CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

AN ENSEMBLE APPROACH FOR CALIFORNIA WILDFIRE PREDICTION USING MINI BATCH AND MULTIPROCESSING

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

by

Sergio Ramirez

December 2024

The thesis of Sergio Ramirez is approved:

Katya Mkrtchyan, Ph.D.	Date
Marjan Asadinia, Ph.D.	Date
Xunfei Jiang, Ph.D., Chair	Date

California State University, Northridge

Acknowledgements

I’m incredibly grateful to my committee chair, Dr. Xunfei Jiang, for her constant support and insightful guidance. Her feedback and encouragement have been essential in shaping my work and driving me toward excellence. I truly appreciate her impact on my academic journey.

I am also profoundly thankful to Dr. Marjan Asadinia and Dr. Katya Mkrtchyan for their support and guidance and for the opportunity to serve as my committee members. In addition, I would like to thank Dr. Li Liu and Dr. Mario Giraldo for their guidance and support throughout the project.

I would also like to extend my sincere and wholehearted appreciation to my wife, Neena Ramirez. From inspiring me to attend graduate school to encouraging me to strive for greatness, her unwavering support has been the cornerstone of this journey. She has stood beside me through late nights, moments of doubt, and countless sacrifices, always offering words of reassurance and kindness. Her faith in me has been a profound gift, one that has carried me through to this achievement.Without her boundless encouragement and unconditional love, none of this would have been possible. For all these reasons and more, I am forever deeply appreciative and grateful for her belief in me. This accomplishment is as much hers as it is mine, and I cannot overstate the impact of her steadfast presence in my life. To Neena, my partner and unwavering source of strength, thank you for everything.

Table of Contents

List of Tables

List of Figures

ABSTRACT

AN ENSEMBLE APPROACH FOR CALIFORNIA WILDFIRE PREDICTION USING MINI BATCH AND MULTIPROCESSING

By

Sergio Ramirez

Master of Science in Computer Science

Given that California deals with massive amounts of wildfires each and every year, there is an extreme importance in studying the prediction of when wildfires are likely to occur. There have been various studies previously that have attempted to study the prediction of wildfires in California; however, very few explore the usage of a decade worth of data. Moreover, given the vast amount of data, previous studies that have used several years of data, suffer low accuracy and require massive amounts of training time, in addition to the heavy compute resources required. This project aims to assist in raising the accuracy and lowering the runtime of such compute models, while also assisting in reducing damages from wildfires in California through the use of: dataset mini batching, Graphical Processing Unit (GPU) multiprocessing, and ensemble methods with multiple machine learning models such as Support Vector Machine (SVM), Gaussian Naive Bayes (GNB), and K-Nearest Neighbor (KNN). Features considered in the dataset are: longitude, latitude, landsat enhanced vegetation index (EVI), thermal anomaly (TA), land service temperature (LST), wind, fire, and elevation. Experimental results show up to a 50% decrease in training time while achieving an accuracy of up to 98.1%.

Chapter 1

Introduction

The state of California is plagued with several thousand wildfires each and every year, with a 5 year average of just over 6,500 reported and over 1.2 million acres burned [1]. With each reported fire there is an inherit risk that such an event may cause harm to structures, nearby vegetation, and most importantly animal and human life. In addition, the major costs associated with firefighting, utility wildfire prevention, and insured property losses can easily exceed \$3 billion, \$5 billion, and \$10 billion respectively, like in the year 2020 [2]. As such, it is essential that further means of efficiently and accurately being able to mitigate and prevent such natural disasters should be examined.

Various past studies have used publicly available data regarding wildfires and their correlated factors, and they have applied a unique number of machine learning approaches on such data. However, previous studies have faced long standing issues such as: long training times with a high dependence on available compute resources, results suffering from large class imbalances present in the datasets used, lack of usage of the graphics processing unit (GPU), and extensibility or applicability. Since the data are contained over a 10-year span, consumption and applying machine learning concepts on such data requires a large amount of time, especially while only running on the central processing unit (CPU). Making any parameter adjustments to any machine learning model also proves to be exceptionally difficult as testing even a single alteration may range from hours to days.

In order to further enhance previous research, this project aims to use the existing wildfire data that was aggregated by researchers [3] [4], and a previous senior design team[1] as a basis for which this study can spring from. We reused their machine learning models, such as: SVM, GNB, and KNN which showed high accuracy on wildfire prediction: 96%, 80%, and 96% respectively.

The objective of this study is to leverage machine learning models and data from California wildfires collected by previous studies [3] and [4], building on and enhancing the time efficiency and performance of wildfire prediction models. The goal is to create a flexible framework that can be scaled and that others can utilize with any type of data or dataset(s).

In this project, we explore the usage of: dataset mini batching, dataset multiprocessing, and ensemble 3 machine learning models: SVM, GNB, and KNN. We also accelerated our training and testing using available GPUs. The method used were also reinforced by testing accomplished on two machines: (1) Windows machine with an Intel 6 core i5 9600k and an Nvidia 2070 GPU and (2) Ubuntu machine with a 20 core Intel Xeon E5-2690 CPU with and Nvidia Tesla P4 GPU. Our experimental results demonstrated great potential for scaling with even larger quantities of data than the 10 years of data used. There was also decreased training and testing time in the 3 model ensemble when compared to models

[1] https://faridko26.github.io/wildfire.github.io

ran independently and only on the CPU. Overall experimental results showed up to a 4% improvement in accuracy and up to 50% reduction in training and testing time when the 3 models ensemble was compared to models running independently and directly on the CPU.

Furthermore, the main contributions of this project include:

- Provided concrete results on the applicability of utilizing both dataset mini batching and multiprocessing on the GPU
- Reduced the machine learning model training time
- Prediction of fire occurrences in California with an accuracy up to 98% using a 3 model ensemble
- Creation of a streamlined and flexible framework that can be used to train large datasets, regardless of size

The rest of this paper is organized as the following: Chapter 2 discusses related work; Chapter 3 presents the design of this study; Chapter 4 describes the methodologies that will be used; Chapter 5 illustrates the results and analysis; and Chapter 6 concludes the project and discusses future work.

Chapter 2

Related Works

The following sections delve into papers specifically focused on architecture for machine learning models and with systems with a focus on California wildfire prediction and analysis. These papers explore innovative designs and optimizations tailored to enhance the efficiency and performance of machine learning algorithms. Further, these works lay an architectural groundwork to further the research done in California wildfire prediction.

2.1 Wildfire Prediction Systems

Ashima et al. sourced data from sources such as: NCEO, fire history data from FRAP, and remote sensing from the Landsat 8, and combined machine learning models to form an ensemble and demonstrated a impactful way to predict wildfire risks. [5]. In the ensemble there were two groups of models: one group had SVM, XGBoost, and Random Forest, and the other group had CNN, LTSM, and SVM models. The data they used consisted of: weather data, fire history data, remote sensing (vegetation) data, and satellite data. Each group of models was fed a different dataset. For the first group, weather data and fire history data were used. In the second group, remote sensing (vegetation) data and fire history data were used. The results of both groups are brought together using a cellular automaton or discrete mathematical model, in order to use majority voting. Overall this architecture and use of datasets produced a result of 100% for fire risk prediction.

Hernandez and Hoskins, introduced an interesting approach to apply machine learning on wildfire prediction [6]. They used randomized data samples from satellite data, which include MERRA-2 and USGS Landsat 8, and used a range from 2013 to 2022. The primary focus of this paper revolves around the prediction of wildfires in the Central Valley. Moreover, the machine learning models used were: Random Forest, Decision Trees, Naïve Bayes, and Neural Networks. Although Random Forest was seen as an ensemble machine classifier in which higher accuracy could be shown, Decision Trees preformed the best with 550 maximum splits and with an F1-Score of 0.689.

Adhikari et al. used unique remote sensing data such as: soil condition, vegetation, and weather, and a unique architecture that focuses on real time wildfire progression prediction [7]. By integrating satellite data and remote sensing techniques. The research combines various data sources to analyze historical wildfire patterns, environmental factors, and geographical features to forecast wildfire spread. In terms of architecture, the paper uses a complex fire progression model that can be described as a convolutional Neural Network that is given a fire spread award and an observed state. Results produced from this study were 85.63% accuracy and an F1 score of 91.98%

Cheng et al. proposed to use two machine learning models to predict wildfire risk in Northern California [8]. The first, An ensemble model that combines the predictions of two base random forest models trained on different subsets of the data. The second, a combined model that uses a single random forest classifier trained on the full dataset including weather, vegetation, terrain, and power line data.Further, the ensemble model uses an AdaBoost meta-estimator to combine the predictions of the two base models, while the combined model directly trains the random forest on the combined dataset. The combined model outperforms the ensemble model, achieving an accuracy of 92% .

Jiang et al. proposed static and dynamic prediction models to analyze and assess wildfire risks in California using multiple environmental factors [9]. For static risk prediction, the authors employ Multi-layer Neural Network (MLNN), Logistic Regression (LR), Support Vector Machine (SVM), and Random Forest (RF) to classify grid cells into low, medium, and high risk zones. For dynamic risk prediction, the authors use a MLNN model to predict whether there will be at least one large wildfire (over 300 acres) in a given grid and year. The authors also performed counterfactual analysis using the trained dynamic prediction model to understand the effects of changing factors such as NDVI, PDSI, tree mortality, and population density on the risk of wildfires. The outcomes of this study demonstrate a 64.5% overall accuracy.

The papers described above are related work that primarily focus on utilizing diverse architectures and datasets. Ashima et al. used an ensemble approach to train two groups of models with different datasets, and results from these models were taken using majority voting [5]. Hernandez and Hoskins focused on a Random Forest with a Decision Tree architecture, and interestingly the Random Forest used an ensemble approach to aggregate the result produced. Adhikari et al. developed a real-time fire spread model with 91.98% F1-score [7]; Cheng et al. achieved 92% accuracy for Northern California wildfire risk prediction using ensemble methods [8]; and Jiang et al. employed static and dynamic models for risk assessment across California, achieving 64.5% accuracy [9]. It should be noted that from the paper's read and analyzed, [9], is in the minority of research that attempts to actively achieve real time results. Collectively, these studies illustrate the effectiveness of diverse machine learning architectures, from ensemble methods and decision trees to neural networks; they relate in enhancing wildfire prediction accuracy across different regions and datasets, with an emphasis on the framework used.

