KB
```

Note. Number of rows in features with non-missing values.

Finally, only the Wind Gust feature is left with a relatively large amount of missing values, with 177 values missing. Wind Gust refers to the maximum wind speed measured in a short amount of time. Checking with the Visual Crossing Weather API documentation, a null value is returned if the maximum wind speed does not exceed 11 mph or 18 kph over the mean wind speed for that day. Therefore, the records with missing values for Wind Gusts will be replaced with data from the Wind Speed feature, as the maximum wind speed possible would be very similar to the value of the Wind Speed. A final check is done on the missing values and related features needed for model development. The columns ‘End Date and Time’, ‘Duration,’ ‘County,’ and ‘Start Date and Time' were dropped as they had missing values and were

irrelevant factors to wildfire severity. Figure 23 shows the information about the final preprocessed and cleaned aggregated dataset ready for the Data Transformation phase.

Figure 23

Information on the Cleaned Data Set

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Name                 731 non-null    object
 1   Date                 731 non-null    object
 2   Latitude             731 non-null    float64
 3   Longitude            731 non-null    float64
 4   Acres Burned         731 non-null    float64
 5   Maximum Temperature  731 non-null    float64
 6   Minimum Temperature  731 non-null    float64
 7   Temperature          731 non-null    float64
 8   Dew Point            731 non-null    float64
 9   Precipitation        731 non-null    float64
 10  Wind Speed           731 non-null    float64
 11  Wind Gust            731 non-null    float64
 12  Wind Direction       731 non-null    float64
 13  Cloud Cover          731 non-null    float64
 14  Relative Humidity    731 non-null    float64
 15  Solar Radiation      731 non-null    float64
 16  Solar Energy         731 non-null    float64
 17  PDSI                 731 non-null    float64
 18  NDVI                 731 non-null    float64
 19  Elevation            731 non-null    float64
 20  Slope                731 non-null    float64
dtypes: float64(19), object(2)
memory usage: 120.1+ KB
```

Note. Count of non-null rows in the dataset after cleaning.

3.4 Data Transformation

Transformation of the dataset is next done to transform the pre-processed datasets into desired formats needed for the preparation phase. With the cleaned data, a severity metric needs to be devised to rank each of the wildfires based on severity, allowing the model to do multiclass classification based on the severity metric chosen for this project. The severity metric

implemented will use the ‘Acres Burned’ column as the primary indicator of wildfire damage and severity. Three methods were used for the severity metric: Range normalization, Percentile Method and Manual Classification.

The Range Normalization method uses the normalized range of the ‘Acres Burned’ field between 0 and 1 (min-max scaling). The records are binned between 0,0.2,0.4,0.6,0.8 and 1. Values between 0 and 0.2 will fall under Rank 1, values between 0.2 and 0.4 will fall under Rank 2, values between 0.4 and 0.6 will fall under Rank 3, values between 0.6 and 0.8 will fall under Rank 4, and values between 0.8 and 1 will fall under Rank 5. Rank 1 is the least severe, and Rank 5 is the most severe.

The Percentile Method has the ‘Acres Burned’ sorted from lowest to highest. Data that falls under the 20th percentile will be assigned Rank 1; records between the 20th and 40th percentile will be assigned Rank 2; records between the 40th and 60th percentile will be assigned Rank 3, records between the 60th and 80th percentile be assigned Rank 4, and records between 80th and 100th percentile will be assigned Rank 5. Rank 1 is the least severe, and Rank 5 is the most severe.

The Manual Classification method refers to the size class of fire used by the National Wildfire Coordinating Group (NWCG). “The National Wildfire Coordinating Group provides national leadership to enable interoperable wildland fire operations among federal, state, local, tribal, and territorial partners” (https://www.nwcg.gov/). The scale used by the NWCG is shown in Table 6.

Table 6

NWCG Size Class of Fire

Fire Class	Acres Affected
Class A	One-fourth acre or less
Class B	More than one-fourth acre, but less than 10 acres
Class C	10 acres or more, but less than 100 acres
Class D	100 acres or more, but less than 300 acres
Class E	300 acres or more, but less than 1,000 acres
Class F	1,000 acres or more, but less than 5,000 acres
Class G	5,000 acres or more

Note. Severity class of fire according to NWCG metric.

However, after trying to implement the NWCG fire classification to the cleaned dataset, there were no acres burned which meets the criteria of Class A. Therefore, a slight modification to the NWCG fire classification scale is made and shown in Table 7.

Table 7

Modified Size Class of Fire

Fire Class	Acres Affected
Class A	More than one-fourth acre, but less than 10 acres
Class B	10 acres or more, but less than 100 acres
Class C	100 acres or more, but less than 300 acres
Class D	300 acres or more, but less than 1,000 acres
Class E	1,000 acres or more, but less than 5,000 acres
Class F	5,000 acres or more

Note. Modified Size Class of Fire.

The three fire classification methods were implemented into the cleaned dataset and compared to select the most promising severity metric used for multiclass classification model evaluation. The next step was to choose the most appropriate severity metric.

With the “Range Normalization” method, most of the wildfires were assigned as Class 1 and thus is not a good metric in this case. Figure 24 shows the distribution of scores using the “Range Normalization” method.

Figure 24

Distribution of classes with the "Range Normalization" method

```
df['Wildfire_Scale_Acres_Normalized'].value_counts()
```

```
1    722
2      7
5      2
3      0
4      0
Name: Wildfire_Scale_Acres_Normalized, dtype: int64
```

Note. Distribution of classes with the "Range normalization" method.

The distribution of classes with the "Percentile" and the modified "NWCG" scale are shown in Figures 25 and 26.

Figure 25

Distribution of classes with the "Percentile" method

```
df['Wildfire_Scale_Acres_Quantiles'].value_counts()
```

```
1    156
5    146
3    145
4    145
2    139
Name: Wildfire_Scale_Acres_Quantiles, dtype: int64
```

Note. Distribution of classes with the "Percentile" method.

Figure 26

Distribution of classes with the modified "NWCG" scale

```
df['Wildfire_Scale_Acres_NWCG'].value_counts()
```

```
B    321
C    179
D     94
F     69
E     63
A      5
Name: Wildfire_Scale_Acres_NWCG, dtype: int64
```

Note. Distribution of classes with the "NWCG" method.

Although the “Percentile” method results in a more favorable distribution of classes, it will vary depending on the dataset. The modified “NWCG” scale has an appropriate distribution of classes. It is a more objective measure of classifying wildfires than the “Percentile” method. Thus, the modified “NWCG” scale is the chosen method to rank the severity of each wildfire.

In this case, the “Acres Burned” field is then dropped as it was initially used to create the target variable, the severity classification.

An iterative approach to calculating the variance inflation factor for each feature was used to reduce the number of predictor variables in the dataset and deal with multicollinearity. At each iteration, the variable with the highest inflation factor more significant than ten was removed, and the variance inflation factor of the remaining variable was recalculated. This process continued until all the variables left had a VIF of less than 10. Figure 27 shows the results at each iteration.

Figure 27

VIF values of dropped variables at each iteration and the VIF of the remaining variables.

```
Removed variable 'Temperature' with VIF 3544.95
Removed variable 'Minimum Temperature' with VIF 142.74
Removed variable 'Maximum Temperature' with VIF 65.24
Removed variable 'Solar Radiation' with VIF 57.18
Removed variable 'Dew Point' with VIF 29.00
Removed variable 'Wind Speed' with VIF 20.82
Removed variable 'Wind Direction' with VIF 11.69

Final VIF for the remaining variables:
            Variable       VIF
0      Precipitation  1.060377
1          Wind Gust  4.633861
2        Cloud Cover  2.093150
3  Relative Humidity  9.393570
4       Solar Energy  9.071706
5               PDSI  1.345320
6               NDVI  6.616311
7          Elevation  2.102603
8              Slope  2.138025
```

Note. VIF values of dropped variables at each iteration and the VIF of the remaining variables.

After reducing the dimensions in the dataset, normalization is done on the cleaned dataset. As mentioned previously, normalization will ensure that each variable is on a Standard scale, thus ensuring that no variable has a minor influence on the model's predictions. Figure 28 shows the transformed dataset.

Figure 28

Dataset post dimensionality reduction and normalization

	Name	Date	Latitude	Longitude	Precipitation	Wind Gust	Cloud Cover	Relative Humidity	Solar Energy	PDSI	NDVI	Elevation	Slope	Severity_Class
0	IronGate Fire	2019-06-16	41.946220	-122.401570	0.0	0.325509	0.010020	0.313070	0.669291	0.461386	0.389540	0.305411	0.053161	B
1	Bikeway Fire	2019-07-05	37.469000	-121.369700	0.0	0.129890	0.015030	0.447918	0.576772	0.726073	0.418023	0.293432	0.283385	B
2	Snowstorm Fire	2019-07-05	40.679318	-120.392768	0.0	0.129890	0.026052	0.338793	0.529528	0.928713	0.311154	0.576926	0.060655	C
3	Hollow Fire	2019-07-03	37.632060	-121.538382	0.0	0.308294	0.038076	0.635021	0.586614	0.729373	0.147918	0.092538	0.086901	C
4	Merced Fire	2019-06-29	37.312877	-120.242411	0.0	0.738654	0.000000	0.442842	0.580709	0.664686	0.209311	0.054987	0.015547	C

Note. Post dimensionality reduction and normalization of the dataset.

3.5 Data Preparation

Data Preparation must be done before the model evaluation phase of the project. The aggregated dataset will be split in the ratio of 60:40 for training and testing sets respectively. In order to achieve this, the stratified train test split function in Python is used to keep the exact distribution of the severity score. The k-fold cross validation procedure will be done on the training dataset (60%) for validation/hyperparameter tuning purposes.

The training set is used to teach the model. The model iteratively adjusts its parameters based on the training data. The training set prevents information leakage from the test set, which will inappropriately influence the model's training. Stratified k-fold cross validation and grid search is used to tune the model's hyperparameters. The test set is used to evaluate the chosen model after the grid search and k-fold cross validation. The test is used to evaluate the final

performance of the model and gives an unbiased determination of how well the model performs on unseen data.

Figures 29 and 30 show the sample training and testing datasets. The training dataset contains 438 records, and the testing dataset contains 293 records. The split datasets will then be ready for model evaluation purposes.

Figure 29

Training Dataset

	Name	Date	Latitude	Longitude	Precipitation	Wind Gust	Cloud Cover	Relative Humidity	Solar Energy	PDSI	NDVI	Elevation	Slope	Severity_Class
51	Patterson Fire	2019-08-15	38.652475	-120.826165	0.0	0.147105	0.000000	0.213058	0.448819	0.728053	0.689445	0.147292	0.033567	B
372	Diamond Fire	2020-10-13	37.983370	-120.645410	0.0	0.255086	0.032064	0.411120	0.232283	0.475248	0.338200	0.131844	0.069506	B
129	Dales Fire	2019-10-07	40.314187	-122.070055	0.0	0.051643	0.004008	0.328758	0.271654	0.651485	0.520225	0.088713	0.015483	B
470	Dolicini Fire	2021-07-14	38.097507	-122.741833	0.0	0.270736	0.460922	0.782905	0.533465	0.168317	0.326711	0.052902	0.019269	B
680	Gold Fire	2020-07-20	41.110370	-120.923293	0.0	0.107981	0.000000	0.263468	0.547244	0.413861	0.249880	0.501316	0.012030	F

Note. Dataset for training.

Figure 30

Testing Dataset.

	Name	Date	Latitude	Longitude	Precipitation	Wind Gust	Cloud Cover	Relative Humidity	Solar Energy	PDSI	NDVI	Elevation	Slope	Severity_Class
18	Milton Fire	2019-07-15	37.944751	-120.843720	0.000000	0.450704	0.000000	0.350329	0.511811	0.726733	0.233365	0.049801	0.048762	C
400	San Dimas Fire	2020-11-06	34.146195	-117.778561	0.003922	0.328638	0.500000	0.488868	0.220472	0.494389	0.393849	0.186969	0.735398	C
355	Johnson Fire	2020-08-28	34.659790	-118.364110	0.000000	0.328638	0.000000	0.195294	0.500000	0.676568	0.262326	0.400299	0.019827	C
171	Holy Fire	2018-08-06	33.698880	-117.520550	0.000000	0.289515	0.163327	0.450571	0.437008	0.348515	0.637626	0.346252	0.754294	F
713	SCU Lightning Complex	2020-08-16	37.439437	-121.304350	0.011765	0.467919	0.156313	0.317799	0.474409	0.409901	0.271900	0.097982	0.126295	F

Note. Dataset for testing.

3.6 Data Statistics

Table 7 shows the multiple steps taken and the changes in Data Dimension during the Data Collection, Preprocessing, Transformation and Preparation phases. The initial data is loaded and filtered to 756 records. Wildfire factor data such as weather, vegetation index, drought index and elevation and slope will be retrieved using the latitude, longitude and date. The collected data are merged and cleaned by dealing with irrelevant features and missing or null values. A severity metric is created to rank the severity of the wildfire and dimensionality reduction is done based on variance inflation factor (VIF). Finally, the data is split using a stratified split function in 60:40 ratio for training and testing purposes.

