



University of Colorado **Denver**

Burned Area Mapping in Alaska

by improving the Machine Learning Algorithm established by
United States Geological Survey for Conterminous US in 2020
SIU YIN LEE

Master of Science in Statistics

Department of Mathematical and Statistical Science
University of Colorado, Denver

Committee:

Dr. Yaning Liu (Chair)
Dr. Erin Austin
Dr. Jan Mandel

Project Presentation Date: May 2, 2022

Abstract

Fires impact our ecosystems. In order to study the impact of past fires and forecast future patterns, a reliable and consistent record of fire is essential. However, very few agencies consistently track fire occurrence over space and time. The incompleteness of fire data makes assessing trends and impacts of fires challenging. To address this issue, USGS (United States Geological Survey) developed a machine learning algorithm to map burned areas for the Conterminous US using Landsat satellite images for date ranges from 1984 to present. The algorithm performs well in comparison to similar research; however, the error rates in USGS's published research results in 2020 suggest rooms for potential improvement. Therefore, prior to expanding its fire mapping coverage from Conterminous US to the next biggest territory, Alaska, USGS sought to modify current algorithm for better prediction performance. This project is inspired by, and partially belongs to USGS's burned area mapping expansion effort, which aims to develop accurate and efficient tools for mapping burned areas in Alaska. In particular, in this project, 5 aspects of the existing USGS algorithm: data-split, model evaluation metric, classifier, feature selection, hyperparameter tuning were examined and discussed. Using the existing USGS algorithm as a baseline model, the 5 aspects were modified one by one progressively for experiments and result benchmarking. At each stage of the modification process, performance accuracy and efficiency were compared between the models before and after the modification. Overall, at the end of the modification process, an improvement of approximately 5.5% in Average-Precision score was achieved. Omission Error and Commission Error were reduced by 9.74% and 2.28% respectively. The process of model training, tuning and evaluation was shortened from a total of 3 Days 18 hours (90 hours) to 19 hours utilizing fewer computation and parallelization resources.

Table of Content

1. **Project Background**
 - 1.1. **The importance of comprehensive fire records**
 - 1.2. **Current issues regarding fire records**
 - 1.3. **Current efforts in burned area archive**
2. **Project Motivation**
 - 2.1. **Train a Machine Learning model specifically for Alaska**
 - 2.2. **Modify existing Machine Learning algorithm to lower error rates**
 - 2.3. **Speed up algorithm training and prediction efficiency**
3. **Project Goal and Scope**
4. **Data**
5. **The 2020 USGS Burned Area Mapping Algorithm for CONUS**
6. **Project Methodology Overview**
 - 6.1. **Benchmark Process**
 - 6.2. **HPC Clusters**
 - 6.3. **Validation Data**
7. **Algorithm Modifications and Benchmark Results**
 - 7.1. **Stage 1: Fix Data Leakage**
 - 7.2. **Stage 2: Change Model Selection Metric**
 - 7.3. **Stage 3: Classifier Modification**
Stage 3.5: Introduce Additional Features
 - 7.4. **Stage 4: Feature Selection Modification**
 - 7.5. **Stage 5: Hyperparameter Tuning Modification**
8. **Concluding Summary**
9. **Future Suggestions**
10. **Acknowledgement**

1. Project Background

1.1. The importance of comprehensive fire records

Fires impact our ecosystems. Their scale, location and frequency can cause significant environmental, ecological, economic and social effects on the lives of all creatures. In order to access and quantify the impact of fires, understand their past occurrences and forecast future patterns, a reliable and consistent record of fire are essential. The more reliable and authoritative the record, the more convincing it is when scientists need to provide evidence to inform public and policy makers about the kinds of immediate actions and resources required to take care of our planet. Under the current debates on the views of climate change that are still going on in 2022, the importance of scientific evidence revealed by well-established data records is more critical than ever.

1.2. Current issues regarding fire records

Even though the necessity of comprehensive fire records seems obvious and burned area was listed as one of the 13 terrestrial essential climate variables for systematic observations by the Global Climate Observing System (GCOS) as an acknowledgement of fire records' importance in the study of climate change, very few agencies track fire records consistently over time and space. The incompleteness of fire data makes the study of fire trends and impacts challenging (Hawbaker T. , et al., 2017) (Hawbaker T. J., et al., 2020).