2.2 Wildfire Prediction Models

In the subsequent section, the related works focus on machine learning models. From models such as XGBoost to deep learning models, the papers listed below attempt to make use of common models in order to achieve the most accurate results. This introduction sets the stage for a deeper examination of how these common machine learning models serve provide excellent accuracy amongst the various features of wildfires.

Kaylee Pham et al. sourced high quality data from California fire incident records and four remote sensing datasets, including NDVI, LST, TA, and burned area (BA) from LP DAAC, to predict wildfire occurrences using machine learning models. [3]. The datasets included fire records to label fire conditions, and remote sensing data to provide environmental features like vegetation health and land surface temperature. Furthermore, they built two datasets out of the cumulative data, (1) Treating California as a single region and

(2) dividing the state into counties. These datasets were used to train five machine learning models ANN, SVM, KNN, GaussianNB, and Logistic Regression. The ANN model achieved the highest accuracy 89% at the state level, while KNN reached 97% accuracy at the county level, followed by SVM at 96%.

Castrejon et al. sourced data from various geospatial datasets including remote sensing data from NASA's AppEEARS tool, fire records from CAL Fire, and weather data from GHCNd, and applied multiple machine learning models to predict wildfire risk in California. [4] They used an architecture that utilized oversampling techniques like SMOTE to address class imbalance and generated 3D visualizations to represent geospatial data over time. The complete data ranged over a decade, from 2013 to 2023. Four machine learning models were used: K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GaussianNB), Logistic Regression (LR), and Multi-Layer Perceptron (MLP), each trained on different combinations of years and sizes of the datasets(s). The data consisted of enhanced vegetation index (EVI), land surface temperature (LST), thermal anomalies (TA), burned area (BA), and average wind speed (AWS). Moreover, the models were trained on a binary classification problem to identify fire-prone regions. The MLP model achieved the highest accuracy at 82.97%.

Singh et al. utilized a diverse dataset encompassing various seasons, locations, and weather patterns from multiple sources, meticulously cleaned for errors and gaps to suit machine learning algorithms [10]. Employing Decision Tree Regressor, XG Boost Regressor, and Artificial Neural Network (ANN), the models forecasted wildfire-affected regions based on satellite imagery and weather data features. Evaluation through RMSE scores revealed the Decision Tree Regressor outperforming XG Boost Regressor and ANN, demonstrating the models' efficacy in locating and predicting wildfire dynamics. Results produced were measured using the Root Mean Squared Error (RMSE), where the lower the value, the better performance was. Results were: 0.1501 for the Decision Tree Regressor, 0.224574 for the XG Boost Regressor, and 0.155 for the ANN.

Pahuja and Rivero use predictive models that anticipate wildfire intensity by harnessing environmental parameters and utilizing weather data [11]. Employing feature engineering techniques, they sift through diverse data sources to extract pertinent insights, thereby improving predictive accuracy. Employing a combination of classification and regression methodologies, they delve into different aspects of prediction, underscoring the significance of comprehensive datasets in achieving dependable wildfire size estimations. The machine learning models uses were: Random Forest, Naive Bayes, and Decision Trees, and Decision Tree's had the highest accuracy of 92%.

Nocerino and Ghosh tested various machine learning methods like logistic regression, decision trees, random forest, Naive Bayes, support vector machines, stochastic gradient descent, and XGBoost to predict building damage during wildfires [12]. The decision tree model reached the highest accuracy at 92%. They checked the models using 10-fold cross-validation and then tested the best one on a separate set.

Malik et al. introduced two data-driven machine learning methods for predicting wildfire risk in Northern California: an ensemble model combining outputs from two base random forest models and a single random forest model utilizing all data features together. Trained on a comprehensive dataset encompassing weather, vegetation, terrain, and powerline data, the combined model achieved superior performance with 92% accuracy, while the ensemble model, stacking outputs from the base models, achieved 84% accuracy [13]. Both models underscore the importance of integrating diverse spatial and temporal data parameters beyond traditional weather data for accurate wildfire risk assessment.

Girtsou et al. introduced a machine learning approach for predicting wildfires the following day, employing and fine-tuning various cutting-edge algorithms such as Tree Ensembles (Random Forest, XGBoost, LogitBoost) and Neural Networks [14]. Through a thorough cross-validation procedure, the authors identify the top-performing models, considering the significant class imbalance and spatial-temporal correlations within the dataset. The findings showcase the efficacy of the proposed models, achieving recall rates exceeding 90% for fire incidents while maintaining satisfactory performance for non-fire instances.

These studies highlight the effectiveness of varied machine learning architectures in enhancing wildfire prediction accuracy across distinct datasets and environmental parameters. Kaylee Pham et al. demonstrated high predictive accuracy in California using ANN and KNN models at state and county levels [3], while Castrejon et al. applied SMOTE and 3D visualizations, achieving an 82.97% accuracy with MLP [4]. Singh et al. found that Decision Trees outperformed other models in wildfire spread prediction, achieving a low RMSE of 0.1501 [10]. Pahuja and Rivero used feature engineering to boost wildfire intensity predictions, with Decision Trees reaching 92% accuracy [11], while Nocerino and Ghosh's model for predicting building damage in wildfires also performed best with Decision Trees [12]. Malik et al. reached 92% accuracy in Northern California wildfire risk prediction by combining random forest models [13], and finally Girtsou et al. achieved over 90% recall in next-day predictions using ensemble and neural network models [14]. Furthermore, together these studies demonstrate the strong potential of ensemble, decision tree, and KNN models for accurate wildfire risk and impact forecasting.

Chapter 3

Design

As shown in Fig. 3.1, this project is composed of 5 components: Data ingestion, Data Processing, mini batching of each dataset, multiprocessing, and ensemble methods. Each dataset will follow the flow depicted and if multiple datasets are selected, then each set will follow this flow in a sequential fashion.

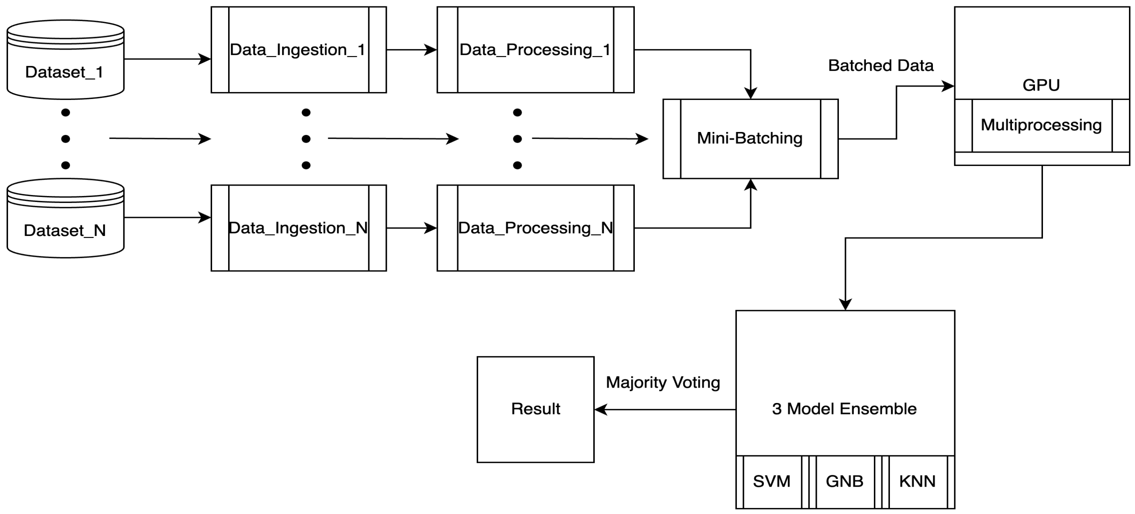


Figure 3.1: Framework of the project

Data ingestion begins by loading the dataset(s) into memory. This step ensures that the raw data is available and ready for the following processing stages. However, when dealing with large datasets, this step may become a slight bottleneck, as substantial memory and processing resources may be required, potentially affecting the speed and efficiency of the processing.

Data processing will be applied to each entry and will involve cleaning, adjusting any class imbalances, sampling, and feature extraction and scaling so that the data will be in a suitable format for model input. This phase helps transforms raw data into features that machine learning models can make use of, ensuring consistency and removing any noise that could negatively affect model performance of each individual model or the whole ensemble.

The mini batch portion of the system will take each of the dataset(s) and divide each one into smaller and more efficient subsets, which allows for a more efficient and manageable training and evaluation routine.

GPU multiprocessing will leverage the parallel compute option available in order to accelerate and enhance the speed of model training.

Processed and batched data will then be fed into a 3 model ensemble that includes SVM, GNB, and KNN. It will predict outcomes based on a majority voting strategy

3.1 Dataset Features

Table 3.1: Dataset Features

Field	Description
Date	Date of observation
Longitude	Longitude coordinate
Latitude	Latitude coordinate
EVI	Enhanced Vegetation Index
TA	Temperature
LST	Land Surface Temperature
Wind	Wind speed
Elevation	Elevation above sea level
Fire	Presence of fire (binary)

As previously mentioned, the data was sourced from past researchers in [3] and [4]. Researchers in those studies sourced data from different sources; however, they combined them to form bulk dataset(s). Table 3.1 displays the 9 attributes considered in each dataset, alongside with a description. Date, longitude, and latitude data facilitate accurate mapping and tracking of wildfire occurrences over time. Enhanced Vegetation Index (EVI), thermal anomalies (TA), and land surface temperature (LST), obtained from NASA's remote sensing data, are used to evaluate vegetation conditions and identify temperature irregularities that may indicate fire hazards. Wind speed information, gathered from weather stations, helps simulate fire spread, while elevation data reveals how terrain influences fire dynamics. Lastly, the fire presence indicator, based on historical fire data, offers clear categorization of fire events an will become the target field, since this is a classification task based on the presence of fire.