Table 8

Dataset Sizes at Different Stages

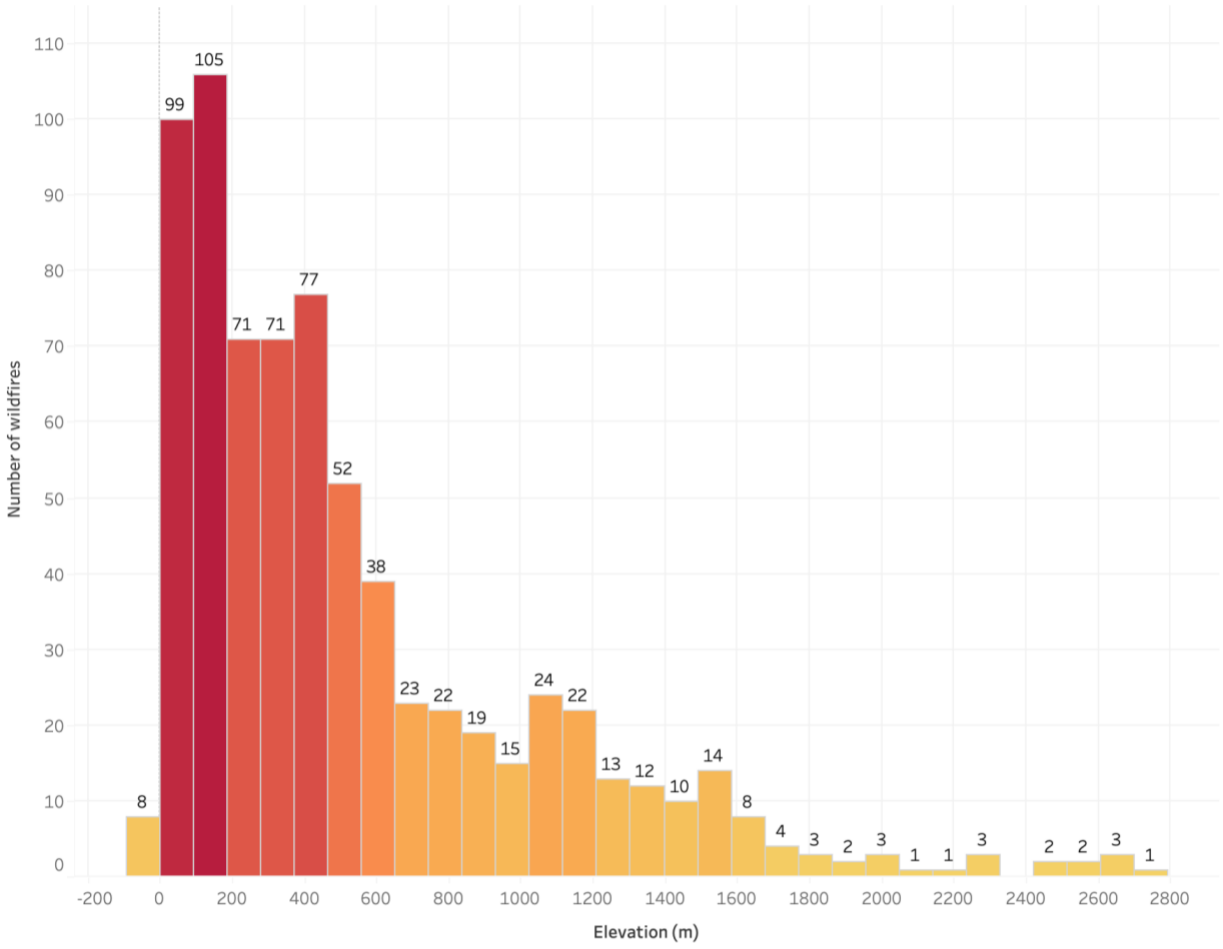
Stage	Dataset	Dimension
Raw	Cal Fire Incident Dataset	2062 x 23
	Cal Fire Incident (Filtered)	756 x 23
	Weather Data	756 x 26
	Vegetation Index	756 x 4
	Drought Index	756 x 4
	Elevation and Slope	756 x 4
Pre-processing	Merged Data (Cal Fire, Weather, Vegetation, Drought, Elevation and Slope)	756 x 50
	Clean Data (Irrelevant Column Removal, Dropped Duplicates, Removal of Records with Missing Values)	731 x 21
Transformation	Create Severity Metric	731 x 22
	Removing Variables with a high VIF	731 x 14
Data Preparation	Training	438 x 14
	Testing	293 x 14

Note. Dataset sizes at different stages.

After the Data Preparation phase, statistical analysis is done on the dataset to check the distribution of the dataset as well as ensure the quality of data used will be suitable moving forward. Insights into numerous features of the data are shown in the figures below:

Figure 31

Distribution of the "Elevation" feature pre-normalization

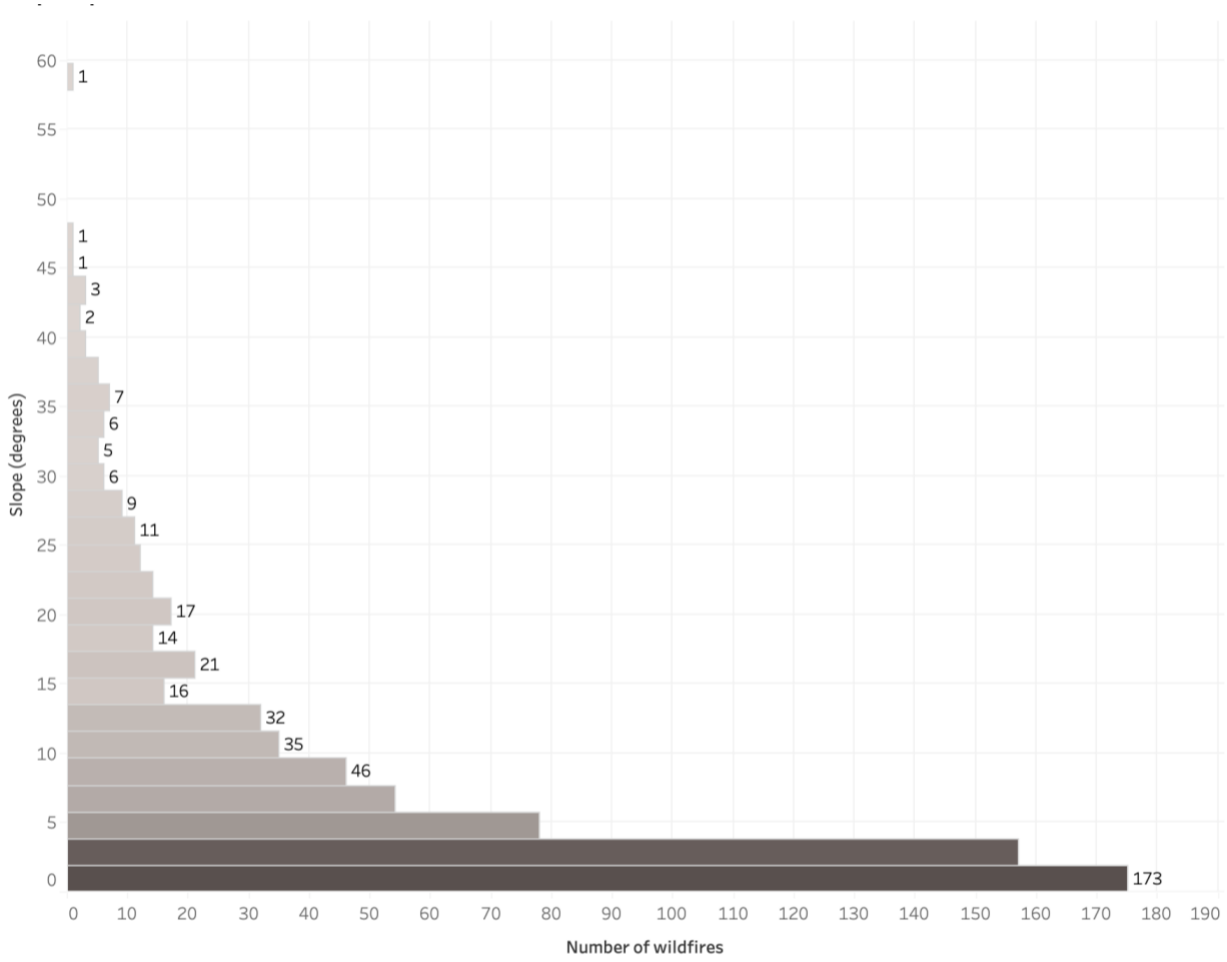


Note. Elevation distribution.

From the above figure, the distribution of elevation is right-skewed. The vast majority of wildfires occurred at an elevation of less than 1000 meters.

Figure 32

Distribution of the "Slope" feature pre-normalization

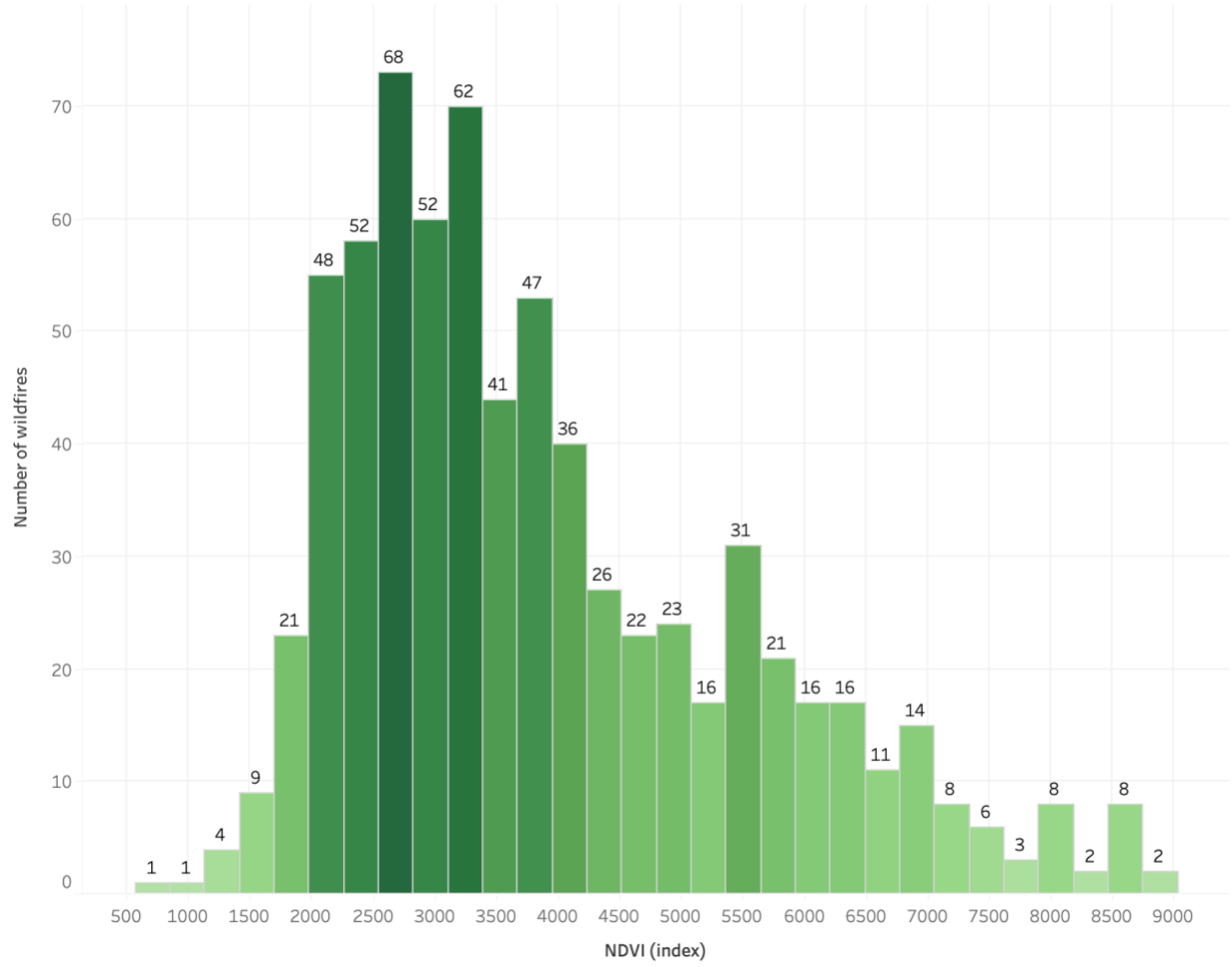


Note. Slope distribution.

From the above figure, the vast majority of wildfires occurred at terrain with slopes of less than 10 degrees.

Figure 33

Vegetation Index (NDVI) Data Distribution

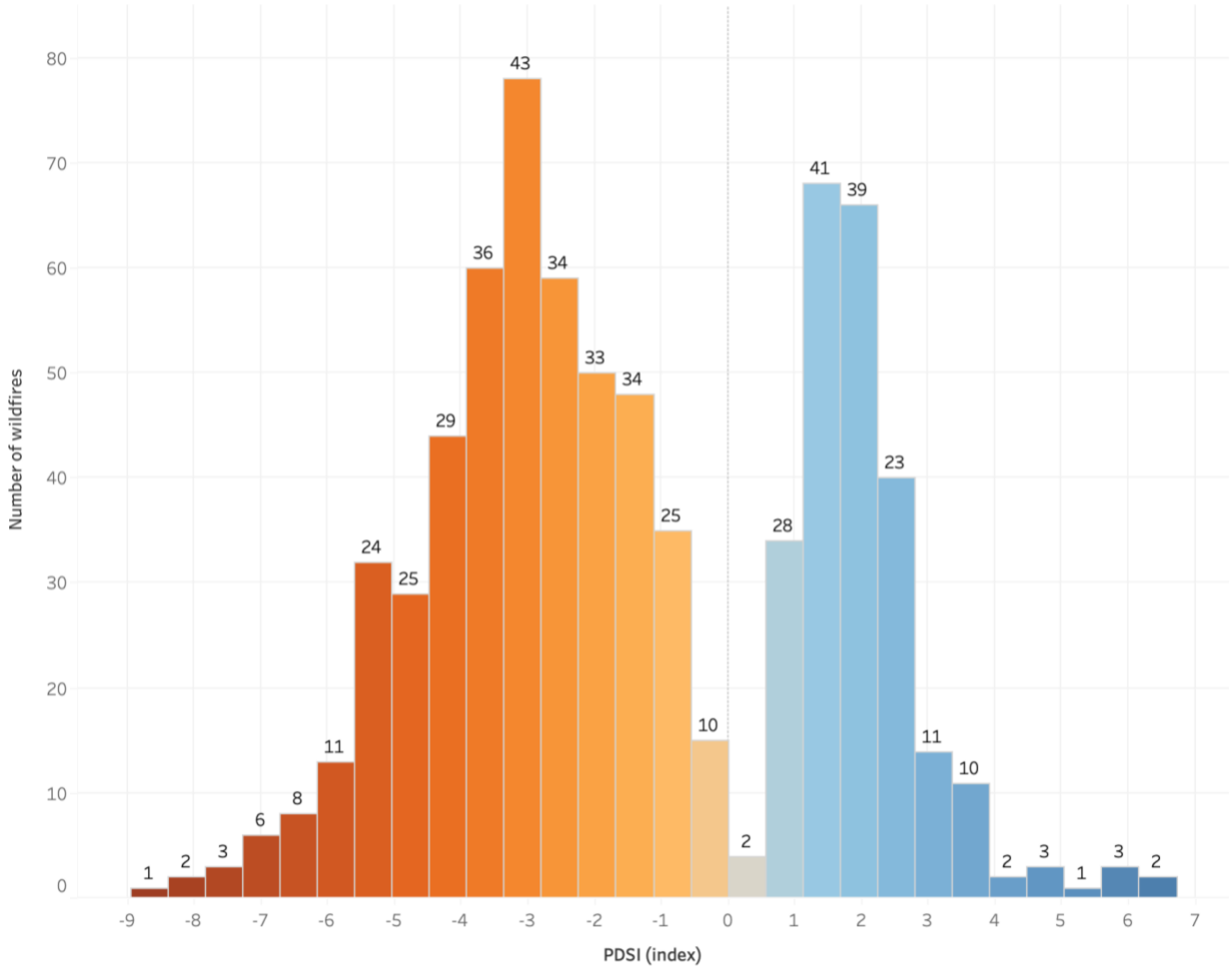


Note. Vegetation index (NDVI) distribution.