For example, in the United States, MTBS (Monitoring Trends in Burn Severity) is an agency that maps fires of the US using the Landsat satellite archive. However, MTBS only records large fires (>1000 acres in western US and >500 acres in eastern US) (MTBS, n.d.). Smaller fires or fires that lasted for a relatively shorter period are often not known or recorded. When the documentation of smaller, shorter fires is neglected, the reliability of scientific assessment and quantification become problematic as the occurrences and patterns of smaller fires themselves can reveal vital patterns regarding the state of our planet.

Moreover, human visual inspections are often used for fire mapping, which leads to the issues of efficiency and accuracy (MTBS, n.d.). As stated in the MTBS's Mapping Methods page,

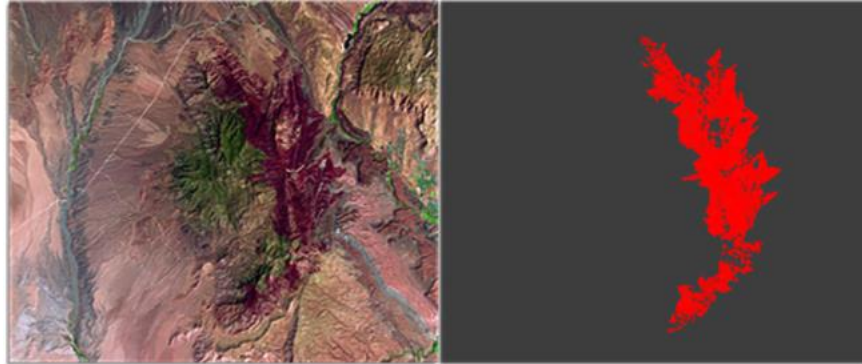
“The process of developing a categorical burn severity product is subjective and is dependent on analyst interpretation... There are uncertainties in this approach stemming from analyst subjectivity.”

1.3. Current efforts in burned area archive for the conterminous US and Alaska

To generate accurate and complete records of fire occurrences, since 2017, USGS (United States Geological Survey) has developed a machine learning algorithm to map burned areas for the conterminous US using Landsat satellite images. Such algorithm was then revised in 2020. This USGS burn area mapping science product provides a burn classification and a burn probability for the area of interest and is available for date ranges from 1984 to present. This burn area archive has benefitted many research communities as the product has a variety

of potential uses such as fire hazard evaluation, fire management, emissions estimation, tracking of greenhouse gas generation etc. (USGS, n.d.)

Below is an example of Landsat satellite image and its corresponding burned area mapped by the algorithm. (USGS, n.d.)



Left: Landsat Surface Reflectance image (Landsat 7 bands 5,4,3) and Right: Derived Burned Area product for an area within Landsat CONUS ARD tile h006v010 acquired on July 7, 2003.

As an expansion of the current Landsat burned area product, USGS plans on including Alaska, the next biggest territory in the US, into their mapping domain as their next phase of development. This paper is inspired by, and partially belongs to USGS's Landsat burned area product expansion effort, which aims at developing accurate and efficient tools for mapping burned areas in Alaska.

2. Project Motivation

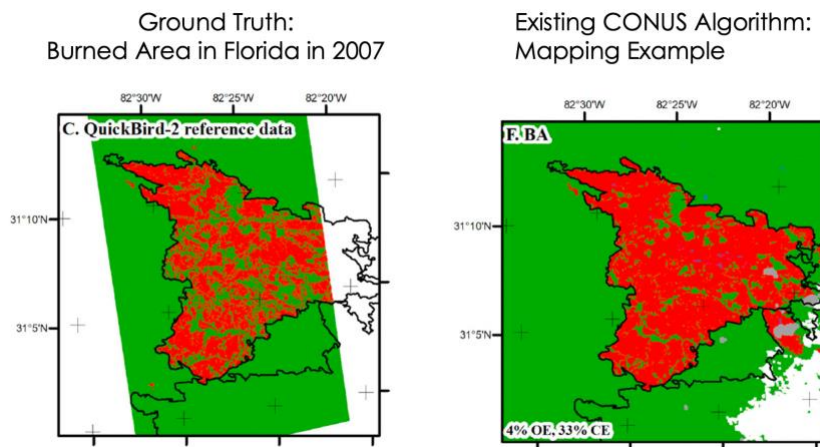
2.1. Train a specific machine learning model for Alaska