Table 3.2: Sample Data View With Data Types (2020)

Date	Longitude	Latitude	EVI	TA	LST	Wind	Elevation	Fire
Object	**float64**	**float64**	**int64**	**int64**	**int64**	**int64**	**int64**	**float64**
2020-01-01	-123.754626	41.994986	3037	5	13962	23	888.430303	0
2020-01-01	-123.750462	41.994986	2468	5	13962	23	851.066667	0
2020-01-01	-123.746298	41.994986	2468	5	13962	23	706.412121	0
2020-01-01	-123.742134	41.994986	2468	5	13962	23	648.927273	0
2020-01-01	-123.737970	41.994986	4013	5	13968	23	614.103030	0
2020-01-01	-123.733806	41.994986	2392	5	13968	23	565.963636	0
2020-01-01	-123.729642	41.994986	2392	5	13955	23	645.315152	0
2020-01-01	-123.725478	41.994986	3340	5	13955	23	601.721212	0
2020-01-01	-123.721314	41.994986	3682	5	13960	23	553.042424	0
2020-01-01	-123.717150	41.994986	3584	5	13960	23	525.763636	0

Table 3.3: Statistical Summary of Selected Features (2020)

Statistic	Longitude	Latitude	Fire	Elevation
count	9.803420×10^7	9.803420×10^7	9.803420×10^7	9.803420×10^7
mean	-1.211427×10^2	3.865945×10^1	7.202691×10^{-4}	1.136879×10^3
std	1.764654×10^0	2.271792×10^0	2.682816×10^{-2}	6.157287×10^2
min	-1.241544×10^2	3.255906×10^1	0.000000×10^0	-6.637576×10^1
25%	-1.227053×10^2	3.720829×10^1	0.000000×10^0	6.334121×10^2
50%	-1.213728×10^2	3.923711×10^1	0.000000×10^0	1.136106×10^3
75%	-1.199695×10^2	4.031610×10^1	0.000000×10^0	1.605794×10^3
max	-1.143189×10^2	4.199499×10^1	1.000000×10^0	3.898818×10^3

Tables 3.2 and 3.3 show the data types of each unique field in each dataset; int64 can be seen as being the most common data type and object as the least common data type. The Statistical summary was derived from the year 2020. It should be noted the mean of the fire variable, is relatively low, suggesting that within that year a fire was a relatively rare occurrence. This results in an overall large class imbalance which will be addressed in the pre-processing step. In addition, the deviations for longitude, latitude, and elevation are on the higher end; this indicates that there are outliers that could impact the machine learning model results. This will also be explored in the pre-processing steps as a means to scale the data appropriately.

3.2 Dataset(s) Ingestion

Since we are dealing with a decade worth of data, the pre-processed datasets are broken down by year. Each dataset is about 9.7 million rows and can range up to 7.81 GB. The time for loading a dataset of that size is system dependent, but can also range between a couple minutes to upwards of more than 5 minutes to accomplish. More often, that not, if the system does not have enough memory to do so, the program will abruptly quit, therefore it is important to have the available resources when handling such large datasets. To depict the runtime of simply loading a dataset into memory, a sample experiment was conducted.

Table 3.4: Preprocessed Datasets

Years	Size	Num of Rows
2013	7.80 GB	9.7 million
2014	7.81 GB	9.7 million
2015	7.81 GB	9.7 million
2016	7.81 GB	9.8 million
2017	7.72 GB	9.7 million
2018	7.72 GB	9.7 million
2019	7.71 GB	9.7 million
2020	7.74 GB	9.8 million
2021	7.71 GB	9.7 million
2022	7.70 GB	9.7 million

As demonstrated in Table 3.4, there are a total of 10 years worth of data, each year split into its own dataset, with an average size of 7.65 GB. Reading one years worth of data into memory can take upwards of 2 to 5 minutes, which correlates to a an increased runtime for any training or prediction using machine learning models. Further, each dataset has an average of about 9.7 million rows. This corresponds to several hundred thousand data points for each day or about 800k rows for each month in that given year. Moreover, it can be seen that such a gigantic dataset for each year cannot be loaded into memory without given a somewhat reasonable amount of time.

There exists two optimizations that can be done in terms of reading the data in from its CSV format into system memory. The first is through the usage of chunking or reading the data in chunks of specified rows. By chunking, the process of reading will be more memory efficient. For example, without chunking it may be required that a system should have more than the total size of any given dataset in available memory in order to read it, but with chunking, the reading can be done with less resources, since at any given time, the system can be fixed to read a certain allocation of data. This comes at the cost of potentially increasing the time to read since there will be more disk input and output reading. Continuing, by default the Pandas library [15], in Python [16], has its own engine or parsing mechanism that it uses to read CSV data, and the second optimization is by utilizing a different CSV parser known as PyArrow [17], that is specifically built for processing large CSV files quickly. Luckily, incorporating the usage is as simple as adding one additional flag to the read function.

Table 3.5 shows the average time to load data into memory for each year of data with the various optimization methods. A chunking size of 10,000 rows was used as it is an effective base to start from, and fits the constraints of most personal machines. For the sake of this project, since the time to read varies just slightly between using Pandas' standard reading algorithm and using PyArrow, the approach taken is to use the standard defined CSV reading that Python's Pandas library provides.

Table 3.5: Time to load datasets into memory (Seconds)

Years	**Pandas**	**Chunking (10,000 Rows)**	**PyArrow**
2013	72.50	125.00	71.20
2014	81.98	128.50	73.40
2015	84.91	123.56	75.60
2016	89.74	124.18	75.60
2017	83.00	118.67	76.12
2018	78.79	90.26	76.51
2019	94.30	119.05	80.30
2020	97.39	124.16	81.23
2021	89.49	123.53	82.60
2022	93.51	124.60	76.12

3.3 Dataset(s) Processing

The main steps for data set processing include data cleaning, data re-balancing, and data scaling.

3.3.1 Data Cleaning

We scan each data set and locate data that is either redundant or perhaps not necessary. Since for both our model training and evaluation we will need to load the entirety of the dataset, reducing the size of the data that is unnecessary by dropping redundant and unnecessary data provides immense benefits. Further, any malformed or data that is not of a numeric format type can also be removed since it does not benefit or aide our calculations. For example, each dataset has a 'Date' column, in the format 'YYYY-MM-DD' which Pandas will see as an object data type. This format is not supported; therefore we manually updated the format to be in the date type format when reading the csv into memory. Further, since there are thousands of data points occurring for each day, and the 'Date' column is static for each day of entry, an initial idea was to drop the entire column from each year; however, we thought that by doing that we would lose the relation between rows of data and that specific date. We also observed that whether or not we kept the column, the time to train and test was relatively similar. It should also be noted that all years of data contained a very small amount of null values for the elevation column, so we used the mean of the values in that column to change the Null values into numeric values. Listing 3.1 shows a script on how the 'Date' would be altered into a date type format and how the null values in elevation could be filled in.

Listing 3.1: Data Cleaning

```
import pandas as pd
# Read data in an manually make the Date column as Date type
df = pd.read_csv(fire_data, parse_dates=['Date'])
# We fill the na values, since some models cannot take in NA input
if df['Elevation']isnull().any():
    df['Elevation'] = df['Elevation'].fillna(df['Elevation'].mean())
```

3.3.2 Data Re-balancing

One issue present in the data is that there is a major class imbalance; 1:1200 to 1:32000 fire-prone to non-fire-prone data points from year to year [4]. Having an imbalance like the formerly mentioned one can lead to issues such as: bias towards a specific class, misrepresentation of results, and even increased risk of over fitting on the minority class. Previous studies utilized: Synthetic Minority Oversampling Technique (SMOTE), Adaptive Synthetic Algorithm (ADASYN), SMOTE with Tomek Links (SMOTE + TOMEK), and SMOTE with Edited Nearest Neighbors (SMOTE + ENN) [4]; however, even with closing the gap of the class imbalance, the model results still suffered greatly. Furthermore, applying these techniques also increases the run time for training, which correlates to an increased time in having to wait for any machine learning model to complete the training/testing loop.

Since the imbalance cannot be ignored and we do not want to add much computational overhead by creating synthetic data, a simple, yet crucial, insight is to under sample the majority class based on the size of the minority class [18]. The implementation can be seen in a series of steps: (1) Get all of the minority and majority occurrences within the dataset, (2) take the count of the minority class occurring within the dataset and make the total majority count equal some ratio greater than or equal to the minority count thereby reducing the count of the majority class, and (3) combine the classes back together to form one new dataset with a smaller imbalance ratio.

Listing 3.2: Addressing Class Imbalance

```
# Separate majority and minority classes for tackling imbalance
minority_class = df[df['Fire'] == 1]
majority_class = df[df['Fire'] == 0]

# 10-20% is good since data is large
total_sample_size = int(0.2 * len(df))

# Use the entire minority class since there are not enough samples
minority_sample = minority_class
minority_sample_size = len(minority_sample)

# 25% larger than the minority class so we don't have equal balance
majority_sample_size = int(minority_sample_size * 1.25)

# Undersample the majority class to match the size of the minority class
majority_sample = majority_class.sample(n=majority_sample_size, random_state=1)

# Combine the minority and majority samples
balanced_sample_df = pd.concat([minority_sample, majority_sample])

# Shuffle the data
balanced_sample_df = balanced_sample_df.sample(frac=1, random_state=1).reset_index(drop=True)
```

Listing 3.2 depicts how such a technique can be implemented fairly quickly and does not add much computation or overhead in terms of runtime. In this example, by applying down sampling, the ratio between the fire-prone to non-fire-prone becomes 5:4 or, on average, 104,148 fire-prone versus 80,038 non-fire-prone points. Table 3.6 shows a randomly shuffled view of what the data looks like after being down sampled. Interestingly, this can be contrasted with table 3.2 to notice the appearance of the fire-prone class being more abundant. Moreover, in general, by applying this technique, the data is ensured to be more balanced and there is no need to create synthetic data. In addition, since each data set is relatively large, it was also crucial to sample and use the ratio of data available in the minority and majority classes. For example, if after re-balancing, the average number of available rows was still around 184,186, then we would be utilized around 1.90% of the original data set(s).