From the above figure, the distribution of NDVI is slightly right skewed. The majority of wildfires occurred with an NDVI of between 2000 and 4000.

Figure 34

Palmer Drought Severity Index (PDSI)

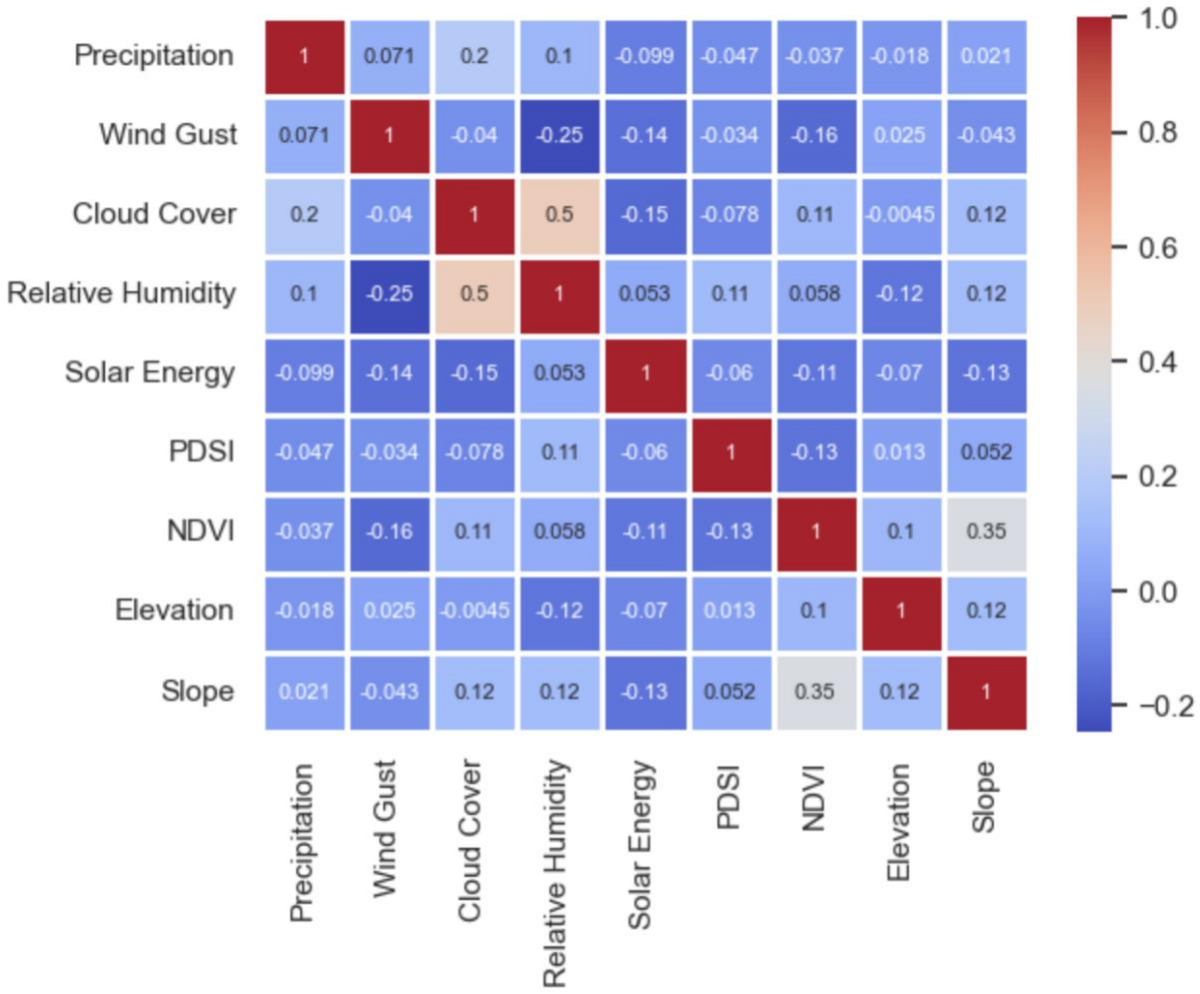


Note. Palmer drought severity index (PDSI) distribution.

From the above figure, the distribution of PDSI is bimodal. It appears that a greater number of wildfires occurred on land with a negative PDSI, indicating more dry conditions.

Figure 35

Dataset Correlation Matrix of the Predictor Variables

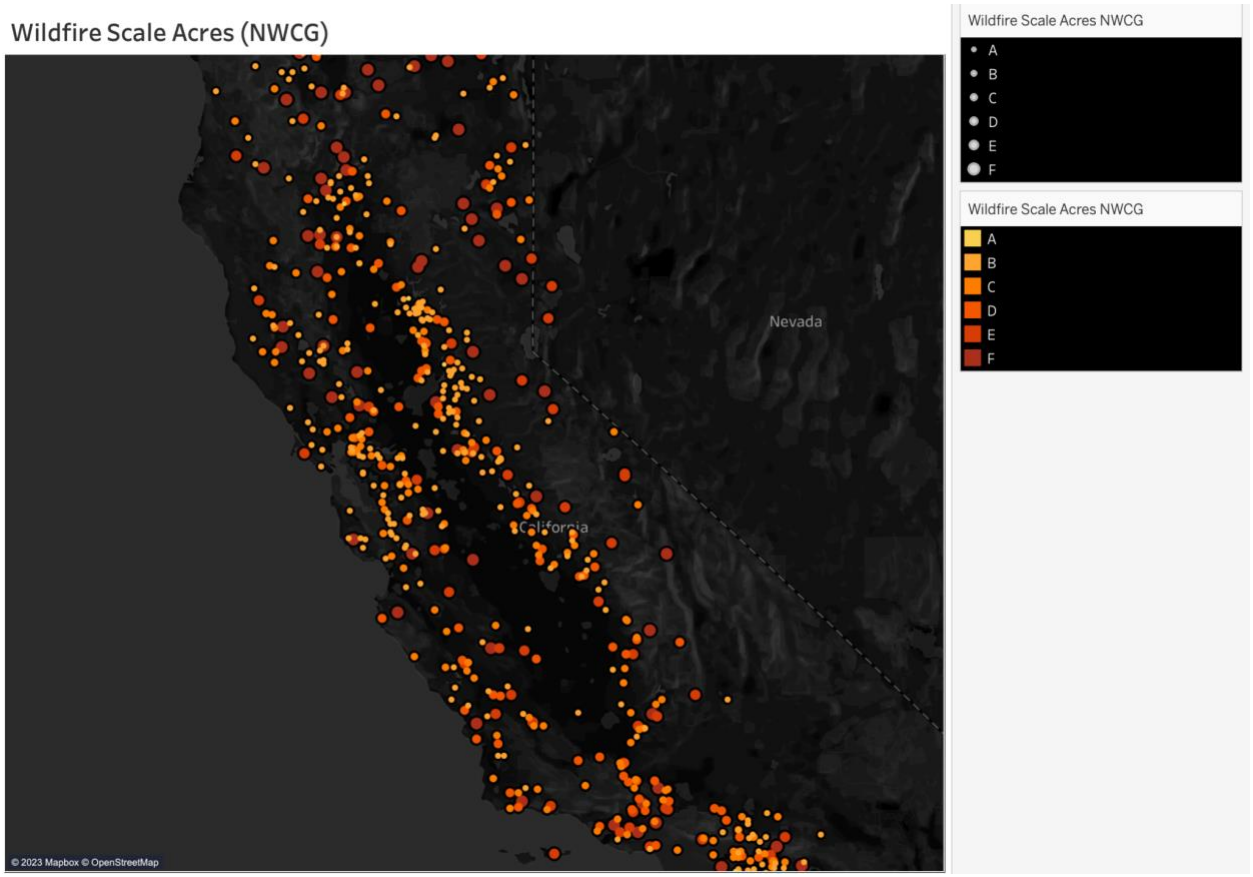


Note. Correlation matrix.

From the above figure, no two pairs of predictor variables have a high degree of correlation. This is a representation of how removing factors with a high VIF gets rid of the multicollinearity issue and ensures that the predictor variables are not highly correlated.

Figure 36

Geographical Representation of Wildfire Incidents and Severity



With the aid of the above visualizations, such as through the distributions of various variables and the correlation matrix, this further confirms that there are no outliers present in the data and that multicollinearity will not affect the performance of the machine learning models. The training, validation, and test datasets are ready for machine learning modeling, which will be presented in the following section.

4 Model Development

4.1 Model Proposals

After data cleaning and transformation, the inputs for every model in this task are the features related to the wildfire severity, including Precipitation, Wind Gust, Cloud Cover, Relative Humidity, Solar Energy, PDSI, NDVI, Elevation, and Slope. The output is a predicted severity class, ranging from A (most severe) to F (least severe). Thus, machine learning models that are suited for multi-class classification tasks will be evaluated. There are a number of applicable machine learning models, ranging from standard machine learning models such as k-nearest neighbors to deep learning models such as artificial neural networks. In this chapter, six models will be evaluated: k-NN, Support Vector Machine (SVM) with a One-vs-One scheme, Random Forest (RF), XGBoost, AdaBoost and Artificial Neural Network (ANN). The models work as follows:

K-NN

K-NN, or k-Nearest Neighbor algorithm, is a similarity-based learning approach that works on the nearest neighbor principle. The nearest neighbor principle, in essence, works by finding the nearest neighbors (training samples with the shortest distance) to the new data point in the feature space, a classification prediction regarding the new data point can be made. The nearest neighbor principle relies on a distance metric as it needs to look for its nearest neighbors, the distance metric commonly used is the Minkowski distance shown in formula (1). Different values of p will result in different types of distances used. Special cases of the Minkowski distance occur when p = 1, which will result in using the Manhattan distance and p = 2 which is the Euclidean distance.

$$\mathrm{dist}(\mathbf{x}, \mathbf{z}) = \left(\sum_{r=1}^{d} |x_r - z_r|^p \right)^{1/p} \tag{1}$$

With the basis of the nearest neighbor principle explained, the general K-NN algorithm can then be elaborated. First, ‘K’ which refers to the number of nearest neighbors is predefined. With the training dataset is initially stored, for each new unlabeled data, the Euclidean distance is calculated with all training data points using (1) where p= 2 (Euclidean distance). Find the k-nearest neighbors and assign the unlabeled data with the class of the maximum number of nearest neighbors (Taunk et. al, 2019). For the K-NN model, ‘k’ is a very important hyperparameter to set when initializing the model. When the value of k is very low or k = 1 this would result in a very low error rate as the model will look for the closest neighbor. A low error rate may sound good, however at k = 1 the boundaries are overfitted. When the k value is too large, the model will look for significantly more neighbors around the unlabeled data. Underfitting will happen as the model will select the majority class of the k neighbors. Using a large k value has risk particularly when dealing with an imbalanced dataset (Kelleher, 2015). Since an imbalanced dataset contains significantly more instances of one class over the other, the model will then classify the unlabeled data as the majority class. This is because the majority class has dominated the feature space. To counter this problem, another version of the K-NN model known as the Weighted K-NN is used. The Weighted K-NN fundamentally uses the same principles, however it considers the weight of each neighbor. The weight for each neighbor uses equation (2). This means that neighbors closer to the unlabeled dataset will have a higher weight compared to further neighbors. As distance d increases, weight approaches zero and vice versa.

$$\frac{1}{dist(\mathbf{q}, \mathbf{d})^2} \tag{2}$$

Support Vector Machine, One-vs-One scheme

Support Vector Machine (SVM) is a supervised learning algorithm that may be applied to classification tasks. SVM works by discovering the optimal decision boundaries, or hyperplanes, that separate data records that belong to different classes of the target feature. The hyperplane operates in an n-1 dimensional space (with n corresponding to the number of features). For instance, in a two-dimensional space, the hyperplane is a line, and in a three-dimensional space, the hyperplane is a plane. The support vectors are the data points that are closest to the hyperplane, and thus influence the position and orientation of the hyperplane. The margin is the distance between the hyperplane and the nearest support vectors from the different classes of the target features. The ideal hyperplane maximizes the margin between classes. Once the hyperplane is determined, a data point is classified into a class based on which side the data point lies. Before training the SVM model, it is important to perform preprocessing steps such as handling missing values and scaling the features (through techniques such as min-max scaling or standardization). Although the base version of SVM is suited for binary classification tasks, the one-vs-one (OvO) scheme is a method to extend the typical binary SVM to handle multi-class classification tasks. In the one-vs-one approach, a SVM is trained for every pair of classes. The number of SVM classifiers that are trained for n classes is shown in (3).

$$\text{\# of Classifiers} = \frac{n*(n-1)}{2} \tag{3}$$

Each binary classifier will predict one class label. The final classification of the data point is obtained using a majority vote scheme. By default, SVM works in linear spaces. However, non-linear spaces can be supported with the use of kernel functions. The idea of kernel functions is to transform the input data into a higher dimensional space in order for the classes to be better separated. An example of a kernel function is a radial basis function, or “rbf”. The hyperparameters involved in SVMs include the kernel function, a kernel coefficient, and a

penalty parameter. According to Harrison (2019), the smaller the penalty parameter, also referred to as ‘C’, the tighter the decision boundary, and thus more overfitting will occur (p. 137). A lower kernel coefficient, also referred to as ‘gamma’, leads to overfitting the training data (p. 138). At its core, the Support Vector Machine algorithm is based on the optimization of a convex quadratic programming problem. The SVM algorithm makes predictions by solving the quadratic programming problem based on certain constraints, which will depend on factors such as which kernel function is used.

Random Forest

The Random Forest is an ensemble collection of multiple decision trees that are constructed from bootstrap samples. First, the principle of decision trees will be explained. Decision tree is a type of supervised learning algorithm which can be used for classification and regression tasks. It recursively splits the data into subsets based on the input data. The decision at each node on which feature to split the data will depend on the information gained from the feature. By splitting nodes with the largest information gain, the tree will be able to minimize the impurity of the data at each node. Information gain can be calculated using Entropy as well as Gini Index. “Claude Shannon’s entropy model defines a computational measure of the impurity of the elements in a set” as explained by Kelleher (2015).

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l))) \tag{4}$$

The Shannon’s Entropy model is shown in equation (4) where levels(t) is the set of levels in the domain of the target feature t, and P(t = l) is the probability of a randomly selected instance having the target feature level l.

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \times \underbrace{H(t, \mathcal{D}_{d=l})}_{\substack{\text{entropy of} \\ \text{partition } \mathcal{D}_{d=l}}} \tag{5}$$

The remainder entropy is calculated by weighting each partition *D* multiplied by the entropy of each partition *D*. The information gain calculated using entropy is shown at equation (6).