Even though fire mapping algorithms already exists for the conterminous US (CONUS), it would be inappropriate to directly apply the trained CONUS model on Alaska's satellite images due to Alaska's unique climate characteristics. For example, Alaska has significantly different length of sunlight/darkness compared to CONUS. The conditions under which fire happens in Alaska are not the same as CONUS. For example, according to Robert Ziel, fire analyst at the University of Alaska Fairbanks, 'While much of the fire-prone landscape in CONUS depend on fuels on the surface to ignite and support spread, Alaska's landscape is characterized by deep duff layers under the boreal forest and tundra environments that dry more slowly, hold heat when dry, and resist extinguishment from all but season-ending weather events. This deep duff layer develops from very slow decomposition, holds a great deal of carbon, and provides a deep insulating blanket for the permafrost below.' (Ziel, 2019)

Due to the environmental differences between CONUS and Alaska, the relationship and pattern between predictors, the satellite indices, and the response, the classification of fire, can be very different between CONUS and Alaska. A new model, therefore, needs to be trained specifically for Alaska using Alaska satellite data.

2.2. Modify existing machine learning algorithm to lower error rates

The CONUS fire mapping algorithm performs well in comparison to other burned area products, yet the error rates are still relatively high. When the model's classification performance was evaluated using an independent high-resolution satellite imagery dataset, the average omission error was 19% and average commission error was 41% among all the predicted pixels in the imagery (Hawbaker T. J., et al., 2020). How the model performs on the Alaska dataset is unknown; however, the error rates in the 2020 USGS paper seem to suggest that there are rooms for improvement for the algorithm's performance. Below is a mapping example using the CONUS Algorithm for a burned area in Florida in 2007. The Algorithm has a Commission Error of 33% and an Omission Error of 4% on this example image.



When the 2020 USGS algorithm was examined, it was suspected that several aspects of the algorithms might have contributed to the error rates. These aspects include data splitting method, model selection metric, classifier, feature selection process and hyperparameter tuning method. In this project, modification regarding these 5 aspects of the algorithms will be experimented. The details regarding these aspects and their modification will be discussed in Section 7: Algorithm Modifications and Benchmark Results of this paper.

2.3. Speed up algorithm training and prediction efficiency

As satellite imagery technology evolves, the availability and volume of data are also rising tremendously. To keep up with the data size growth, machine learning algorithm must find a way to speed up model training and evaluation. In the satellite imagery field, even though the availability of analysis ready data, ARD, has simplified a lot of the preprocessing work, compared to the older path/row data format, ARD data volume has increased 340% for satellite images of the same region and time period (Hawbaker T. J., et al., 2020). With the recent release of the Harmonized Landsat Sentinel-2 data by NASA, which is generated by combining the observations captured by two different satellite series – Landsat and Sentinel-2, researchers will have access to images that are of even higher resolution and denser time-series. Such great news implies the potentials for more accurate and vaster analysis, yet it also implies the need for more efficient machine learning algorithm is now immediate.

3. Project Goal and Scope

To summarize, the goal of this project is to modify the 2020 CONUS burned area mapping algorithm in order to develop a more accurate and efficient model for mapping burned area in Alaska.

The scope of the project focuses on the domain of machine learning algorithm analysis and the task of pixel-level burned area classification. Areas related to ecological study, the preprocessing of satellite images, and the sampling and validation process of high resolution satellite images are not the subjects of study in this project.

Moreover, this project aims to identify potential causes for 2020 USGS algorithm's error rates and to offer model modification suggestions that are backed up by practical needs and justifications. It is not meant to be a survey or an exhaustive testing of methods that aims at searching for the ultimate solution.

4. Data

The dataset used in this study has a total of 886,792 datapoint, in which consists of about 94.6% (839059) of unburned datapoint and 5.4% (47733) of burned datapoint. Every data point represents a pixel that was sampled and processed by the USGS science team. The fire date range of these samples ranges from 2000 – 2017.

4.1. Response variable

The response label “fire” is a binary variable, where 1 represents a burned pixel and 0 represents an unburned pixel. The primary source of the fire label was the MTBS (Monitoring Trends in Burn Severity) data and thus the label was mostly generated by MTBS's scientists' visual inspections.

4.2. Predictor variables

Predictors in the dataset include 133 continuous variables. They are satellite spectral indices that indicate land surface conditions, reference conditions and change metrics. Raw band indices Band 1-7 are also present in the predictor variable set. For detailed description of these indices, please refer to the USGS 2020 paper's section 2.1. (Hawbaker T. J., et al., 2020)

5. The 2020 USGS burned area mapping algorithm for CONUS

In order to make changes for improvement, we first need to understand the algorithm used in the 2020 USGS paper. In this section, the burned area mapping algorithm used by USGS in 2020 for CONUS will be illustrated.