Table 3.6: Down Sampled Data (2020)

Date	Longitude	Latitude	EVI	TA	LST	Wind	Fire	Elevation
2020-07-12	-122.900995	41.790854	4008	5	15113	45	1	951.315152
2020-10-23	-119.457322	38.558061	1040	5	14761	9	0	1725.509091
2020-09-04	-120.768999	39.020483	4700	5	15388	21	1	888.775758
2020-07-23	-122.834370	41.861675	2429	5	15850	30	1	684.460606
2020-09-21	-120.302625	41.882505	1843	5	15452	24	1	1599.169697
2020-07-06	-122.851027	41.886671	3024	5	15253	45	1	1163.303030
2020-07-25	-123.171659	41.032647	3976	5	15168	3	0	1405.327273
2020-08-25	-120.814804	38.987156	4104	5	15103	16	1	655.848485
2020-07-10	-122.888503	41.782522	3661	5	15086	30	1	1001.593939
2020-04-30	-123.409010	41.124299	3411	5	14473	43	0	1384.478788

3.3.3 Feature Scaling

Since the data set contains data that is not properly scaled, it is important to apply some normalization or scale to the data. This is useful to make sure that all data can be bounded on a common scale. Furthermore, this project focuses on using the standard scalar found within sklearn [19], since features such as: longitude, latitude, EVI, TA, and wind can often vary in their respective ranges. It can also help our ensemble model by allowing: SVM to find a more accurate decision boundary, GNB to predict based on data that is closely centered around 0, and KNN to see each feature equally instead of assigning a bias. IN addition, the standard scale will bound values between -3 to 3 and is effective with distance-based models, in our case SVM and KNN.

Listing 3.3 depicts a sample script of how the formerly mentioned information and ideas would be accomplished. Table 3.7 shows the results of applying the standard scaler to all data set features, excluding the 'Date' and 'Fire' column, since 'Date' is not numeric, and the 'Fire' column is already scaled and is our target variable.

Listing 3.3: Example Feature Scaling

```
# Train-test split
#Since we have large datasets, a 90/10 split works well
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,
    random_state=1)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Table 3.7: Sample Scaled Data Table

Date	Longitude	Latitude	EVI	TA	LST	Wind	Elevation	Fire
2020-07-12	-0.915430	1.072927	1.289162	0.024163	0.097370	1.480273	-0.112217	1
2020-07-12	0.848820	-0.193400	-1.668167	0.024163	-0.491626	-1.230013	1.374322	0
2020-07-12	0.176826	-0.012263	1.978674	0.024163	0.557523	-0.326585	-0.232299	1
2020-07-12	-0.881297	1.100669	-0.284161	0.024163	1.330581	0.350987	-0.624607	1
2020-07-12	0.415757	1.108828	-0.868054	0.024163	0.664614	-0.100727	1.131736	1
2020-07-12	-0.889831	1.110460	0.308700	0.024163	0.331630	1.480273	0.294824	1
2020-07-12	-1.054096	0.775928	1.257277	0.024163	0.189401	-1.681728	0.759537	0
2020-07-12	0.153360	-0.025318	1.384817	0.024163	0.080637	-0.703013	-0.679546	1
2020-07-12	-0.909030	1.069663	0.943410	0.024163	0.052191	0.350987	-0.015676	1
2020-07-12	-1.175695	0.811829	0.694309	0.024163	-0.973532	1.329701	0.719506	0

3.4 Training Approach And Testing using Ensemble Model

Figure 3.2 depicts the overall training and testing flow. Processed and batched data will be fed into a 3 model ensemble that includes SVM, GNB, and KNN, and it will predict outcomes based on a majority voting strategy. Each model will contribute to its prediction. It should be noted that in instances where an even number of models are used, when the models predict in an exactly equal amount, 50% of the models predict one classification while 50% predict a different classification, a default decision needs to be made. Furthermore, in a 2 model ensemble this reduces the need for an ensemble completely, since the default choice will likely be a default model, or the model that is more likely to predict more accurately.

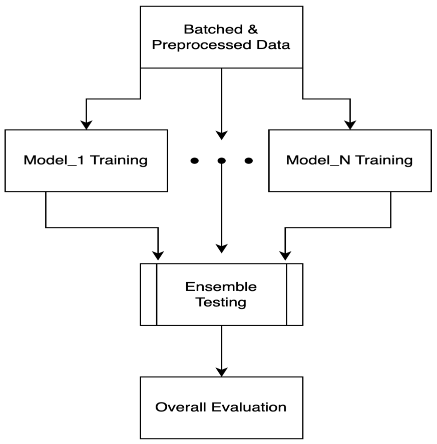


Figure 3.2: Training & Testing Flow

Approach Overview:

1. Each model receives the same of batched and processed data.
2. Each model will be individually trained.

3. Each trained model will be part of the ensemble that occurs during the testing/validation portion.

4. Each result derived comes from the ensemble model.

Since this project focused on the usage of 3 machine learning models, the ensemble will combine these predictions based on a majority voting strategy to determine the final classification outcome. This technique leverages the strengths of each classifier, providing a more robust and balanced performance across varied data patterns.

SVM can be noted as a supervised learning model used for classification and regression tasks. It works by finding the optimal hyperplane that separates data points of different classes with the maximum margin [20]. For this project, SVM supports finding the separation between fire-prone and non fire-prone areas.

KNN can be described as: a non-parametric, supervised learning method used for classification and regression. It classifies data points based on the closest training examples, k [21]. For this project, KNN is suitable because it can adapt changes or variations in the data, since wildfire data can have various changes when a fire occurs.

GNB is a supervised learning algorithm that is used for classification. It applies Bayes' Theorem with the assumption of feature independence and models continuous features as Gaussian distributions, making it suitable for many real-world datasets [22]. GNB is essential to this project, since it runs extremely quickly and handles imbalanced data quite well.

Listing 3.4: Training Script

```
# Training function for SVM, KNN, and GNB
def train_gpu_models_online(dataloader):
    svm_clf = cuSVM(kernel='linear', probability=True)
    gnb_clf = cuGNB()

    # Initialize FAISS GPU KNN
    d = next(iter(dataloader))[0].shape[1]
    knn_index = faiss.IndexFlatL2(d)
    res = faiss.StandardGpuResources()
    knn_index = faiss.index_cpu_to_gpu(res, 0, knn_index)

    # Training loop using DataLoader for mini-batching
    for batch_idx, (X_batch, y_batch) in enumerate(dataloader):
        X_batch = X_batch.numpy().astype('float32')
        y_batch = np.array(y_batch)

        # cuML does not support true online learning so this is a work around
        # If we don't do this then on each batch the models may reset, which we want to avoid
        if batch_idx == 0:
            # Initial fit with the first batch of SVM and GNB
            svm_clf.fit(X_batch, y_batch)
            gnb_clf.fit(X_batch, y_batch)
        else:
            # Otherwise re-fit on cumulative data for SVM and GNB
            svm_clf.fit(np.vstack([X[:batch_idx * len(X_batch)], X_batch]),
                        np.hstack([y[:batch_idx * len(y_batch)], y_batch]))
            gnb_clf.fit(np.vstack([X[:batch_idx * len(X_batch)], X_batch]),
                        np.hstack([y[:batch_idx * len(y_batch)], y_batch]))

        # Update FAISS index approximate batch learning
        if batch_idx % 5 == 0:
            knn_index.reset()
            knn_index.add(np.vstack([X[:batch_idx * len(X_batch)], X_batch]))

    # joblib to save SVM and GNB models
    joblib.dump(svm_clf, 'svm_model.joblib')
    joblib.dump(gnb_clf, 'gnb_model.joblib')
    #faiss library way of saving model
    faiss.write_index(knn_index, "knn_model.index")
```

Listing 3.4 demonstrates the flow of testing all three models on a given set of data. It should be noted that due to the nature of needing to utilize the GPU, many of the standard approaches of batched training were unavailable. As a result, there was a workaround that needed to be used so that each model would retain information from the previous batch trained on; otherwise the model would completely reset for each new batch, which would eliminate the benefits of batched training. The listing also demonstrates how each model, after being trained, will be saved, so that this process does not need to be executed once more in the testing portion.

Chapter 4

Methods

4.1 Mini Batching & Multiprocessing

Mini-batching involves breaking down the data set into smaller subsets, or batches, to facilitate iterative processing in machine learning algorithms. It is effective in scenarios where memory constraints or high data volume make it challenging to process the entire dataset in one complete pass. Moreover, the batches can be fed into an ensemble model, enabling the model to learn from subsets of the data iteratively. This approach enhances training efficiency and scalability, especially for large datasets, by optimizing computational resources and reducing memory requirements.

The idea of multiprocessing can be thought of as using multiple compute cores to run compute tasks in parallel. This provides much needed practicality since complex models and large datasets typically require longer amounts of training time. Using multiple GPU cores, we could compute results faster and reduce overall computation time. For example, using the Pytorch library, with GPU support enabled, one could make use of the 'Dataloader' class [23]. The Dataloader class provides an efficient way to load data in mini-batches, shuffle it, and optionally perform operations like data augmentation or parallel processing. Since this approach distributes the workload across multiple processes, enhancing performance and scalability by leveraging available computational resources efficiently. Overall, training time can be lowered.

The Python language also has a global interpreter lock (GIL) that allows only one thread to access the interpreter; however, since we do not need to directly invoke various python scripts, the DataLoader class will spawn a specified number of processes to help us retrieve and load data faster and more efficiently. In order to find a "sweet spot" for the size of batches to use, a number of small tests were run, and ultimately, for the configuration depicted, batch sizes of 10,000 were found to be the most optimal.

Listing 4.1 shows how the data set will be broken down into smaller batches, for example, in this case 10,000 in size, reducing memory usage and improving computational efficiency. By processing these batches sequentially, tasks such as model training and evaluation become more manageable and scalable, making it feasible to analyze each year of data. In addition, each N number of workers or processes is spawned by the DataLoader class. More specifically, each of these workers will transport a given batch to the main process that is training the model.

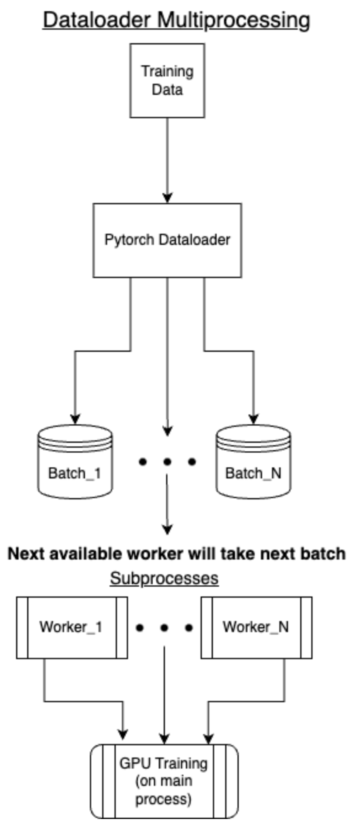

Dataloader Multiprocessing
Training Data
Pytorch Dataloader
Batch_1
Batch_N
Next available worker will take next batch
Subprocesses
Worker_1
Worker_N
GPU Training (on main process)

Figure 4.1: Design for Multiprocessing and Mini-batch Scheme.

As depicted in Figure 4.1, worker processes are started and queued to receive the next available batch of data. These batches are formed as subsets or mini-batches of the available training data. In unison, this allows the primary process to train a given model on the GPU while the CPU is used in parallel to partition and retrieve the data.

Listing 4.1: Example Mini Batching And Multiprocessing