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - rem(d, \mathcal{D}) \tag{6}$$

Information gain can also be calculated by using the Gini index given by formula (7) which is a measure of impurity.

$$Gini(t, \mathcal{D}) = 1 - \sum_{l \in levels(t)} P(t = l)^2 \tag{7}$$

Where *D* is a dataset with a target feature t; levels(t) is the set of levels in the domain of the target feature; and P(t = l) is the probability of an instance of *D* having the target level l. The information gain can also be calculated by using the formula (8)

$$IG\ (d,\ D) = Gini\ (d) - Gini\ (d,\ D) \tag{8}$$

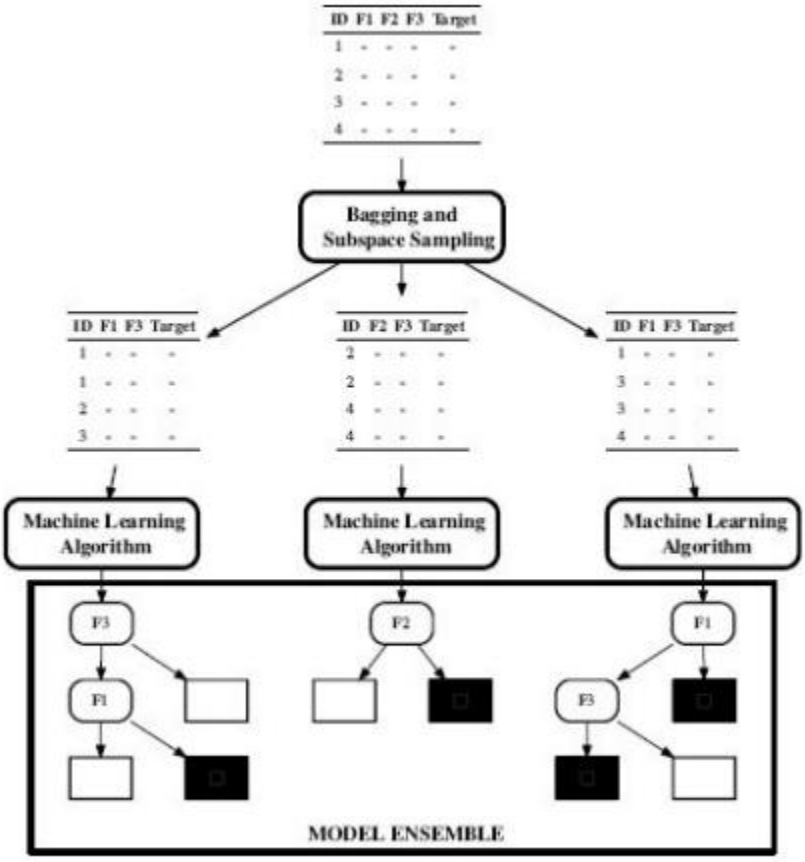
Here the Gini index for the whole dataset d is subtracted by the Gini index of the sum of the weighted Gini index scores for the partitions created by splitting with the feature *D*. Kelleher (2015) explained that the Iterative Dichotomizer 3 (ID3) algorithm uses the information gained to choose the best descriptive feature, which then creates a root node with the selected feature and partitions the data. A branch is grown from the node and process is repeated until all instances in a partition have the same target level, this would be a leaf node as it has reached the end. The concept for Random Forest is shown in Figure 37 (Kelleher, 2015). Kelleher (2015) explained that by using the bagging ensemble method, each model is trained on a random sample which is the same size as the dataset and sampling with replacement is used. Subspace sampling

is when a subset of descriptive features is randomly selected for sampling. ‘The combination of bagging, subspace sampling and decision trees is known as a random forest model’ explained Kelleher (2015). As the random forest model utilizes multiple trees, the predictions from each tree are collected and the mode of the classification will be calculated and projected as the final prediction of the random forest model.

Figure 37

Process of creating model ensemble using bagging and subspace sampling

(Kelleher, 2015). Image reproduced from a book source.



XGBoost

XGBoost, or extreme gradient boosting, is an ensemble method which is a more efficient and scalable implementation of the gradient boosting algorithm. XGBoost can perform regression and classification tasks and has become popular due to its speed and performance. Brownlee (2016) stated that XGBoost had been "dominating applied machine learning and Kaggle competitions for structured or tabular data". XGBoost works by iteratively combining weak classifiers (which are usually decision trees), to create a stronger classifier that can make more accurate predictions. Initially, a weak model is created (typically a shallow decision tree) to make the preliminary predictions. New decision trees are added in a sequential manner, with each new model attempting to correct the errors made by the previous model. The structure of the tree is determined through minimizing the objective function, which consists of the loss function and the regularization term. The loss function measures how well the model fits on the training data. The regularization term creates a penalty for the complexity of the model (such as the number of leaves in the trees) and helps to prevent overfitting. Assuming there are k trees, the formula of the objective function (consisting of the loss function and regularization term) is shown in (9).

$$Objective = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \quad (9)$$

The XGBoost algorithm may end through conditions such as specifying a maximum number of iterations, or through early stopping, which will end the algorithm if the performance of the model does not improve for a pre-specified number of consecutive iterations. In order to make a final prediction, a weighted sum of the individual models' predictions is calculated. Before training an XGBoost model, it is not too essential to perform preprocessing steps such as scaling the features and imputing missing values since XGBoost can handle missing values and

learn the best ways of imputing the missing values. There are a number of hyperparameters involved in XGBoost. These include the learning rate (also known as ‘eta’, which helps to control overfitting), the number of boosting rounds (number of iterations until the algorithm ends), and the maximum tree depth (a higher maximum tree depth will make the model more complex which may lead to overfitting). Tuning the hyperparameters in XGBoost is important to achieve better performance. XGBoost can also provide an importance score for each feature, which helps with the interpretability of XGBoost compared to other machine learning models.

AdaBoost

AdaBoost (Freund, Y., & Schapire, R. E.., 1997), also known as Adaptive Boosting, is a highly effective machine learning algorithm that is applicable to both classification and regression problems. The model architecture of AdaBoost is an ensemble of weak learners, where each weak learner is a simple decision tree with a single node. The algorithm operates iteratively, adding new models to the ensemble with more emphasis on the instances that were misclassified by the previous models. Weak learners are trained on distinct subsets of the data. The resulting model represents a weighted sum of the weak learners, where each learner's weight is proportional to its accuracy. The final model is then used to make predictions on new data. As a result, AdaBoost is a powerful tool for improving the accuracy and reliability of machine learning models. The form of the boosted classifier is shown in (10):

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \tag{10}$$

Each function is a weak learner that takes x as input and returns the class. As an output, each weak learner produces hypothesis h for prediction $h(x_i)$ for each instance in the training set. With each iteration t, a weak learner is picked and assigned a coefficient α_t, thus, resulting in the total error E_t of the boosted classifier being minimized shown in (11):

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)] \quad (11)$$

Where F_{t-1} is the boosted classifier, which results from the previous iteration of training and $f_t(x) = \alpha_t h(x)$, the weak learner that is picked as an addition to the resulting classifier. The algorithm for the AdaBoost is shown in Table 9.

Table 9

AdaBoost algorithm

No.	AdaBoost Step
1	Set the weights of the training data to be uniform.
2	For each iteration *t = 1,...,T:*
2.1	Fit a weak classifier *h_t(x)* to the training data using the current weights.
2.2	Calculate the weighted error of the classifier as: *e_t = sum(w_i * I(y_i = h_t(x_i)))/sum(w_i),* where *w_i* is the weight of *i* training example, y_i is the true label of *i* training example, and *I(...)* is the indicator function.
2.3	Calculate the weight of the classifier as: *alpha_t = log((1 - e_t)/e_t)*
2.4	Update the weights of the training data as: *w_i = w_i * exp(alpha_t * I(y_i = h_t(x_i))),* where *I(...)* is again the indicator function.
3	Output the final classifier as: *H(x) = sign(sum(alpha_t * h_t(x)))*
4	The basic equation for the final prediction of AdaBoost *H(x)* is a weighted sum of the individual predictions from each weak classifier *h_t(x)*, where the weight of each classifier *alpha_t* is proportional to its performance on the training data. The sign function at the end simply converts the weighted sum into a binary prediction: *H(x) = sign(sum(alpha_t * h_t(x)))*

In the context of the wildfire severity classification task, AdaBoost was chosen because it is a powerful algorithm for handling imbalanced datasets, where the number of instances in each class is not equal. This is the case with the wildfire severity dataset, where the majority of instances are in the classes “B” and “C”. AdaBoost is also known for its ability to handle noisy

data and avoid overfitting, making it a good choice for a task where the dataset may have high variance or noise. The algorithm's ensemble approach and use of decision trees as weak learners also make it well-suited for the task of classifying severity levels based on a set of input features.

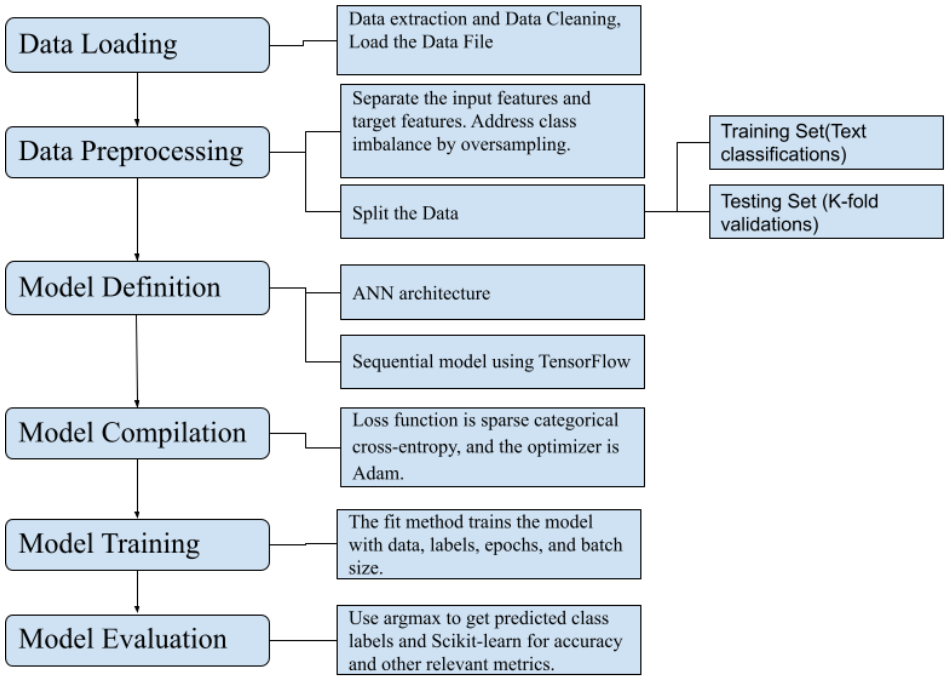
AdaBoost utilizes the SAMME (Stagewise Additive Modeling using a Multiclass Exponential loss function) algorithm, which builds upon the original binary AdaBoost algorithm. SAMME is specifically tailored for multiclass classification tasks that involve more than two classes (Hastie, T., Rosset, S., Zhu, J., & Zou, H.., 2009). To further enhance its capabilities, the SAMME.R (SAMME with Real) algorithm is a variation of SAMME that can accommodate continuous prediction values, in addition to discrete values. In general, these algorithms equip AdaBoost with the versatility and adaptability required to tackle a wide range of complex machine learning problems.

ANN

Artificial neural network (ANN) is one of the most dependable machine learning methods for classification and prediction tasks. Given the input variables, ANN determines the conditional probability of the class. An ANN, on the other hand, is a more adaptable and reliable model since it can manage more intricate and nonlinear interactions between the input and output variables. Furthermore, working with an ANN has the benefit of handling multi-class classification issues. For instance, predicting the severity of a wildfire may be a multi-class issue in this study since the severity of a wildfire might vary from low to high.

Figure 38

Model architecture



As a result, by studying the patterns and connections between the input variables (such as precipitation, wind gusts, cloud cover, relative humidity, solar energy, drought, vegetation, elevation, slope, etc.) and the output variable (severity), an ANN is trained to predict the severity of a wildfire. An artificial neural network (ANN) works similarly to the human brain, consisting of a single input layer, one or more hidden layers, and an output layer. The number of input characteristics (68 in this case) determines the number of neurons in the input layer, while the number of neurons in the hidden layers is chosen using heuristics. The Python keras package implemented the ANN with the number of layers and neurons specified using the Dense constructor. In addition, the 'adam' optimizer and 'Categorical CrossEntropy loss function were utilized to minimize errors.

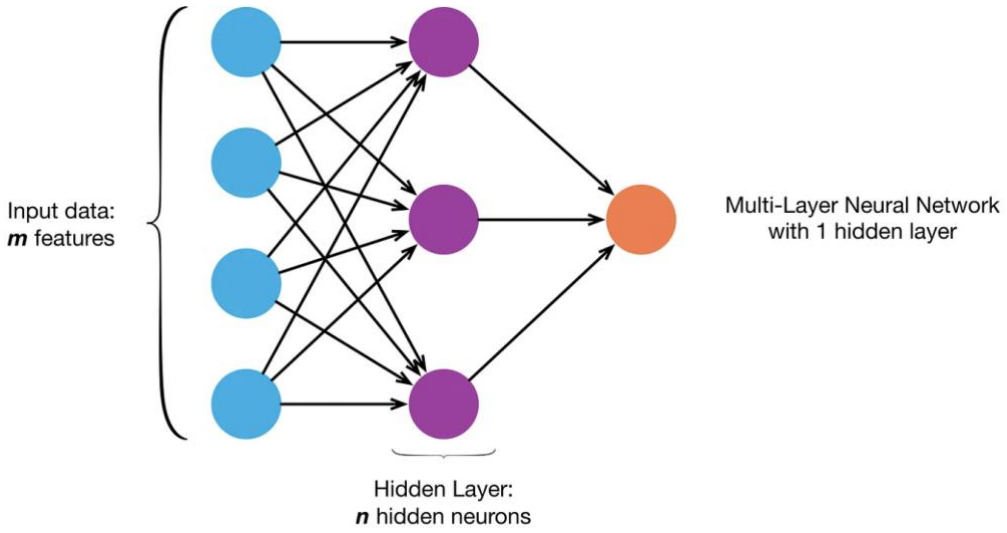
The model would take the input variables and send them through several layers of nodes (neurons) that execute nonlinear changes on the input data to train an ANN for wildfire severity prediction. The model discovers patterns and correlations in the data, which provide a

complicated decision boundary between the input and output variables. Each node in the ANN's output layer would stand for a severity class (low, medium, or high). To reduce the error between the anticipated output and the actual output, the model then employs backpropagation to modify the weights of the network's nodes. Adam is an optimization technique used to update the neural network's weights during training using backpropagation; thus, Adam is the optimizer as the model is being put together. To minimize the loss, the optimizer computes the gradients of the loss function explicitly according to the weights and biases in the network via backpropagation.

Figure 39

ANN hidden layers backpropagation method

(Nahua Khang.,2017). Image reference from a publication..