5.1. Data Split of 2020 model

The 2020 training data was split into two parts: 50% train set and 50% test set. Besides the training data, two sets of completely separate reference data, which include a set of high-resolution commercial images, were used to benchmark final performance before publication.

5.2. Algorithm details of 2020 model

	Aspect	2020 CONUS Algorithm	Hyper-parameter	Values tested
1	Data Split	50% Train Set: 50% Test Set		
2	Model Selection Metric	AUC Score	N_estimator	1000
3	Classifier	Gradient Boosted Trees	Learning_rate	0.01, 0.05, 0.1
4	Feature Selection	Forward Selection	Max_depth	1, 3, 5, 7
5	Hyperparameter	Only 12 combinations tested		

The 2020 model adopted a Gradient Boosted Tree Model to estimate the probability that each pixel in a Landsat ARD tile had burned. The metric used for evaluation was AUC score. The feature selection method chosen was Forward Selection and 12 combinations of hyperparameters (1 N_estimator x 3 Learning_rate x 4 Max_depth) were tested during hyperparameter tuning.

5.3. Flowchart of 2020 model algorithm



Descriptions of the flowchart:

1. For each of the 12 hyperparameter combination, fit a Gradient Boosted Tree model with 1000 trees.
2. Each of the 12 models went through individual stepwise forward feature selection process which had an improvement threshold of 0.001 in AUC score. A correlation threshold of 0.95 was also set to eliminate candidate feature that had a correlation of 0.95 or higher with the already selected feature.
3. Choose the final model that has a highest AUC score on the test set.

6. Project Methodology Overview

In this project, 5 aspects of the USGS 2020 model will be examined: Data Split, Model Selection Metric, Classifier, Feature Selection Method and Hyperparameter Tuning Method. Some special technical notes are described in the following:

6.1. Benchmark Process

The methodology of this project is analogous to a recursive car tuning and test-drive process. At each stage, only one component of the model will be examined and modified. Benchmark will be conducted to compare model performance before and after the specific component change at each stage. Model modification will be done in a cumulative manner throughout the process and benchmark will be conducted between the model at current stage and the model at the prior stage.

The process starts with Stage 0, at which we will train a model using the same algorithm in the 2020 USGS paper and use it as our first baseline model. At Stage 1, the data-split aspect of the algorithm will be modified and the Stage 1 model will be compared against the Stage 0 model. At Stage 2, an extra component, model evaluation metric, will be modified on the Stage 1 model. The resulted Stage 2 model will then be compared against the Stage 1 model. This modification process will be performed for 5 stages in total. In the end, the final Stage 5 model will be compared against the original Stage 0 model for an overall comparison. The following table lists out each stage's modification aspect:

Stage	Aspect	Original	Change
1	Data Split	50% Train Set 50% Test Set	60% Train Set 20% Test Set; 20% CV Set
2	Model Evaluation Metric	AUC Score	Average Precision Score
3	Classifier	Gradient Boost	XG Boost
4	Feature Selection	Forward Selection	Boruta + Feature Importance Thresholding
5	Hyperparameter Tuning	Grid Search of 12 combinations tested	Optuna Bayesian TPE 11 dimensions - 200 trials

6.2. HPC Clusters

This project utilized the Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University. All analyses were conducted on the SHAS partition, which consists of 380 nodes, 24 cores/nodes, 4.84 GB Ram/core

6.3. Validation Data

In the 2020 USGS paper, two independent reference datasets were used to validate final algorithm performance before publication. These two reference datasets include a set of high-

resolution satellite data and a set of Landsat reference data. (Hawbaker T. J., et al., 2020) As these independent reference datasets for Alaska were not available or accessible for this project, 20% of the Alaska training dataset was held up to be the validation set for final benchmarking prior to any further train-test splits for model training.

As our final Alaska model benchmark were not conducted on equivalent reference datasets prepared by the same detailed procedures described in the 2020 USGS paper, it will not be a fair assessment to compare this project's validation scores with the 2020 USGS paper's validation scores. For objective comparisons, when evaluating the effectiveness of each stage's modification, performance should only be compared among the validation scores within this project.

7. 2020 Algorithm Issues, Corresponding Modifications and Benchmark Results

In this section, the 5 aspects of the 2020 USGS Algorithm mentioned: data-split, model evaluation metric, classifier, feature selection and hyperparameter tuning will be examined and discussed. Each aspect's details, corresponding modification and benchmark results will be provided in its individual subsection. Overall result summary and concluding remarks can be found in Section 8.