```
from torch.utils.data import DataLoader
# Samples per batch, which will be our 'mini batches'
batch_size = 10000

# num_workers will be the amount of subprocess to spawn and share the resources.
# This is how we will utilize multiprocessing. Here we spawn 6 subprocesses.
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
    num_workers=6)
test_loader = DataLoader(test_dataset, batch_size=batch_size, num_workers=6)
```

4.2 Ensemble Methods

Ensemble methods combine the predictions of multiple individual models to produce a stronger overall prediction. They leverage the diversity of these models, often trained on different subsets of the data or using different algorithms, to mitigate biases and improve robustness. In the context of the data sets that we will use, the ensemble methods would aggregate the predictions of the models trained on each dataset, harnessing the collective intelligence derived from various temporal patterns and variations throughout the year to improve the predictive precision and stability of the final model.

The reason behind the selection of KNN, GNB, and SVM is due to the positive and high results achieved by previous researchers, when using all models individually in a similar data set and since all models work well for classification tasks [3] [4]. All models, respectively, handle a large amount of data well depending on the configuration selected. From the initial tests conducted, it was found that a linear kernel and K = 5 were the most optimal for the given data.

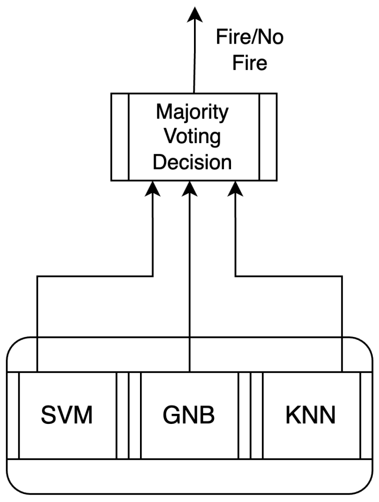


Figure 4.2: Majority Voting Decision

Listing 4.2: Ensemble Model

```
import numpy as np
import pandas as pd
import faiss
from cuml.svm import SVC as cuSVM
from cuml.naive_bayes import GaussianNB as cuGNB
from cuml.neighbors import KNeighborsClassifier as cuKNN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, precision_score, confusion_matrix
import joblib

def train_ensemble_model(csv_file):
    # Assume csv file is read and preprocessed already and in data
    y = data['Fire'].astype('int32')
    X_scaled = data.drop(columns['Fire']).astype('float32')

    # Load pre-trained models from memory, since
    # training and testing are seperate steps
    svm_clf = joblib.load('svm_model.joblib')
    gnb_clf = joblib.load('gnb_model.joblib')
    knn_clf = faiss.read_index("knn_model.index")

    # Predictions from each model
    svm_pred = svm_clf.predict(X_scaled).astype(int)
    gnb_pred = gnb_clf.predict(X_scaled).astype(int)
    knn_pred = knn_clf.predict(X_scaled).astype(int)

    # Majority voting for ensemble prediction
    # Notice that zip will create a 3-tuple for each corresponding model
    # vote on each datapoint from each model
    ensemble_pred = np.array([np.bincount([svm, gnb, knn]).argmax() for svm, gnb, knn
    in zip(svm_pred, gnb_pred, knn_pred)])

    # Calculate metrics
    accuracy = accuracy_score(y, ensemble_pred)
    f1 = f1_score(y, ensemble_pred)
    precision = precision_score(y, ensemble_pred)
    conf_matrix = confusion_matrix(y, ensemble_pred)

train_ensemble_model('wildfire_20xx.csv')
```

Figure 4.2 illustrates the 3-model ensemble, and how the overall model's result is a product of each individual model's result being combine via majority voting. Listing 4.2 serves to show how the 3-model ensemble is evaluated once each model has been individually trained. It should be noted that for each data point that is predicted by a model, the data point will have a 3-tuple returned. For example, for a given data point the return value could be (0,1,1), indicating that 2 models predicted there was a fire while 1 predicted that there was no fire. The aspect of majority voting is utilized here since the result with the highest count will be become the overall model result. So for the case of (0,1,1), the model would output 1. As a result, the model's overall metrics are derived from the overall model's predictions based on the majority voting result.

Chapter 5

Results

5.1 Testbed

The final work encompasses the usage of two distinct machines (as shown in Table 5.1): (1) A conventional desktop computer equipped with an Intel i5 9600K processor boasting 6 cores operating at 4.7 GHz, coupled with 16GB of RAM rated at around 3200 MHz, and an NVIDIA 2070 graphics processing unit (GPU), (2) A compute server with an Intel Xeon E5-2690 processor with 98 GB of RAM and featuring an NVIDIA Tesla P4 GPU.

Table 5.1: Systems Used

OS	Processor	Memory	Other
Windows	Intel i5 9600k (6 cores, 4.7 GHz)	16 GB RAM	NVIDA 2070 GPU
Ubuntu	Intel Xeon E5-2690 (20 Cores, 3.00GHz)	98 GB RAM	NVIDA Tesla P4 GPU

Table 5.2: Software Tools Used in the Project

Software	Description
Python 3.10	Main programming language used for development
CUDA Toolkit	GPU computing platform and API for accelerating calculations
RAPIDS	GPU-accelerated libraries for data science, utilized for SVM, GNB, and KNN
Pandas	Data manipulation and analysis library
NumPy	Numerical computation library for handling arrays and matrices
Scikit-learn	Machine learning library for model training and evaluation
Joblib	Library for loading and saving machine learning models
PyTorch	Deep learning framework, used for model training and testing
Faiss	Efficient similarity search and clustering of dense vectors library
Anaconda	Package and environment management tool

Using GPUs on the desktop and compute server was imperative in enabling the experiments described in this study. Both systems shared similar development environments, including Python 3.10 [16], CUDA 12.1 Toolkit [24] (though running different versions), RAPIDS for GPU acceleration [25] that allows out of the box utilization for SVM and GNB, and common Python libraries such as: Pandas [15], NumPy [26], Scikit-learn, Faiss [27], Joblib [28], and PyTorch [23]. Table 5.2 lists the software tools used in this project.

The environments were orchestrated via Anaconda [29] to ensure consistency and ease of portability. Listing 5.1 depicts an example of utilizing Conda to create an environment with a specific Python, Pytorch, and Rapids version.

Listing 5.1: Conda environment using RapidsAI