Let y be the dependent variable; x1, x2, x3, to xn are independent variables with the respective coefficients being ω1,ω2,ω3, to ωn, while b is the intercept (S. D. Ali *et al.*,2022).

$$y=\omega 1x1+\omega 2x2+\omega 3x3+\cdots+\omega nxn+b \quad (12)$$

The Artificial Neural Network (ANN) regression model consists of hidden layers W1 to Wn, each with n nodes, biases Bl, B2, B3, and an activation function α. For an n hidden layer ANN, the formulation would be as follows:

$$y = WTn\alpha(WTn\text{-}1\alpha(....(WT2\alpha(WT1\alpha(\omega Tx+b)+B1)+B2)....+Bn\text{-}1)+Bn) \quad (13)$$

The activation function α used here is the Rectified Linear Unit (ReLU), expressed as follows:

$$\alpha=\{x,x>00,\ x\leq 0(8) \quad (14)$$

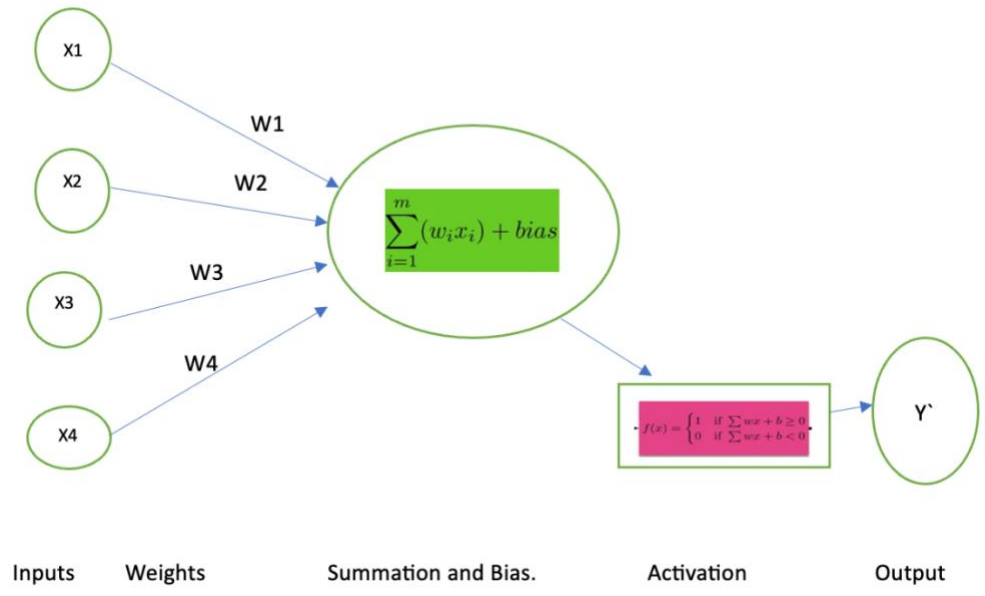
The activation function softmax function for multi-class classification is expressed as

$$\sigma(\mathbf{z})_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}} \text{ or } \sigma(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}} \text{ for } i = 1, \ldots, K. \quad (15)$$

The letter "Z" stands for the outcomes produced by the neurons. The exponential factor acts as a vital non-linear component that amplifies the results. The probabilities are created by normalizing the results through division by the sum of exponential values.

Figure 40

ANN model



Artificial Neural Network (ANN) is used for multi-class classification to predict the severity of wildfires. Encoded the categorical target variable into integers with the LabelEncoder from sci-kit-learn. The ANN architecture includes four dense layers with ReLU activation and dropout regularization to prevent overfitting. The output layer uses softmax activation to distribute probability over the six classes.

ANN model uses the sparse categorical cross-entropy loss function and the Adam optimizer to classify data. The suitable technique trained it using a specific batch size and number of epochs. A classification report, accuracy score, and ROC-AUC score were used on a test set to evaluate its performance. Each class's ROC curves and AUCs were calculated and displayed using the matplotlib package. Additionally, a confusion matrix was generated using the Seaborn library to demonstrate the model's ability to accurately identify examples for each class. The model is designed to correctly categorize instances based on the training data's attributes.

Figure 41

The ANN model algorithm will repeat until the severity of the classification is predicted.

```
Step 1:  Import libraries like Numpy and Pandas to prepare for multiclassification ANN Model.
Step 2:  To build the architecture of the neural network model, you can use the Keras Sequential API and call it
         "create_model."
def create_model():
    model = Sequential()
    model.add(Dense(32, input_dim=X_train.shape, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(y_train_cat.shape, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    return model

Step 3:  After loading the dataset from a CSV file, use pandas to separate the predictor variables (X) from the target
         (y).
Step 4:  To split the data into training and testing sets, utilize the train_test_split function from the sklearn
         library.
Step 5:  Convert non-numeric values in the target variable;  start by using LabelEncoder.
Step 6:  To convert the target variable's values into categorical information, use the
         "to_categorical" function.
Step 7:  To set tuning-related hyperparameters, create a dictionary called "param_grid."
param_grid = {
    'epochs': [20, 50],
    'batch_size': [16, 32],
    }
Step 8:  To create a neural network model in Keras, first, build a wrapper. Then, for cross-validation, utilize
         StratifiedKFold.
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

Step 9:  To find the best hyperparameters for your neural network model, utilize Sklearn's GridSearchCV for grid
         searching.
Step10:  Print both the optimal settings and the corresponding optimal result.
Step11:  To evaluate the test set, utilize the finest model and compute metrics such as MCC, AUC-ROC, accuracy, and
         classification report.
Step12:  Print to generate a categorization report.
```

4.2 Model Supports

The machine learning models are developed on local machines. The models are developed with the use of Python, Jupyter Notebook, and various libraries. The machine learning environment is configured with the Anaconda Python distribution, which simplifies package management and comes with many data science packages pre-installed.

Python is a widely used programming language in the data science community due to relative ease of use compared to other programming languages, and a large number of libraries which facilitate machine learning model development. Jupyter notebook is a web-based IDE which allows for executing Python code in different cells, visualizing the results, and includes markdown to document the coding process. The Pandas library is used to load and preprocess the data. Matplotlib and Seaborn are used to create compelling visualizations. Scikit-learn is the primary machine learning library in Python and allows for model training and evaluation. Scikit-learn includes methods to split the data into training and testing sets by specifying the test size, random state to ensure the same split occurs every time and includes a stratify parameter to ensure the distribution of a certain feature is maintained after splitting the data. It also includes functions to set up a grid search to evaluate different combinations of hyperparameters, and a function to set up cross validation by specifying the number of folds. Metrics such as accuracy and AUC-ROC can also be determined through scikit-learn. The scikit-learn library also includes the SVM classifier (called SVC), which includes a parameter to set up a Support Vector Machine model with a one-vs-one implementation to handle multi-class classification. The XGBoost library includes the classifier to train the XGBoost model.

Table 10

Python Libraries Used

	Library	Method	Usage
Scikit-Learn	Sklearn.ensemble	RandomForest Classifier	Implement RF Classifier
		AdaBoost Classifier	Implement AdaBoost Classifier
	Sklearn.svm	Support Vector Classifier	Implement SVM Classifier
	Sklearn.neighbors	KNeighborsClassifier	Implement K-nearest neighbor classifier
	Sklearn.model_selection	Train_test_split	Split data into training and testing sets
		StratifiedKFold	Cross validation for modeling
		GridSearchCV	Grid search for optimal parameter search.
	Sklearn.metrics	Accuracy_score	Model Evaluation
		Classification_report	
		Roc_auc_score	
		Matthews_corrcoef	
		Make_scorer	
		precision_recall_fscore_support	
		confusion_matrix	
	sklearn.preprocessing	Label Encoder	Encode target labels with values between 0 and n_classes-1.
XGBoost		XGB Classifier	Implement XG Boost Classifier
Tensorflow	keras	Sequential	A tf.keras.Model is created by sequentially grouping a linear stack of layers.
	keras.layers	Dense	Tightly connected layer in a neural network.
		Dropout	Applies dropout to the input.
Pandas	DataFrame	Head, Shape, Drop	Dataframe manipulation
Numpy	Random	Seed	Ensure same data used for all models
Matplotlib	matplotlib.pylot	Plot	Plot the values of y and x as lines, markers, and/or text labels.
Seaborn	Seaborn	Plot	Plot visualizations of correlations between different dataset variables

The machine learning data flow consists of the following steps. The cleaned and transformed dataset from the previous chapter is loaded with Pandas. The predictor variables and

the target variable are then separated into two different data frames. Training and test sets are created with a 60-40 split using train_test_split and are stratified based on the target feature in order for the distribution of the target feature to be preserved after the split. When necessary, the class labels of the target feature are encoded using LabelEncoder. The base classifier (such as SVC from scikit-learn or XGBClassifier from the XGBoost library) is then initialized. A hyperparameter search grid, which is a dictionary comprising of a hyperparameter as the key and a list of different values of the hyperparameter to be evaluated as the value, is defined to be used later in the grid search for hyperparameter tuning. A grid search model using stratified k-fold cross validation is initialized (with GridSearchCV) using the base classifier, the parameter grid, a stratified k-folds cross validator (using StratifiedKFold from scikit-learn which preserves the distributions of classes) and specifying a custom scorer with the Matthews correlation coefficient (MCC) metric. An example setup for the grid search model in Python is shown in Figure 42, with cv = StratifiedKFold(n_splits = 5):

Figure 42

Example of setting up a Grid Search model in Python using the SVM classifier

```
grid_search = GridSearchCV(svc, param_grid, cv=cv, scoring=make_scorer(matthews_corrcoef), verbose=1, n_jobs=-1)
```

Afterwards, the grid search cross validation model is fit on the training data. The Matthews correlation coefficient metric is used since the primary purpose of our project is to give emergency personnel a certain degree of reliability as to the probability that a certain wildfire will be a Class A wildfire, a Class B wildfire, etc. Thus, the ability of the model to predict individual classes is important. The test set is imbalanced, and thus the Matthews correlation coefficient is an appropriate metric for this project (this metric will be discussed in

more detail in section 4.4). Stratified k-fold cross validation is used in this case due to an imbalance of classes in the dataset, an illustrative example is shown in Figure 43.

Figure 43

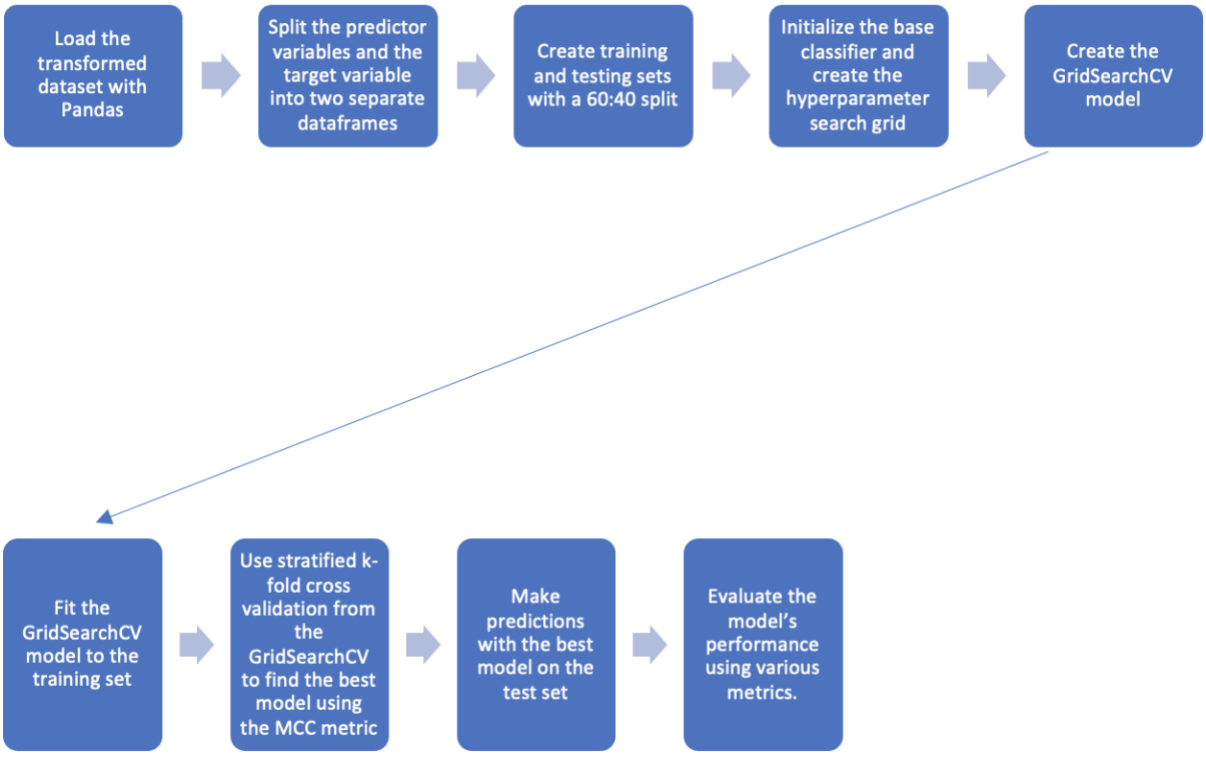
K-Fold Cross Validation with K=5.

Fold 1	Test	Train	Train	Train	Train
Fold 2	Train	Test	Train	Train	Train
Fold 3	Train	Train	Test	Train	Train
Fold 4	Train	Train	Train	Test	Train
Fold 5	Train	Train	Train	Train	Test

Stratified k-fold cross validation involves splitting the training data into k-equal folds, with each fold containing approximately the same distribution of classes as the original data. For each iteration, one-fold is used as the testing set and the other folds are used for training. During each iteration, the performance is evaluated on the test set (in this case five folds are used, and the Matthews correlation coefficient metric is used). This approach is preferred over the usual hold-out method where the data is trained and tested only once (Yadav, 2016). This allows the data to be trained and tested 'K' number of times. After all the iterations, the average MCC for all the iterations is calculated, which provides a better understanding of how well the model generalizes to new data. After the model is tuned using stratified k-fold cross validation on the training set, the best performing model is chosen and evaluated on the test/hold-out set created earlier using train_test_split. Metrics such as the MCC, macro precision/recall/F1-score, AUC-ROC score, and accuracy are then obtained. The machine learning modeling process is shown in Figure 44.

Figure 44

Machine Learning Modeling Process



4.3 Model Comparison and Justification

The chosen models of k-NN, Random Forest, Support Vector Machine with a One-vs-One scheme, XGBoost, AdaBoost, and ANN are suitable for multi-class classification tasks. The project aims to classify wildfires based on initial meteorological and environmental conditions into Class A through Class F, which is very similar to the wildfire severity metric used by the National Wildfire Coordinating Group, or NWCG. Since there are more than two levels to the target feature (six levels in total), the machine learning task is a multi-class classification task and thus the models being used are suitable for this project.

All of the models are supervised machine learning algorithms. Every model except for ANN is a standard machine learning model, while ANN is a deep learning model with a completely different approach to make predictions. In models such as SVM, k-NN, and ANN it

is essential to scale the features through approaches such as min-max scaling or standardization. By contrast in models such as XGBoost, Random Forest, and AdaBoost, it is not as important that the features are scaled (although it can be a better approach). Handling missing values is required for all the models except for XGBoost, which can automatically handle missing values and determine the best ways of imputing missing values. For every model, it may be optimal to perform dimensionality reduction to improve efficiency and reduce the chances of overfitting. The performance of the SVM, XGBoost, Random Forest, and ANN models can rely heavily on the choice of hyperparameters. For instance, for SVM, the performance can be heavily impacted on the choice of the kernel function that is used and for XGBoost, if the hyperparameters do not lend to proper regularization, the model may overfit the data. SVM and k-NN are computationally expensive for large datasets, while the other models are efficient for large datasets. For instance, Harrison (2019) states that the scikit-learn implementation of SVM is $O(n^4)$ (p.135). The interpretability of the SVM and ANN models may be very difficult/non-existent, and for SVM it may depend on factors such as the kernel function that is used. On the other hand, Random Forest, XGBoost, and AdaBoost provide scoring of each feature's importance, which allows for easier interpretation of the model's decisions. The following table, Table 11, summarizes the comparison between every model:

Table 11

Comparison of Every Model

Criteria	SVM	XGBoost	k-NN	Random Forest	AdaBoost	ANN
Basic Idea	Derive an ideal hyperplane to separate classes	Ensemble method that combines weak learners	Classify according to the k nearest neighbors	Ensemble of decision trees, using bagging method	Sequentially boosts weak learners	Simulates how a human brain works, three layers of nodes
Scaling of Data	Required (through min-max scaling, standardization, etc)	Not as essential	Required	Not as essential	Not as essential	Required
Missing Values	Required to handle missing data	Can automatically handle missing values	Required to handle missing data	Required to handle missing data	Required to handle missing data	Required to handle missing data
Hyperparameters	Relatively few (C, gamma, kernel, etc.)	Many (n_estimators, learning_rate, max_depth, early_stopping_rounds, etc.)	Few (k, distance metric, algorithm, weights)	Many (n_estimators, max_depth, max_leaf_nodes, etc.)	Few (estimator, n_estimators, learning_rate, algorithm)	Many (number of nodes, activation function, learning rate, batch size, etc.)
Ability to handle large data	Computationally expensive, $O(n^4)$	Designed for efficiency and scalability	Computationally expensive	Efficient for large datasets	Efficient for large datasets	Efficient for large datasets
Interpretability	Difficult/non-existent, varies for instance on kernel function	Easier interpretation, provides an importance score for each feature	Easy interpretation	Easier interpretation, provides an importance score for each feature	Easier interpretation, provides an importance score for each feature	Difficult/non-existent, neural networks are often referred to as a "black box"

Overall, all of the models are suitable for multi-class classification tasks, but their performance and applicability depend on factors such as the nature of the data and the objectives of the machine learning task. For instance, k-NN and Support Vector Machine with a one-vs-one scheme may work very well if the classes have a relatively clear margin of separation, and the dimensionality of the dataset is not very high. On the other hand, XGBoost, Random Forest, AdaBoost, and ANN are designed to work well with large data. If interpretability and having an important score for each feature is required for the given task, then Random Forest, XGBoost, and AdaBoost will be a better choice. As with any machine learning model, it is essential to experiment with all of the models by tuning hyperparameters and using cross-validation to evaluate the performance. The chosen model should be based on the performance and applicability to the data and to the specific problem.

4.4 Model Evaluation Methods

There are a number of different metrics that can be used to evaluate multi-class classification algorithms. Some of these, including the Matthews correlation coefficient, accuracy, macro precision/recall/F1-score, and AUC-ROC (adapted for multi-class classification) will be discussed.

After the models have been trained and tested on the test dataset for multiclass classification tasks, evaluation methods and metrics must be utilized to evaluate the performance of the model. The Matthews Correlation Coefficient is an evaluation metric used to measure both performances for binary and multiclass classification. For multiclass classification, the formula is shown in equation (8) where 't_k' is the number of times class k truly occurred, 'p_k' is the number of times class 'k' was predicted, 'c' is the total number of samples correctly predicted and 's' is the total number of samples. The MCC can be derived from a confusion matrix, which is a

square matrix where each row represents the true class label, and each column represents the predicted class label. Values of the MCC metric lie between -1 and 1. 1 refers to perfect classification prediction, 0 refers to random classification predictions and -1 refers to inverse classification prediction. According to Jurman et. al (2012), "MCC is a good compromise among discriminancy, consistency and coherent behaviors with varying number of classes, unbalanced datasets, and randomization." This works well as the current dataset is unbalanced with certain classes dominating the feature space more than others. The MCC will then be used as a main metric for GridSearch scoring.

$$MCC = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{(s^2 - \sum_k^K p_k^2) \times (s^2 - \sum_k^K t_k^2)}} \tag{15}$$

The accuracy metric is a measure of how well an algorithm can predict the correct class labels and is one of the most commonly used evaluation metrics for classification. Accuracy is defined as the ratio of the number of correct predictions to the total number of instances in the dataset. The formula for accuracy is shown in (16):

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Instances} \tag{16}$$

One way of calculating accuracy is by using a confusion matrix. The diagonal elements of the matrix correspond to the correct predictions. Thus, in order to calculate accuracy, one can take the sum of the diagonal elements and divide them by the total number of instances. Although accuracy is an easily interpretable metric, it may not be the best metric when dealing with heavily imbalanced datasets.

In order to calculate macro precision/recall/F1-score in a multi-class classification setting, the precision, recall, and F1-score has to be calculated for each individual class.

Precision is defined as the ratio of true positives to the sum of both the number of true positives and false positives. In a multi-class setting, calculating true positives/false positives for an individual class can be done by treating it as a binary classification problem (i.e., the class of interest against all the other classes). According to Grandini et. al (2020), "Precision tells us how much we can trust the model when it predicts an individual [class] as positive". The formula for precision is shown in (17):

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{17}$$

Recall is defined as the ratio of true positives to the sum of both the number of true positives and false negatives. According to Grandini et. al (2020), "The Recall measures the model's predictive accuracy for the positive class: intuitively, it measures the ability of the model to find all the Positive units in the dataset." The formula for recall is shown in (18):

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{18}$$

F1-score is a widely used evaluation metric that takes into account both precision and recall. It may be a particularly good metric when dealing with an imbalanced dataset. The F1-score is the harmonic mean of precision and recall, and is shown in (19):

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{19}$$

Macro precision/recall/F1-score involves taking the precision, recall, and F1-scores for each individual class and taking the average. The average is unweighted and considers classes of all sizes equally. It is also possible to calculate the weighted averages of the three metrics, which assigns weight based on the size of the individual class. In an imbalanced dataset, it may be a better option to consider the weighted averages. The closer these metrics are to 1, the better the model's ability in predicting the correct class for each individual class.

The AUC-ROC score, or the area under the receiver operating characteristic curve, is used to measure the performance of a binary classifier by plotting the true positive rate against the false positive rate at various thresholds. The true positive rate is also known as recall, and the false positive rate is defined as the ratio of false positives to the sum of both the number of false positives and true negatives. In a multi-class classification setting, the AUC-ROC score can be obtained by calculating the AUC-ROC score for each individual class using one-vs-one or one-vs-rest schemes, and then taking the average. A completely random algorithm would have an AUC-ROC score of 0.5, while a perfect algorithm would have a score of 1.

4.5 Model Validation and Evaluation

After the model has been trained and optimized through finding the best hyperparameter combinations resulting in the highest MCC score after the GridSearch procedure, the resulting model is evaluated on the test set.

K-NN Model Results

The base model for the K-NN model uses the KNeighborsClassifier from scikit-learn. The parameter grid prepared for the K-NN model is shown in Figure 45. The neighbors to be tested during the optimization will include neighbors 1-20 and different weights, whether the distance between points have any weight or not.

Figure 45

K-NN Hyperparameter Grid

```
n_neighbors = np.arange(1,20)
weights = ['uniform', 'distance']
algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute']
```

Using the GridsearchCV for parameter optimization with the scoring set to the Matthews Correlation Coefficient, the best model returned the following hyperparameter combinations

shown in Figure 46. Number of nearest neighbors at 18 and using the distance weight for each data point.

Figure 46

K-NN Hyperparameter combination and Results

```
Parameters: {'algorithm': 'auto', 'n_neighbors': 18, 'weights': 'distance'}
Training MCC: 0.11841616061120648
Test MCC: 0.08957992812457329
Testing Accuracy: 0.4246575342465753

AUC-ROC score: 0.5257895886128685
```

The fact that the training MCC value is greater than 0 shows that it performs better than a random classifier. Values under 0 would mean that a possible inverse relationship may exist between the features and the target class. When the hyper parameter combination model which achieved the best Matthews Correlation Coefficient (MCC) scoring was run against the testing dataset, the testing accuracy achieved was around 0.425. Figure 47 shows the evaluation classification report for the K-NN model.

Figure 47

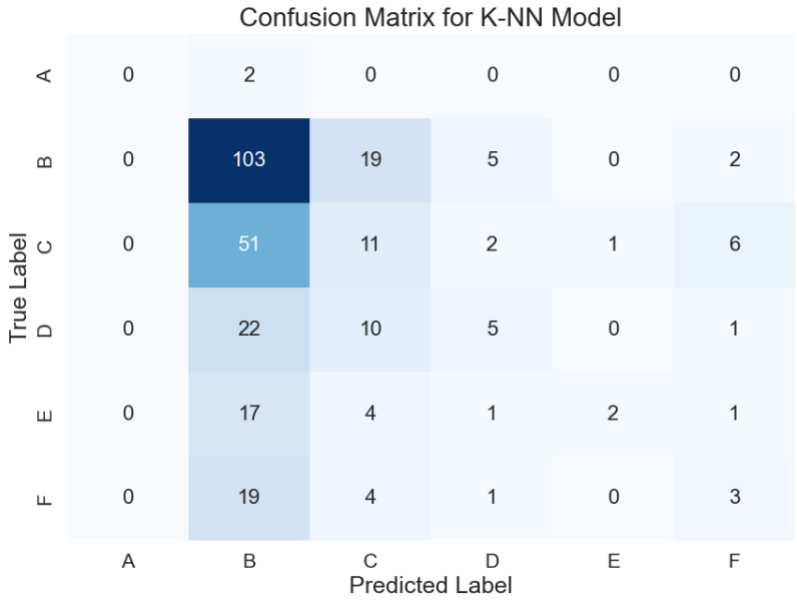
K-NN model classification report

```
Classification report:
              precision    recall  f1-score   support

           A       0.00      0.00      0.00         2
           B       0.48      0.80      0.60       129
           C       0.23      0.15      0.18        71
           D       0.36      0.13      0.19        38
           E       0.67      0.08      0.14        25
           F       0.23      0.11      0.15        27

    accuracy                           0.42       292
   macro avg       0.33      0.21      0.21       292
weighted avg       0.39      0.42      0.36       292
```

The classification report shows that class B has the best performance for recall and f1-score. This may be due to the large distribution of class B data in the dataset. Class E however, when using the K-NN model achieved the highest precision compared to the other classes. Meaning that the trained K-NN model can detect the Class E instanced in the feature space better than the other classes which is surprising considering that Class B has a total of 129 instances while Class E only has 25. Further insights can be seen from the confusion matrix shown in Figure 48.

Figure 48

K-NN Model Confusion Matrix



The K-NN model confusion matrix shows that the model classified most of the data instances into class B. This may be due to the original dataset being imbalanced with most of class B data instances occupying the feature space along with the fact that a relatively large number of K = 18 was used when configuring the nearest neighbor algorithm. Even though the weight was set to distances, the large number predefined ‘k’ value and the majority of class B instances could explain why the majority of the predicted label is classified as class B.

Support Vector Machine, One-vs-One scheme Model Results

The base SVM model was made using SVC from scikit-learn with the decision_function_shape parameter set to ‘ovo’ to handle multi-class classification. The hyperparameters that were tested with the grid search model are shown in Figure 49.

Figure 49

Hyperparameters that were tested for the SVM classifier.

```
param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
}
```

Using the training set and grid search with stratified k-fold cross validation, the best SVM model had the hyperparameters ‘C’ equal to 1, and ‘gamma’ equal to 10. The Matthews correlation coefficient, or MCC, on the training set was around 0.13, and the MCC on the test set was around 0.05. The results are shown in Figure 50.

Figure 50

Best SVM model hyperparameters, along with the training MCC and test MCC

```
Best model parameters: {'C': 1, 'gamma': 10}
Training MCC: 0.12894588949217328
Test MCC: 0.0540821426425056 25
```

Since the test MCC is slightly above 0, this means that the SVM model is performing marginally better than a random classifier. The macro precision/recall/F1-score and accuracy of the chosen SVM model on the test set are shown in the following classification report from the scikit-learn library:

Figure 51

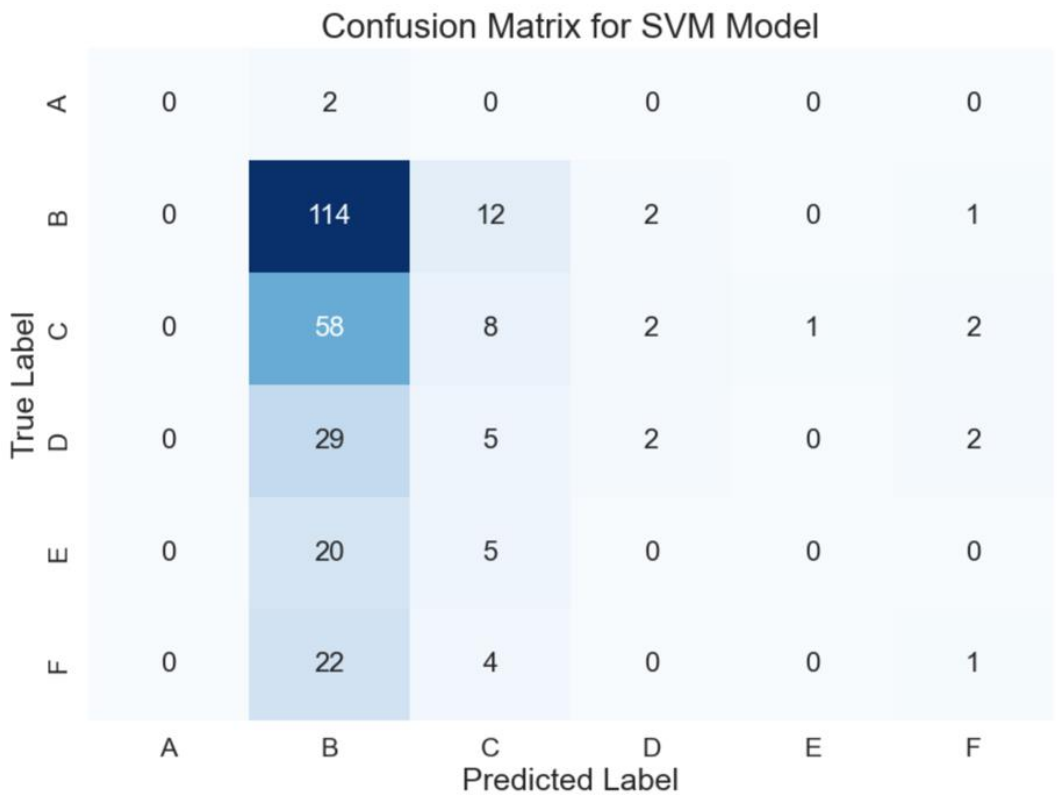
Classification report of the chosen SVM model

```
Classification report:
              precision    recall  f1-score   support

           A       0.00      0.00      0.00         2
           B       0.47      0.88      0.61       129
           C       0.24      0.11      0.15        71
           D       0.33      0.05      0.09        38
           E       0.00      0.00      0.00        25
           F       0.17      0.04      0.06        27

    accuracy                           0.43       292
   macro avg       0.20      0.18      0.15       292
weighted avg       0.32      0.43      0.32       292
```

From the classification report, it is observed that the model had low performance in general. Class B has the best performance, while the other classes have a low precision, recall, and F1-score (Classes A and E have 0 for these metrics indicating that none of the test set records corresponding to classes A and E were identified correctly). The test accuracy is 0.43, which is not too relevant since the test set is imbalanced. More insight into these metrics can be observed in the confusion matrix in Figure 52.