```
conda create -n rapids-24.10 -c rapidsai -c conda-forge -c nvidia  \
    rapids=24.10 python=3.10 'cuda-version>=12.0,<=12.5' \
    'pytorch=*=*cuda*'
```

5.2 Machine Learning Model Measurement

In order ensure transparency and adhere to standard guidelines for evaluating machine learning models, multiple metrics were used to assess the results of this project. Table 5.3 presents the confusion matrix, a common tool for visualizing the performance of classification models.

Table 5.3: Confusion Matrix

	Predicted Negative	**Predicted Positive**
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Furthermore, the evaluation metrics are calculated as follows:

$$\text{Precision} = \frac{N_{\text{TP}}}{N_{\text{TP}} + N_{\text{FP}}} \tag{5.1}$$

$$\text{Recall} = \frac{N_{\text{TP}}}{N_{\text{TP}} + N_{\text{FN}}} \tag{5.2}$$

$$\text{F1-score} = \frac{2 \times (\text{Recall} \times \text{Precision})}{\text{Recall} + \text{Precision}} \tag{5.3}$$

$$\text{Accuracy} = \frac{N_{\text{TP}} + N_{\text{TN}}}{N_{\text{TP}} + N_{\text{FP}} + N_{\text{TN}} + N_{\text{FN}}} \tag{5.4}$$

5.3 Experimental Results

Table 5.4: Setting for Training Approaches

Approach #	**Mini batch & Multiprocessing**	**Ensemble**
1	No	No
2	Yes	No
3	Yes	Yes

We designed 3 sets of experiments (as shown in Table. 5.4) to evaluate different approach for model training: (1) train the ML models using the data from any given year(s) directly; (2) Employ both mini batching and GPU multiprocessing on scaled and balanced data; (3) Use our proposed approach to train on all scaled and balanced data. For each testing approach, we utilized 2 different sets of data to repeat the experiments: (1) set 1 used data from the year 2020 since it is one of the largest datasets available; (2) set 2 was conducted on the full 10 years worth of data. Further, all 3 approaches are compared in terms of metrics and time spent during the training phase.

5.3.1 Approach 1: Base Approach

Table 5.5 shows the training results and running time by applying the base approach in the data set for the year 2020. In addition, the only data pre-processing that was done was limited to feature scaling, so the data was majority imbalanced. The runtime for SVM is drastically longer than that of KNN and GNB and GNB can be seen as having the fastest run time.

Table 5.5: Training Results for Approach 1 (2020)

Machine	ML Model	Runtime (rounded to min)	Accuracy	Recall for class 1
Windows (Intel i5)	SVM	1,152	97%	0.1%
	KNN	70	96%	0%
	GNB	60	80%	0%
Ubuntu Server	SVM	1,064	97%	0.1%
	KNN	64	96%	0%
	GNB	56	80%	0%

Figure 5.1 shows the confusion matrices and measurement values generated from running the above configuration on the windows machine. It can be seen that since both models had an extremely low recall for the minority classes and predicted most true negative, which are a part of the majority class. Moreover, the high accuracy shown does not translate to meaningful results for the sought after class, the minority class in this case. Due to this, the confusion matrix, precision, recall, and F1-score measurements are 0 and omitted from presentation.

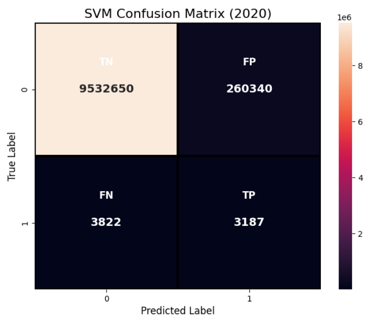
Table 5.6: Estimated Runtime for Approach 1 (2013-2022)

Machine	ML Model	Runtime Estimate (Days)
Windows (Intel i5)	SVM	8
	KNN	0.481
	GNB	0.422
Ubuntu Server	SVM	7.391
	KNN	0.442
	GNB	0.393

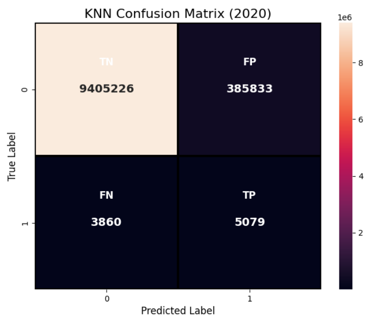
Table 5.6, depicts an estimate of what using a decades worth of data, with the lack of usage of any described method. It should be noted that since there is such a high run time for the given years of data, it is much too massive to be able to derive meaningful results, and therefore it is only estimated what the runtime would be.

5.3.2 Approach 2: Mini batching and Multiprocessing

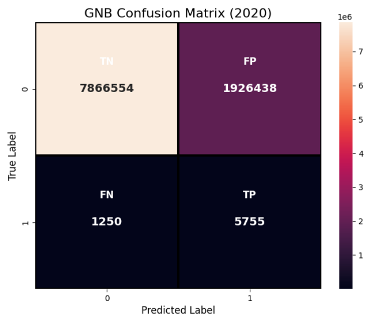
Shown in Figure 5.1, the confusion matrices for all 3 models are presented. Notice that while the count of TN, FN, TP, and FP seem to be greater and trending towards measuring the models in a positive manner, the values still leave much room for improvement.



(a) SVM Confusion Matrix



(b) KNN Confusion Matrix



(c) GNB Confusion Matrix

Figure 5.1: Confusion Matrices for Approach 2 (2020 Dataset)

Table 5.7: Training Results for Approach 2 (2020)

Machine	**ML Model**	**Runtime (min)**	**Accuracy**
Windows (Intel i5)	SVM	5.012	97%
	KNN	3.122	96%
	GNB	2.962	80%
Ubuntu Server	SVM	4.913	97%
	KNN	2.852	96%
	GNB	2.643	80%

Table 5.8: Training Metrics for Approach 2 (2020)

ML Model	**Precision**	**Recall**	**F1-Score**
SVM	0.0121	0.4547	0.0236
KNN	0.0130	0.5682	0.0254
GNB	0.0030	0.8215	0.0059

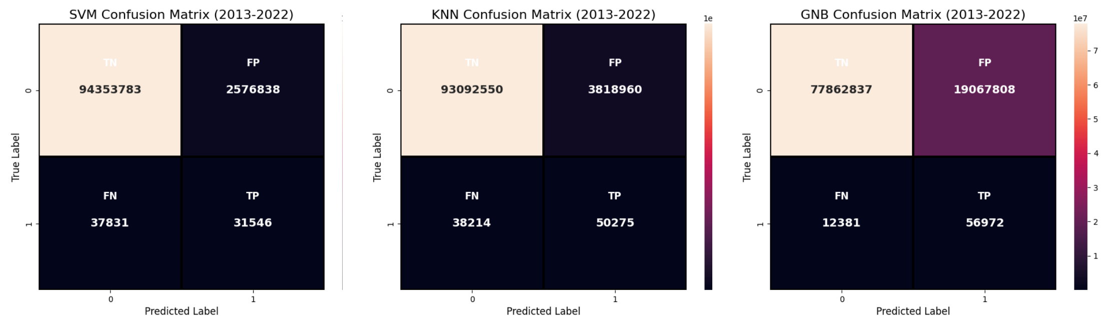
Depicted in Table 5.7, one can see the results for both individually run machine learning models. From a quick glance, the benefit of utilizing the GPU can be quickly seen and understood. And, again, it can be noted that the ranking in terms of run time from highest to lowest was: SVM, KNN, and GNB. Furthermore, table 5.7 shows that applying the mini batching and multi-processing keeps the metrics relatively similar while lowering the run time. Table 5.8 demonstrates the metrics for the approach 2 on the year 2020. It can be seen that the scores are relatively low for the model, which is most likely a result of the patterns lost when balancing the data.

Table 5.9: Training Results for Approach 2 (2013-2022)

Machine	ML Model	Runtime (min)	Accuracy
Windows (Intel i5)	SVM	70.134	97%
	KNN	35.231	96%
	GNB	31.230	84%
Ubuntu Server	SVM	68.345	97%
	KNN	32.321	96%
	GNB	28.142	84%

Table 5.10: Training Metrics for Approach 2 (2013-2022)

ML Model	Precision	Recall	F1-Score
SVM	0.0121	0.4547	0.0236
KNN	0.0130	0.5682	0.0254
GNB	0.0030	0.8215	0.0059



(a) SVM Confusion Matrix (2013-2022) (b) KNN Confusion Matrix (2013-2022) (c) GNB Confusion Matrix (2013-2022)

Figure 5.2: Confusion Matrices Approach 2 (2013-2022)

Table 5.9, shows that even on a large scale, in this case using all datasets, the results do not take much time to produce. Moreover, there is no additional loss of accuracy. In addition, Table 5.10 demonstrates the metrics for the given approach and for the given range of data, 2013-2022. Again, it can be seen that the metrics are not negatively impacted. Figure 5.6 also shows the confusion matrices, and it can be noted that the GNB has the highest recall for fire prone instances, while SVM and KNN miss quite number of fire prone instances, which can be noted by their recall scores.

5.3.3 Approach 3: Mini batching, Multiprocessing, and Ensemble method

Table 5.11: Training Results for Approach 3 (2020)

Machine	**Runtime (min)**	**Accuracy**
Windows (Intel i5)	15.425	96%
Ubuntu Server	14.254	96%

Table 5.12: Training Metrics for Approach 3 (2020)

ML Model	**Precision**	**Recall**	**F1-Score**
Ensemble	0.0119	0.45	0.0231

Tables 5.11 and 5.12 demonstrate the results of all three approaches applied on a single year of data, 2020. It can be noted that although the runtime is increased as opposed to running any single model, the difference in time is not significant as compared to running a single model using approach 1. Additionally, the accuracy remains fairly similar to the original model, while improving the model's overall metrics. This is most likely due to the effective utilization of majority voting in the ensemble model.

If the same experiment were to be run on only the CPU, it should be noted that the run time would be extremely large and the trade-off would not be worth the ensemble of 3 models. In addition, these methods appear to work the best when used in tandem with the GPU.

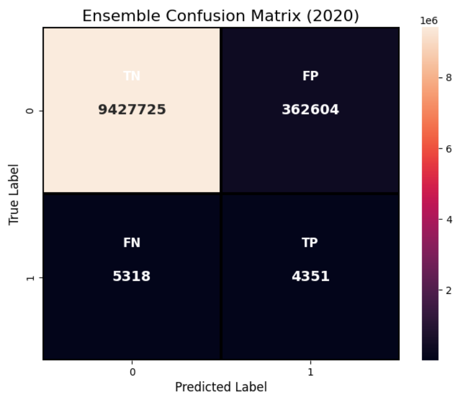


Figure 5.3: Ensemble Model Matrix (2020)

Figure 5.3, depicts the confusion matrix. It can be seen that, when in an ensemble, the model's metrics are overall greater than when any single model is run on its own.

Table 5.13: Training Results for Approach 3 (2013-2022)

Machine	**Runtime (min)**	**Accuracy**
Windows (Intel i5)	150.000	98.1%
Ubuntu Server	146.134	98.1%

Table 5.14: Training Metrics for Approach 3 (2013-2022)

ML Model	**Precision**	**Recall**	**F1-Score**
Ensemble	0.0263	0.5	0.05

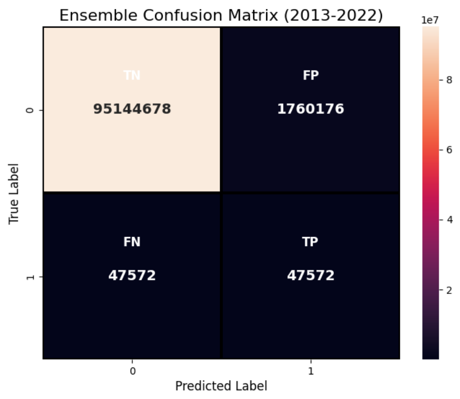


Figure 5.4: Ensemble Model Matrix (2013-2022)

Finally, after applying all methods described on all years of data, table 5.13 displays the results achieved. The run time of this approach is substantially faster than that of Approach 1, although the computation is more extensive. Also, notice that the accuracy for the model is higher than any individually ran model, meaning that the ensemble did indeed help create a rise in accuracy.