Figure 52

Confusion Matrix for the chosen SVM model



From the confusion matrix, we can observe that the SVM model classified the large majority of instances into Class B. This resulted in poor performance for the other classes. The macro-AUC-ROC score using a one-vs-one approach is around 0.52, which, along with the MCC, signifies that the SVM model is very slightly performing better than a completely random classifier.

Random Forest Model Results

The base model for the Random Forest model uses the RandomForestClassifier from scikit-learn. The parameter grid prepared for the Random Forest is shown in Figure 8. The criterion refers to how the model will split each node using different methods such as the GINI index, log loss or entropy calculations. N_estimators refer to the number of decision trees that will be running during the Random Forest training process. Max depth refers to the max depth of the tree, ensuring the number does not go too large as it may lead to overfitting. Setting a low

number of results in a very shallow tree which in turn causes underfitting. Min_samples_leaf refers to the minimum number of samples in a node for it to be split. Bootstrap can be set to True or False however, it is set to true as it reduces the training time for each tree and enhances the diversity of trees used in the ensemble (Kelleher, 2015).

Figure 53

Random Forest Model Hyperparameter Grid

```
criterion = ['gini','entropy','log_loss']
n_estimators = [100,200,300,500,1000]
max_depth = [25]
min_samples_leaf = [2]
bootstrap = [True]
```

Using the GridSearchCV for hyperparameter optimization with the scoring set to the Matthews Correlation Coefficient, the best model returned the following hyperparameter combinations shown in Figure 54. The optimized model has hyperparameter which includes the number of estimators set to be at 500 and the criterion to utilize the GINI index to determine the information gain of each descriptive feature at each root or node.

Figure 54

Random Forest Hyperparameter combination and results

```
Parameters: {'bootstrap': True, 'criterion': 'gini', 'min_samples_leaf': 2, 'n_estimators': 500}
Training MCC: 0.13213081430032153
Test MCC: 0.08497878800700892

AUC-ROC score: 0.5292571306670026
```

For the RF results, the fact that the training MCC value is greater than 0 shows that it performs better than a random classifier model. Values under 0 would mean that a possible inverse relationship may exist between the features and the target class. When the hyperparameters which yielded the best model result for MCC scoring had the testing dataset run

through the model, the testing accuracy achieved was around 0.425. Figure 10 shows the classification report for the Random Forest model.

Figure 55

Random Forest Model Evaluation Classification report

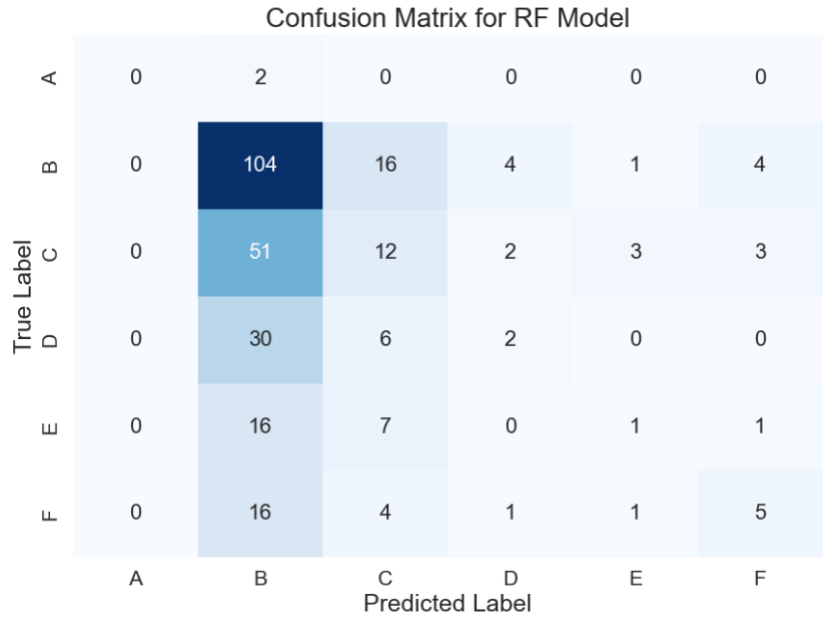
Classification report:

	precision	recall	f1-score	support
A	0.00	0.00	0.00	2
B	0.47	0.81	0.60	129
C	0.27	0.17	0.21	71
D	0.22	0.05	0.09	38
E	0.17	0.04	0.06	25
F	0.38	0.19	0.25	27
accuracy			0.42	292
macro avg	0.25	0.21	0.20	292
weighted avg	0.35	0.42	0.35	292

The classification report shows that class B has the best performance for all precision, recall and f1-score metrics. Meaning that the trained Random Forest Model is best at classifying instances into class B instance. As explained, this is mainly because the current dataset is very imbalanced with records from class B dominating the feature space. Figure 56 shows the random forest model confusion matrix.

Figure 56

Random Forest Model Confusion Matrix



Similar to the K-NN model, there is a very large classification of data in the B class due to the data source being heavily imbalanced. Given that the RF is a complex model the fact that the dataset only has 730 records might not be enough for the model to completely optimize itself. For hyperparameter tuning, given a more balanced dataset as the training dataset, less restrictions can be put on the parameters such as Bootstrap, max tree depth and min leaf samples as other possible hyperparameters may be able to yield better results.

XGBoost Model Results

The base XGBoost model was made using XGBClassifier from the XGBoost library. The hyperparameters that were tested with the grid search model are shown in Figure 57.

Figure 57

Hyperparameters that were tested for the XGBoost classifier.

```
param_grid = {
    'n_estimators': [50, 100, 200, 500, 1000],
    'learning_rate': [0.001, 0.01, 0.1, 1],
    'max_depth': [3, 5, 7, 9]
}
```

Using the training set and grid search with stratified k-fold cross validation, the best XGBoost model had the hyperparameters ‘n_estimators’ equal to 1000, ‘learning_rate’ equal to 0.001, and ‘max_depth’ equal to 5. The MCC on the training set was around 0.17, and the MCC on the test set was around 0.06. The results are shown in Figure 58.

Figure 58

Best XGBoost model hyperparameters, along with the training MCC and test MCC

```
Best model parameters: {'learning_rate': 0.001, 'max_depth': 5, 'n_estimators': 1000}
Training MCC: 0.1697981272008093
Test MCC: 0.06068150826814804
```

Since the test MCC is slightly above 0, this means that the XGBoost model is performing marginally better than a random classifier. The macro precision/recall/F1-score and accuracy of the chosen XGBoost model on the test set are shown in the classification report in Figure 59.

Figure 59

Classification report of the chosen XGBoost model

Classification report:

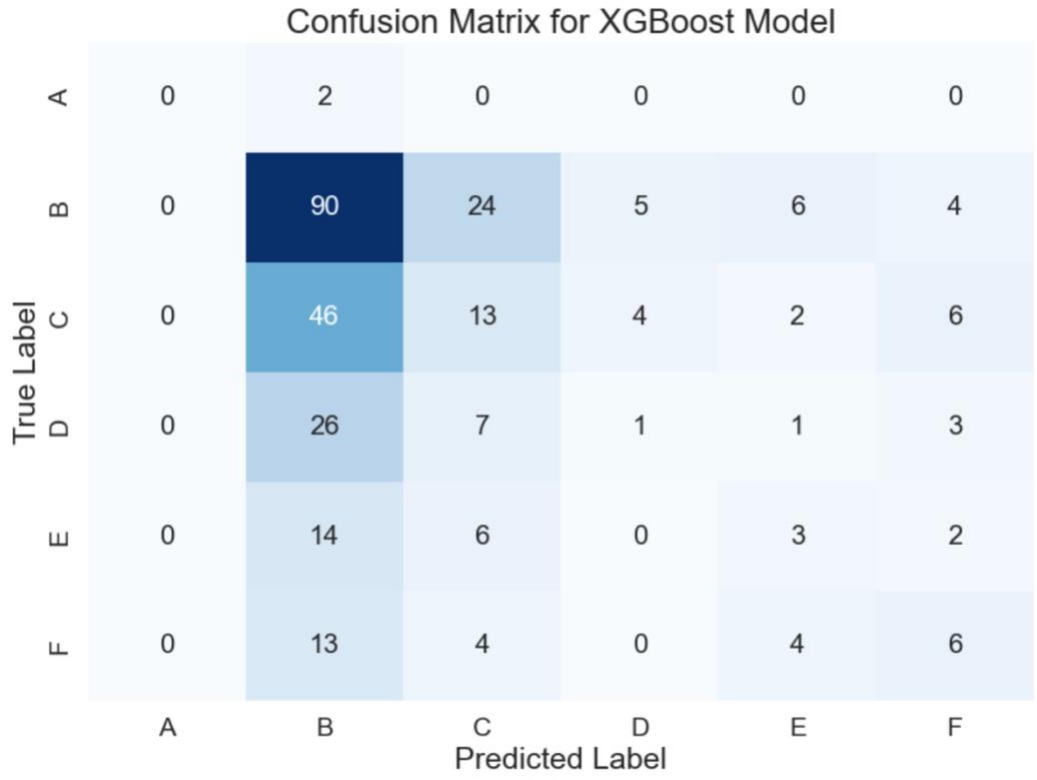
	precision	recall	f1-score	support
A	0.00	0.00	0.00	2
B	0.47	0.70	0.56	129
C	0.24	0.18	0.21	71
D	0.10	0.03	0.04	38
E	0.19	0.12	0.15	25
F	0.29	0.22	0.25	27
accuracy			0.39	292
macro avg	0.21	0.21	0.20	292
weighted avg	0.32	0.39	0.34	292

From the classification report, we can observe that the XGBoost model similarly exhibits a low performance. Class B has the best performance, while the remaining classes do not

perform well. For instance, a weighted macro average F1-score of 0.36 demonstrates that the model does not have good predictive ability on average when considering all classes. The test accuracy is 0.39, which again is not too relevant since the test set is imbalanced. In order to generate more insights into the classification report, a confusion matrix of the chosen XGBoost model on the test set is shown in Figure 60.

Figure 60

Confusion Matrix for the chosen XGBoost model



The majority of classes were classified as Class B. This resulted in low performance for the other classes. The macro-AUC-ROC score using a one-vs-one approach is around 0.56, which along with the MCC, further signifies that the XGBoost model is very slightly performing better than a completely random classifier.

AdaBoost Model Results

The base AdaBoost model was made using AdaBoostClassifier from the Scikit learn library. The hyperparameters that were tested with the grid search model are shown in Figure 61.

Figure 61

Hyperparameters that were tested for the AdaBoost classifier.

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.001, 0.01, 0.1, 1],
    'algorithm': ['SAMME', 'SAMME.R']
}
```

Using the training set and grid search with stratified k-fold cross validation, the best AdaBoost model had the hyperparameters ‘algorithm’ equal to ‘SAMME’, ‘learning_rate’ equal to 0.001, and ‘n_estimators’ equal to 50. The MCC on the training set was around 0.13, and the MCC on the test set was around 0.03. The results and best hyperparameter combinations are shown in Figure 62.

Figure 62

Best AdaBoost model hyperparameters, along with the training MCC and test MCC

```
Best model parameters: {'algorithm': 'SAMME', 'learning_rate': 0.001, 'n_estimators': 50,
Training MCC: 0.12634052135480076
Test Accuracy: 0.4006849315068493
Test MCC: 0.029683796198342556
```

The macro precision/recall/F1-score and accuracy of the chosen AdaBoost model on the test set are shown in the classification report in Figure 63.

Figure 63

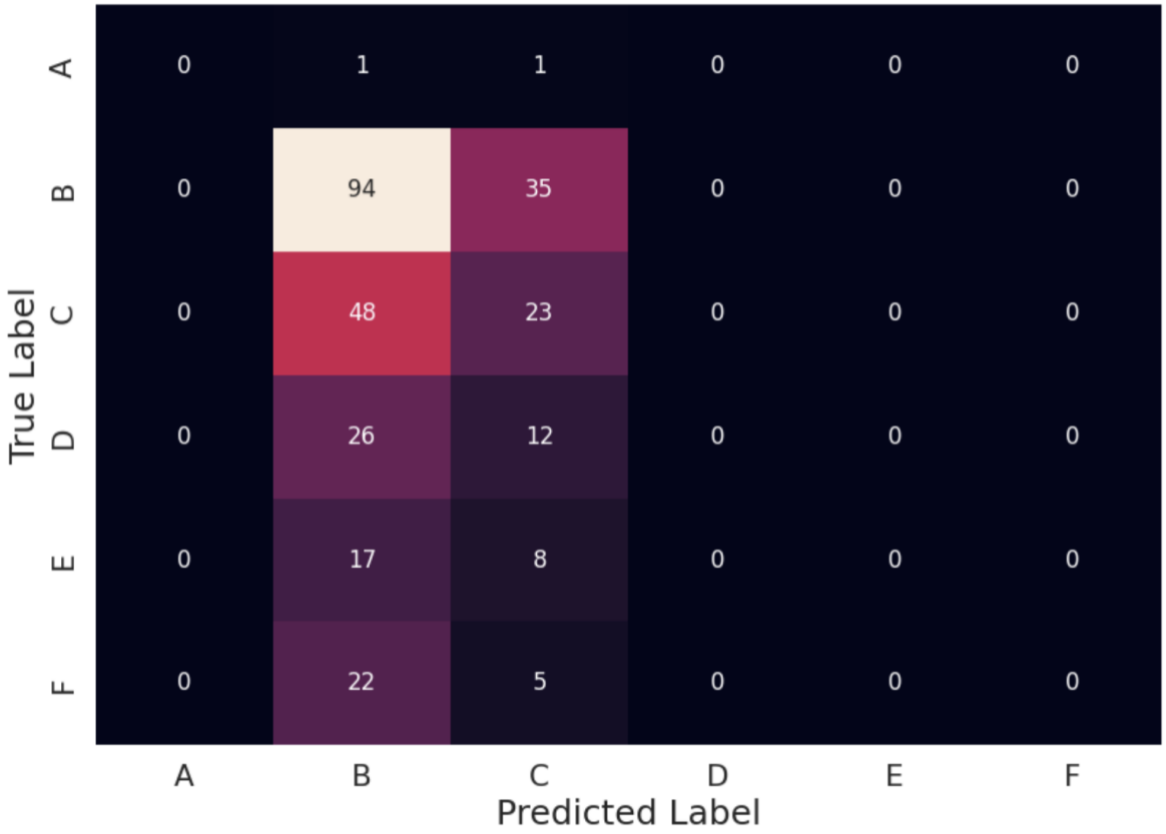
Classification report of the chosen AdaBoost model

```
Classification report:
              precision    recall  f1-score   support

           A       0.00      0.00      0.00         2
           B       0.45      0.73      0.56       129
           C       0.27      0.32      0.30        71
           D       0.00      0.00      0.00        38
           E       0.00      0.00      0.00        25
           F       0.00      0.00      0.00        27

    accuracy                           0.40       292
   macro avg       0.12      0.18      0.14       292
weighted avg       0.27      0.40      0.32       292
```

Looking at the classification report, we can see that the model is performing poorly for all classes, with precision, recall and F1-score being low for all classes. For example, the precision for class A is 0.0, which means that the model did not correctly predict any instance of class A. The recall for class D is also 0.0, which means that the model did not correctly identify any instance of class D. The F1-score is a measure of the balance between precision and recall, and it is also quite low for all classes. The results of the model indicate that the performance of the model is not very good, as the accuracy of the test set is only 0.400, which means that the model only correctly classified 40% of the instances in the test set. A confusion matrix of the chosen AdaBoost model on the test set is shown in Figure 64.

Figure 64

Confusion Matrix of the chosen AdaBoost model



The confusion matrix further confirms the poor performance of the model. For example, for class A, the model predicted one instance as class B and one instance as class C, which means that it did not correctly identify any instance of class A. For class B, the model correctly predicted 94 instances as class B, but it incorrectly predicted 35 instances as class C. This shows that the model is having difficulty distinguishing between class B and class C. The AUC-ROC score for the model is 0.505, which is only slightly better than random guessing (which would be a score of 0.5), indicating that the model is not very good at distinguishing between differences instances of the target variable.

ANN Model Results

A Sequential model with two hidden layers, each with a ReLU activation function and an output layer with a Softmax activation function, is used to build the model's architecture. The model's loss function is categorical_crossentropy, and "Adam" is the optimizer. First, the model's

architecture is built using the create_model() method, and the hyperparameter grid for grid search is defined using the GridSearchCV() function in Scikit-Learn. The model's KerasClassifier wrapper is then constructed, and cross-validation is performed using Scikit-Learn's StratifiedKFold() method.

The hyperparameters are tuned using the GridSearchCV() function, the KerasClassifier model, the hyperparameter grid, the StratifiedKFold method, and the Matthews Correlation Coefficient (MCC) as the scoring metric. Finally, the best hyperparameters and MCC scores are printed.The hyperparameters that were tested with the grid search model are shown in Figure 65.

Figure 65

Hyperparameters that were tested for the ANN model.

```
param_grid = {
    'epochs': [20, 50],
    'batch_size': [16, 32],
    }
```

Further analysis found that using a 16-batch size and 50 epochs produced the best results for the model. The MCC score for the training set was 0.12, indicating a relatively low correlation between the predicted and actual labels. However, the MCC score for the test set was lower at 0.08. The results are shown in Figure 66.

Figure 66

Best ANN model hyperparameters, along with the training MCC and test MCC

```
Best parameters: {'batch_size': 16, 'epochs': 50}
Training MCC: 0.11983625089894004
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
Test MCC: 0.08419620396993974
```

The macro precision/recall/F1-score and accuracy of the chosen ANN model on the test set are shown in the classification report in Figure 67.

Figure 67

Classification report of the chosen ANN model

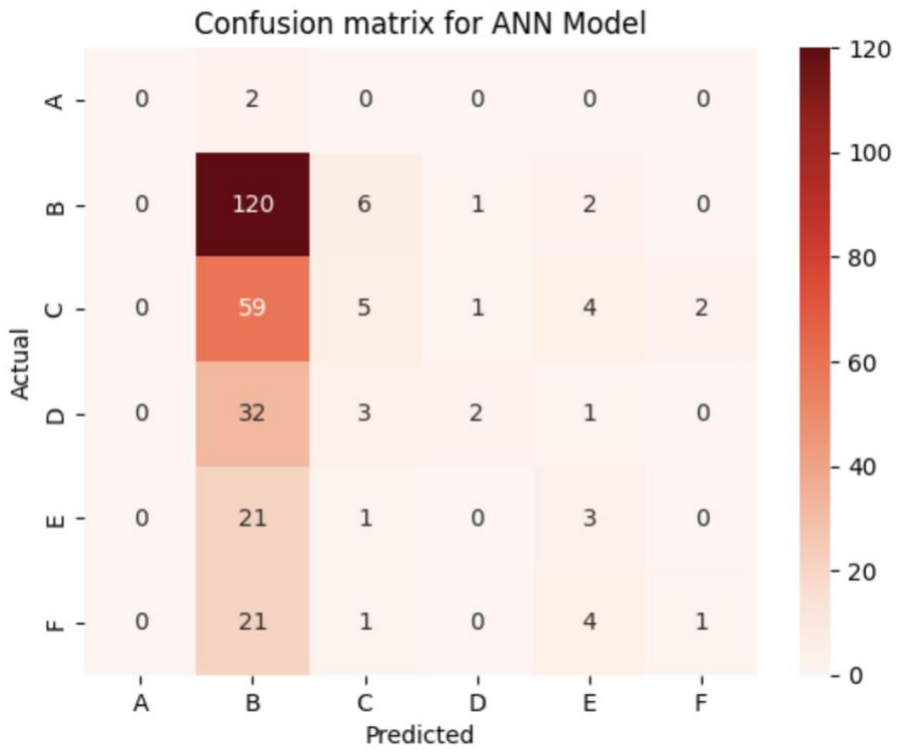
Classification report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.49	0.85	0.62	129
2	0.24	0.17	0.20	71
3	0.50	0.03	0.05	38
4	0.12	0.04	0.06	25
5	0.29	0.07	0.12	27
accuracy			0.43	292
macro avg	0.27	0.19	0.17	292
weighted avg	0.38	0.43	0.34	292

The model performs best when categorizing class 1 (class “B” originally), achieving a precision of 0.49, recall of 0.85, and F1-score of 0.62. However, it could improve its performance when classifying classes 0, 2, 3, 4 and 5, as its F1 scores are near zero for these categories. A weighted macro average F1-score of 0.34 shows how the model does not have a good ability to predict a class on average when considering every class. A confusion matrix of the chosen ANN model on the test set is shown in Figure 68.

Figure 68

Confusion Matrix for the chosen ANN model



The majority of classes were classified as Class B. This resulted in low performance for the other classes. The macro-AUC-ROC score is around 0.57, which along with the test MCC of 0.08, further shows that the ANN model is slightly better than a random classifier.

Evaluation Comparison of the different models

The following table, Table 10, provides an overview of the performances of every model with regards to various metrics. Out of the macro precision/recall/F1-score metrics, the weighted F1-score metric is chosen due to the imbalance of data in the test set.

Table 10

Summary of the performance of the different models with various evaluation metrics

Model	Training MCC	Test MCC	Weighted F1-Score	Macro AUC-ROC score	Test Accuracy
KNN	0.12	**0.09**	0.36	0.53	**0.43**

SVM	0.13	0.05	0.32	0.52	**0.43**
RF	0.13	0.08	0.35	0.53	**0.43**
XGBoost	**0.17**	0.06	**0.39**	0.56	0.39
AdaBoost	0.13	0.03	0.32	0.51	0.40
ANN	0.12	0.08	0.34	**0.57**	**0.43**

The k-NN, SVM, Random Forest, and ANN models have the highest test accuracies. Due to the imbalance present in the test set, the accuracy metric is not the most appropriate in this case. The k-NN model has the highest test MCC by a slight margin, with ANN and RF having the second highest test MCC values. The XGBoost model has the weighted average F1-score, while the ANN model has the highest macro-AUC-ROC score. Considering that the ANN model has the second highest test MCC value that is only 0.01 lower than the highest MCC test value and that the ANN model has the highest macro AUC-ROC score, it appears that the ANN model is a marginally better choice than the rest of the models (though the MCC and macro AUC-ROC scores signify that all of the models are slightly better than a random classifier). It is also worth noting that oversampling the training data (creating samples to make the number of instances of all the minority classes equal to the majority class) with SMOTE yielded very similar results on the test set.

Conclusion and Future Scope

In the future, the team aims to include a much larger amount of data. The collected historical wildfire data is solely from CAL FIRE and will thus gather wildfire data from other sources for future models. The team aims to include more predictor variables that can influence the scale of wildfires, such as the corresponding land cover (the type of land that is present in the area, for instance evergreen needleleaf forests versus deciduous needleleaf forests) for each

wildfire event. The methods to assign labels to the severity class target feature will also be re-examined. Usage of different augmentation techniques to combat the imbalance of the target features as well as different models with each of their own hyperparameter combinations should be explored as these models may yield better results. The team hopes that with the aforementioned steps, there will be a better separation between different severity classes and thus more accurate predictive models will be created. Better predictive models will lead to better applications for first responders during the event of multiple wildfires which will eventually lead to wildfire damage mitigation, saving the environment and lives in the process.

References

A. S. Mahdi and S. A. Mahmood, "Analysis of Deep Learning Methods for Early Wildfire Detection Systems: Review," 2022 5th International Conference on Engineering Technology and its Applications (IICETA), Al-Najaf, Iraq, 2022, pp. 271-276, doi: 10.1109/IICETA54559.2022.9888515.

Brownlee, J. (2016, August 17). A gentle introduction to XGBoost for applied machine learning. MachineLearningMastery.com. Retrieved April 28, 2023, from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 55(1), 119–139. https://doi.org/10.1006/jcss.1997.1504

Grandini, M., Bagli, E., & Visani, G. (2020, August 13). Metrics for multi-class classification: An overview. arXiv.org. Retrieved April 28, 2023, from https://arxiv.org/abs/2008.05756

Girtsou, S., Apostolakis, A., Giannopoulos, G., & Kontoes, C. (2021). A machine learning methodology for next day wildfire prediction. *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*. https://doi.org/10.1109/igarss47720.2021.9554301

Harrison, M. (2019). Machine learning pocket reference: Working with structured data in Python. O'Reilly Media, Inc.

Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class AdaBoost. Statistics and Its Interface, 2(3), 349–360. https://doi.org/10.4310/SII.2009.v2.n3.a8

Huot, F., Hu, R. L., Goyal, N., Sankar, T., Ihme, M., & Chen, Y.-F. (2022). Next day wildfire spread: A machine learning dataset to predict wildfire spreading from remote-sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1–13. https://doi.org/10.1109/tgrs.2022.3192974

Jiang, T., Bendre, S. K., Lyu, H., & Luo, J. (2021). From static to Dynamic prediction: Wildfire risk assessment based on multiple environmental factors. *2021 IEEE International Conference on Big Data (Big Data)*. https://doi.org/10.1109/bigdata52589.2021.9672044

Jurman, G., Riccadonna, S., & Furlanello, C. (2012). A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE*, *7*(8). https://doi.org/10.1371/journal.pone.0041882

Kelleher, J. D., Namee, M. B., & D'Arcy, A. (2015). Fundamentals of Machine Learning for Predictive Data Anayltics: Algorithms, Worked Examples, and Case Studies (1st ed.). The MIT Press.

Liang, H., Zhang, M., & Wang, H. (2019). A neural network model for wildfire scale prediction using meteorological factors. *IEEE Access*, *7*, 176746–176755. https://doi.org/10.1109/access.2019.2957837

Nahua Kang (2017, April 13). Multi-Layer Neural Networks with Sigmoid Function: Deep Learning for Rookies (2). Towards Data Science. Retrieved from https://medium.com/towards-data-science/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

OEHHA. (2022). (rep.). *Indicators of Climate Change in California (2022)*. Retrieved March 5, 2023, from https://oehha.ca.gov/media/04wildfires.pdf.

Rhee, J., & Carbone, G. J. (2007). A Comparison of Weekly Monitoring Methods of the Palmer Drought Index, *Journal of Climate*, *20*(24), 6033-6044. doi: https://doi.org/10.1175/2007JCLI1693.1

Salas, E. B. (2023, April 17). U.S. area burned by wildfires by state 2021. Statista. https://www.statista.com/statistics/217072/number-of-fires-and-acres-burned-due-to-us-wildfires/#statisticContainer

S. D. Ali et al., "GeoAI for Disaster Mitigation: Fire Severity Prediction Models using Sentinel-2 and ANN Regression," 2022 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES), Yogyakarta, Indonesia, 2022, pp. 1-7, doi: 10.1109/ICARES56907.2022.9993515.

Taunk, K., De, S., Verma, S., & Swetapadma, A. (2019). A brief review of nearest neighbor algorithm for learning and classification. *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. https://doi.org/10.1109/iccs45141.2019.9065747

UCSD. (n.d.). Data to knowledge. WIFIRE. https://wifire.ucsd.edu/data-to-knowledge

U.S. Department of the Interior. (2022, March 8). *Wildfire causes and evaluations (U.S. National Park Service)*. National Parks Service. Retrieved March 5, 2023, from https://www.nps.gov/articles/wildfire-causes-and-evaluation.html

Yadav, S., & Shukla, S. (2016). Analysis of k-fold cross-validation over hold-out validation on colossal datasets for Quality Classification. *2016 IEEE 6th International Conference on Advanced Computing (IACC)*. https://doi.org/10.1109/iacc.2016.25

Zope, V., Dadlani, T., Matai, A., Tembhurnikar, P., & Kalani, R. (2020). IOT sensor and Deep Neural Network based wildfire prediction system. *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. https://doi.org/10.1109/iciccs48265.2020.9120949