Table 5.14 and figure 5.4, depict metrics and the confusion matrix. It can be seen that, when in an ensemble, the model's accuracy is overall greater than when any single model is run on its own; however, the metrics related to F1-score, recall, and precision are still relatively low. This may be due to the model's ability to correctly predict non-fire cases, but conversely it does not do as well for the fire cases. In addition, since each model may independently make errors on a given data point, if more than 1 model makes that same error, then the ensemble will suffer from making that same mistake. Furthermore, the ensemble captures more positives use cases, but it does not necessarily reduce similar errors across individual models, leading to excessive false positives

5.4 Analysis

5.4.1 Comparisons of results

The setups range from a Windows Desktop with i5 9600 to an Ubuntu server. It quickly becomes apparent that the hardware exerted a significant influence on runtime. Most notably, the Support Vector Machine model demonstrated the longest runtime across these platforms, highlighting the pivotal role of hardware power in achieving computational efficiency. However, it should also be noted that while the hardware played a pivotal role, software schemes and applicable usage of a GPU made the largest difference. Moreover, if the results achieved with the more mundane or common hardware could be applicable, it would produce a greater impact Combining mini-batching with multiprocessing significantly improved how fast the program could run and how well it could handle large datasets. These techniques made the program run much faster without hurting its accuracy, which is important for working with large datasets and is a key goal of this research.

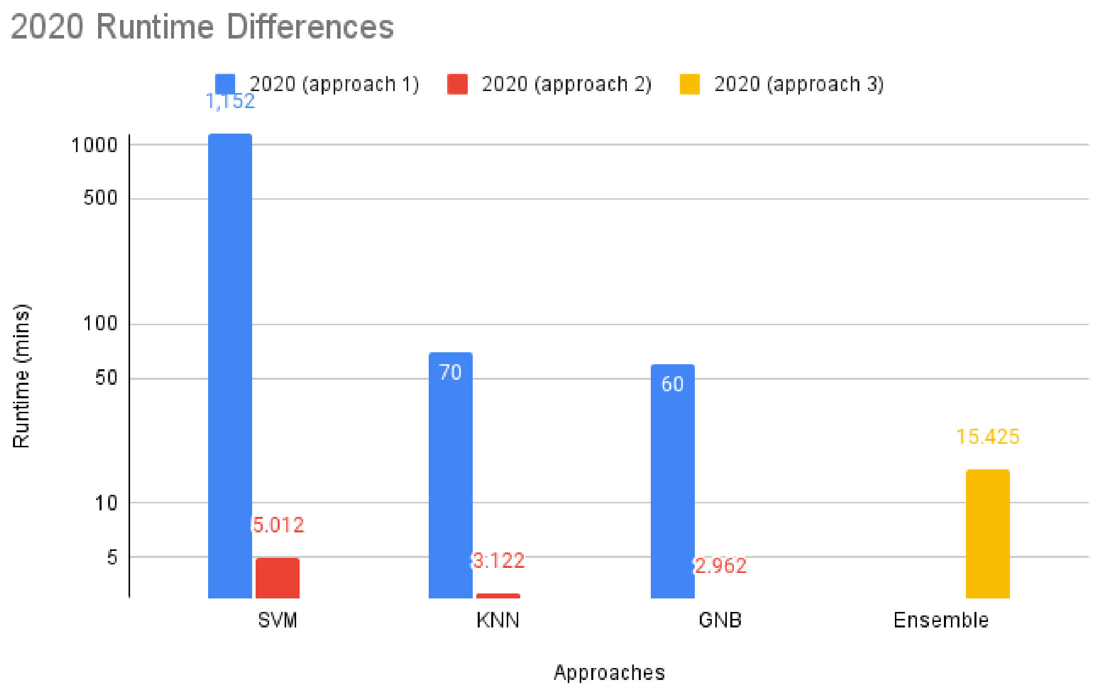


Figure 5.5: Runtime Chart on 2020

In the first figure 5.5, SVM (approach 1) has the highest run time at 1,152 minutes, far exceeding all other models and approaches. This could be seen as most likely being a result of the complexity of the model. In comparison, SVM (approach 2) and approach 3 are much faster, with training run times of 5.012 minutes and 70.134 minutes, respectively. For KNN, the runtime drops significantly from 70 minutes in approach 1 to 3.122 minutes in approach 2. GNB also shows a similar reduction, going from 60 minutes in approach 1 to 2.962 minutes in approach 2. The ensemble model, has a runtime of 15.425 minutes, increasing the speed than approach 1.

Figure 5.6 illustrates that approaches 2 and 3 have shorter run times of 70.134 minutes and 35.231 minutes, respectively, compared to the longest runtime in approach 1. For KNN, runtime grows from 3.122 minutes (2020) to 35.231 minutes with the larger dataset.

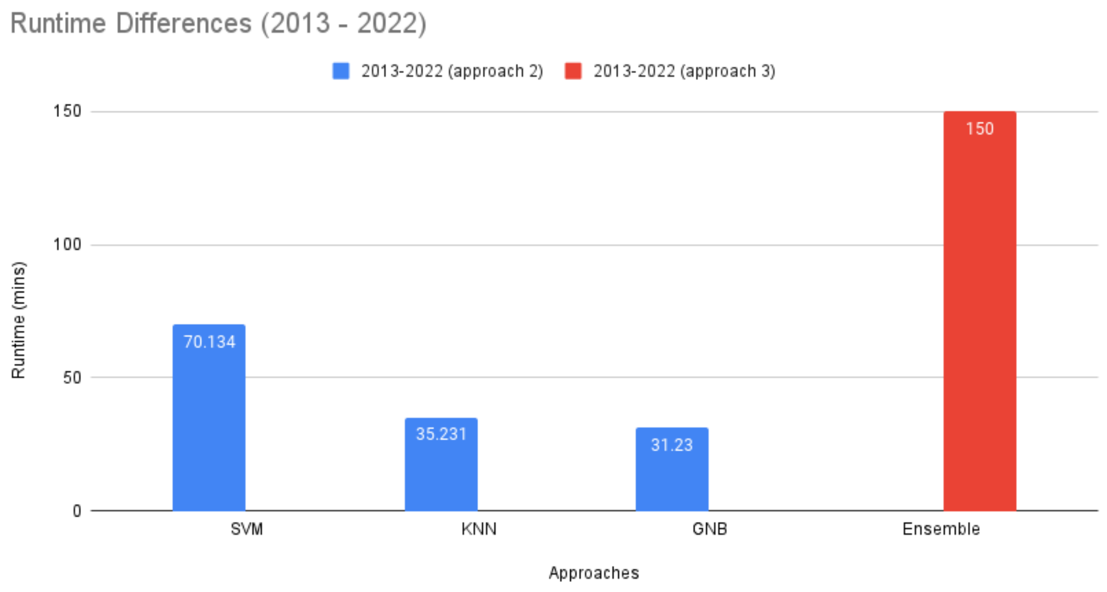


Figure 5.6: Runtime measurements (2013-2022)

Similarly, GNB increases from 2.962 minutes (2020) to 31.23 minutes (2013–2022). The ensemble model also shows a significant increase, rising from 15.425 minutes (2020) to 150 minutes with the larger dataset, highlighting the additional computational cost of processing more data.

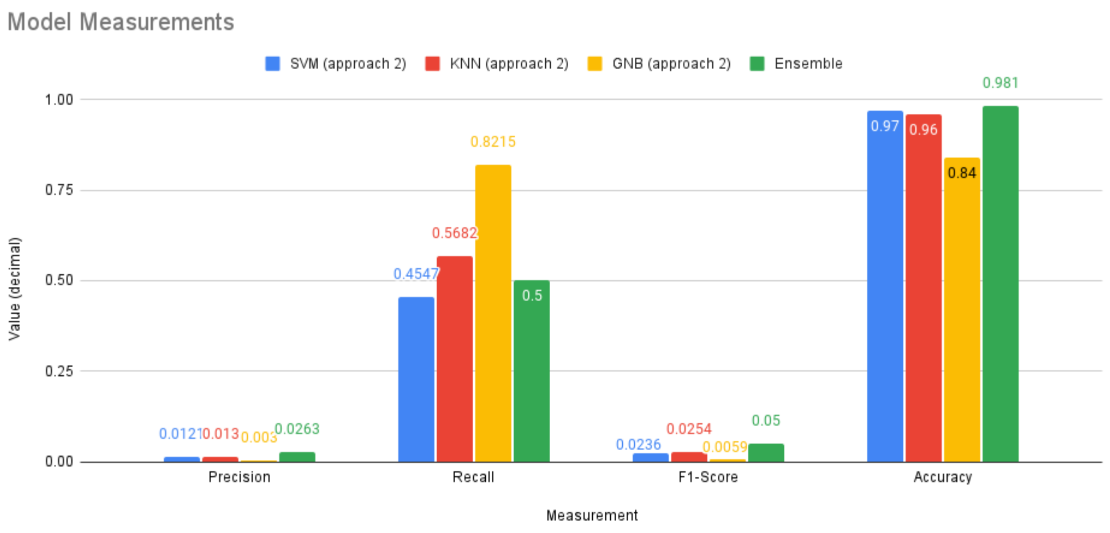


Figure 5.7: Model Metrics (2013-2022)

Figure 5.7 compares the performance of four models: SVM, KNN, GNB, and the ensemble, based on precision, recall, F1 score, and accuracy. The precision is notably low across all models, with SVM at 0.0121, KNN at 0.013, GNB at 0.005, and the ensemble slightly higher at 0.0263. This suggests that while accuracy is high on all models, the models struggle to correctly classify positive cases. This can be likely inferred as a direct result of the severe class imbalance of the dataset.

Moreover, Approach 3 outperforms Approaches 1 and 2 by providing a better balance between precision and recall. While its precision of 0.0263 remains low, it surpasses SVM (0.0121), KNN (0.0130), and GNB (0.0030) in approach 2, indicating fewer false positives. Although GNB achieves the highest recall (0.8215) in approach 2, its extremely low precision results in excessive false positives, while the recall of approach 3 of 0.5, coupled with a higher F1 score (0.05 versus 0.0236, 0.0254, and 0.0059), reflects a more balanced and reliable performance.

Furthermore, based on the training metrics for Approach 3 from 2013 to 2022, the model shows a precision of 0.0263, a recall of 0.5, and an F1-score of 0.05. Although a recall of 50% suggests that the model can capture half of the true positive cases, the very low precision indicates that only 2.63% of its positive predictions are actually correct. This high rate of false positives significantly decreases the effectiveness of the model, as it frequently misclassified non-target fire instances as positives. The F1 score of 0.05, which balances precision and recall, highlights the overall challenge that this model faces in providing reliable predictions. As a future work, it could be advisable to spend more time on the model selection as a more diverse selection of models may help drive the relevant metrics upwards.

SVM achieves 0.97, KNN scores 0.96, GNB 0.84, and the ensemble leads with 0.981. Although these figures suggest strong performance, they likely reflect a bias on the majority class. Again, this could be due to the nature of the collected data and how the likelihood of a fire occurring is actually an anomaly throughout the year.

As such, it would be advisable to test these processes on other systems to obtain more consistent and well-rounded results, as well as experiment to see what capacity the batch or process size can be adjusted, since every machine may benefit from a different configuration. In addition, the individuals models may further benefit from hyper parameters adjustments.

5.4.2 Implications and Conclusion

The preliminary results presented in the tables above, demonstrate a framework for further analysis and optimization. 10 years worth of data was observed and considered, with an aim to provide good accuracy and a reasonable run time; however, extending this past that point would be an excellent area for further research. Further, while there are still issues or areas of concern: such as the still high runtime and low recall score of the model, those concerns can be eliminated with continued work on the collection and refinement of the applicable data.

Moreover, the highlighted experiments demonstrated the feasibility of utilizing ensemble methods, with majority voting, to enhance model performance. And although it can be seen that there is an increased in the overall runtime, the model did not lose accuracy compared to its original counterpart. As the work is further optimized it could be noted that runtime can be reduced by reducing the number of models being put in an ensemble approach. That is to say, instead of having 3 models put together, we can select the top 2 best performing models. This is a key insight that was noticed, and can be further looked into in additional studies. In contrast, it should also be noted that while all models performed relatively well, there could perhaps be a better combination of models that, when put into an ensemble, would yield even greater results.

Through the preliminary results presented, it is evident that the scalability of the proposed methodologies is plausible and consistent. It can be seen by the varying durations, ranging from one year of data to ten year accumulations. Despite the computational challenges posed by a longer run-time, the results suggest that this is a viable strategy to scale. It is advisable that, the use of these methods can be drastically scaled in regards to implementation and maximum use of computational resources, further proving that these combined approaches provide an efficient and attainable way to train and test.

5.4.3 Refinements to be considered

After an analysis of the data, it can be seen that there are various levels of refinement or optimization that can be done. The most apparent are: (1) improving the recall score of each model individually before furthering the experiment, (2) considering trying other models or increasing the number in the ensemble model, (3) considering utilizing more compute resources to train and test, (4) further exploration of the data set to see if a greater amount of redundant data can be combined or discarded, (5) exploration of the usage of Apache PySpark to employ clustering, (6) refining the data collection in order to see if the class imbalance can be addressed at the data preparation stage, and (7) Exploring real time prediction using real time streams of data.

By improving the recall score of each model before moving into the ensemble and testing phase, it would be beneficial to further test and modify the parameters of each model so that individually each model outputs its most optimal performance. This would also help in deciding whether or not a model is a good candidate to be in the ensemble. Furthermore, if more powerful compute options are available, then greater utilization of those resources could also result in decreased runtime. This could also allow a greater emphasis on the pre-processing steps or the gathering new data, which in turn could result in higher quality data. Continuing, having higher quality data would benefit the model's metrics since the imbalance may be better addressed, and there may not be a need for such large quantities of data. Since there is such a large amount of data and processing, training, and testing take time, our classification problem may be a good candidate for the usage of clustering or a framework like PySpark since they enable large-scale data Finally, by considering real-time data, the applicability of framework could be further tested.

Chapter 6

Conclusion

As this research draws to a close, there is key insight in regards to the impact and effectiveness of this project, being able to take advantage of the GPU and existing software or libraries. Without the usage the usage of GPU, the time taken to conduct such experiments would still be measurable; however, the time taken would still be far too long to demonstrate meaningful usage or practicality. Being able to utilize the GPU meant that effective multiprocessing and batching could be done, without the cost of expensive overhead or complex architecture. The Python ecosystem is mature and has been developed and advanced to a point where much of this research ideas are readily available in many Python libraries. Furthermore, existing software and libraries within the Python ecosystem allowed a speedy, efficient, and modular implementation that required minimal custom implementations.

Since the notion of being able to depict wildfires desires a low runtime with high amounts of accuracy, especially if the data is being observed with little time to spare; efforts and methodologies used within this project demonstrated excellent results in which we lowered the runtime by 50% while simultaneously increasing accuracy up to 98%. Likewise, the results derived from the combined usage of mini-batching, multiprocessing, and a 3 model ensemble also proved to show that the practicability of such a design could be reproduced and extended to other types of experiments. It is expected that these results could be extended to data that ranges far longer than 10 years or for uses cases where similar types of classification problems must be analyzed, solved, and predicted within reasonable run times.

Overall, the project accomplished it's contributions of providing concrete results with the mentioned methods, lowering training time, reaching an accuracy of 98% in the 3 model ensemble, and creating and modularizing a framework that can be extensible.

In terms of future work on this research, there a few things that immediately come to mind: extending this research from 10 years to anything beyond that, further refining the existing data such that redundant data or unnecessary data can filtered out in a pre-processing layer, exploring and seeing the impact of more than 3 models in the ensemble model, and finally exploring the utilization of this using a machine that has the top of the line specifications.

To further elaborate, it would be excellent to see these ideas extending for datasets that range from 20 - 30 years worth of data, and so on. The dataset used did indeed pose the issue of being extremely large and having a large amount of unnecessary or redundant data, therefore, if the data could be further cleaned before being used to train and evaluate, that could also provide a reduction in runtime. This would be immensely beneficial if the pre-processing work could be saved into the original dataset, so that once the data is loaded it can directly be used to train and test. Moreover, testing the usage of more than 3 models could provide a greater accuracy, at the cost of runtime, but may still be worthwhile, especially considering that there are various models that may have a similar runtime to our lowest model GNB. Further, hyper parameter tuning would also be an excellent addition. The experiments conducted in this research were executed on older machines, so a top of the line machine with the newest and greatest hardware specifications may be able to further reduce the runtime without having to change much of the configuration. Finally, exploring the utilization of a real time data stream, may allow for dynamic predictions as well as allow stronger identification patterns to be taken into account. As a result, it may also allow for iterative improvements on each model's performance.

References

[1] CalFire, “Current emergency incidents.” [Online]. Available: https://www.fire.ca.gov/incidents

[2] CCST, “The costs of wildfire in california overview.” [Online]. Available: https://ccst.us/reports/the-costs-of-wildfire-in-california/

[3] K. Pham, D. Ward, S. Rubio, D. Shin, L. Zlotikman, S. Ramirez, T. Poplawski, and X. Jiang, “California wildfire prediction using machine learning,” in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022.

[4] D. J. Castrejon, C. Wang, D. Osmak, B. Kukadiya, L. Liu, M. Giraldo, and X. Jiang, “Machine learning-based california wildfire risk prediction and visualization,” in *2023 International Conference on Machine Learning and Applications (ICMLA)*, 2023.

[5] A. Malik, N. Jalin, S. Rani, P. Singhal, S. Jain, and J. Gao, “Wildfire risk prediction and detection using machine learning in san diego, california,” in *2021 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 2021.

[6] A. H. Kassandra Hernandez, “Machine learning algorithms applied to wildfire data in california’s central valley,” *Trees, Forests and People*, 2024.

[7] D. Adhikari, W. Chen, Y. Guo, L. Huang, and J. Gao, “Wildfire progression prediction and validation using satellite data and remote sensing in sonoma, california,” in *Wildfire Progression Prediction and Validation Using Satellite Data and Remote Sensing in Sonoma, California*, 2023.

[8] S. Cheng, Y. Jin, S. P. Harrison, C. Quilodrán-Casas, I. C. Prentice, Y.-K. Guo, and R. Arcucci, “Parameter flexible wildfire prediction using machine learning techniques: Forward and inverse modelling,” *Remote Sensing*, 2022.

[9] T. Jiang, S. K. Bendre, H. Lyu, and J. Luo, “From static to dynamic prediction: Wildfire risk assessment based on multiple environmental factors,” in *2021 IEEE International Conference on Big Data (Big Data)*, 2021.

[10] A. Singh, R. Yadav, G. Sudhamshu, A. Basnet, and R. Ali, “Wildfire spread prediction using machine learning algorithms,” in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2023.

[11] N. K. Pahuja and M. Rivero, “Predicting the impact of wildfire using machine learning techniques to assist effective deployment of resources,” in *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2022.

[12] M. Nocerino and S. Ghosh, “Maximizing prediction accuracy in wildfire severity: A comprehensive analysis of machine learning models using environmental features,” in *2023 IEEE Global Humanitarian Technology Conference (GHTC)*, 2023.

[13] A. Malik, M. R. Rao, N. Puppala, P. Koouri, V. A. K. Thota, Q. Liu, S. Chiao, and J. Gao, “Data-driven wildfire risk prediction in northern california,” *Atmosphere*, 2021.

[14] S. Girtsou, A. Apostolakis, G. Giannopoulos, and C. Kontoes, “A machine learning methodology for next day wildfire prediction,” in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, 2021.

[15] Pandas, “Pandas.” [Online]. Available: https://pandas.pydata.org/

[16] Python, “Python 3.10.0 release.” [Online]. Available: https://www.python.org/downloads/release/python-3100/

[17] Apache, “Pyarrow - apache arrow python bindings.” [Online]. Available: https://arrow.apache.org/docs/python/index.html

[18] Google, “Datasets: Imbalanced datasets.” [Online]. Available: https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets

[19] scikit learn, “scikit-learn: Machine learning in python.” [Online]. Available: https://scikit-learn.org/stable/

[20] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[21] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, 1967.

[22] I. Rish, “An empirical study of the naive bayes classifier,” *IJCAI 2001 workshop on empirical methods in artificial intelligence*, 2001.

[23] PyTorch, “Torch.utils.data documenation.” [Online]. Available: https://pytorch.org/docs/stable/data.html

[24] Nvidia, “Nvidia cuda toolkit.” [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[25] RAPIDSAI, “Rapids installation guide.” [Online]. Available: https://docs.rapids.ai/install/

[26] NumPy, “Numpy.” [Online]. Available: https://numpy.org/

[27] facebookresearch, “Faiss.” [Online]. Available: https://github.com/facebookresearch/faiss

[28] Joblib, “Joblib: running python functions as pipeline jobs.” [Online]. Available: https://joblib.readthedocs.io/en/stable/

[29] Anaconda, “Anaconda installation.” [Online]. Available: https://docs.anaconda.com/anaconda/install/