Stage 1: Fix Data Leakage

7.1. Stage 1: Fix Data Leakage

The goal of predictive modeling is to create a model that can make accurate prediction on unseen dataset. In order to evaluate a model's generalization ability on new dataset, a common technique used in Machine Learning is to hold up a portion of the data prior to training and only use that held-up dataset for final evaluation. (Brownlee, Data Leakage in Machine Learning, 2020)

In the 2020 USGS algorithm, the training data set was split into a train set and a test set. This train-test split, as mentioned, is a standard Machine Learning modeling practice. Nonetheless, in the 2020 USGS algorithm, during the feature selection phase, the test set had been used to evaluate and select features prior to final validation. Having the test set exposed during the model selection phase caused a problem of Data Leakage.

7.1.1. What is Data Leakage?

According to Kaufman et al., data leakage is deemed “one of the top ten data mining mistakes”. (Kaufman, Rosset, Perlich, & Stitelman, 2012) In Machine Learning, Data leakage happens when the model, during the training process, is exposed to information it is not supposed to know. (Brownlee, Data Leakage in Machine Learning, 2020)

In practice, the introduction of this illegitimate information is unintentional, and is facilitated by the data collection, aggregation and preparation process.’ (Kaufman, Rosset, Perlich, & Stitelman, 2012)

7.1.2. Why is Data Leakage a problem?

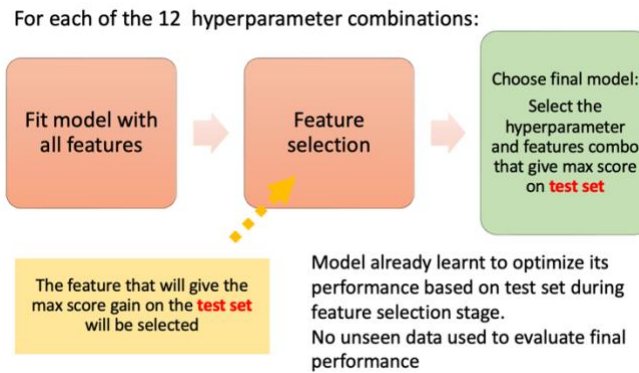
The main problem caused by Data Leakage is that it invalidates the estimation of the model's performance on unseen data and misleads researchers to have an overoptimistic belief on their model's performance. (Brownlee, Data Leakage in Machine Learning, 2020)

As summarized by Kaufman et al., ‘Leakage is undesirable as it may lead a modeler...to learn a suboptimal solution, which would in fact be outperformed in deployment by a leakage-free model that could have otherwise been built. At the very least leakage leads to overestimation of the model's performance. A client for whom the modeling is undertaken is likely to discover the sad truth about the model when performance in deployment is found to be systematically worse than the estimate promised by the modeler.’ (Kaufman, Rosset, Perlich, & Stitelman, 2012)

7.1.3. Where is the Data Leakage in the 2020 USGS Model?

As illustrated in the diagram below, in the 2020 USGS model training process, during the feature selection phase, a feature was selected if it gave a maximum AUC score gain on the test set. At the end, the performance of all 12 resulting models (with their subset features) were evaluated again on the same test set for final model selection. The issue

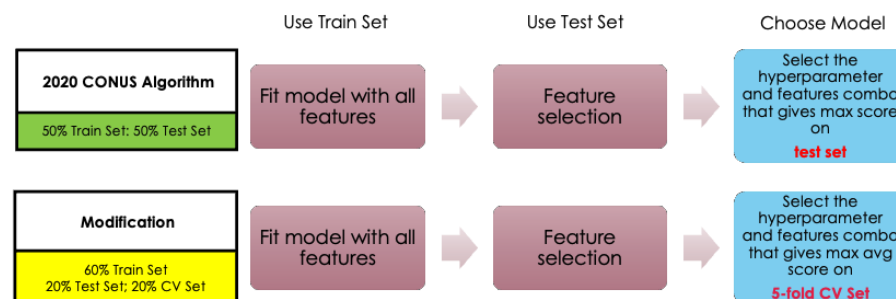
with this approach is that, final model evaluation was not done on an ‘unseen’ dataset. Data in the test set had already been ‘exposed’ to the classifier during the feature selection phase and each of the 12 model had already learned to optimize its performance on the test set by then. Therefore, the evaluation using test set was biased and could no longer show the real generalization capacity of the chosen model at the final model selection stage.



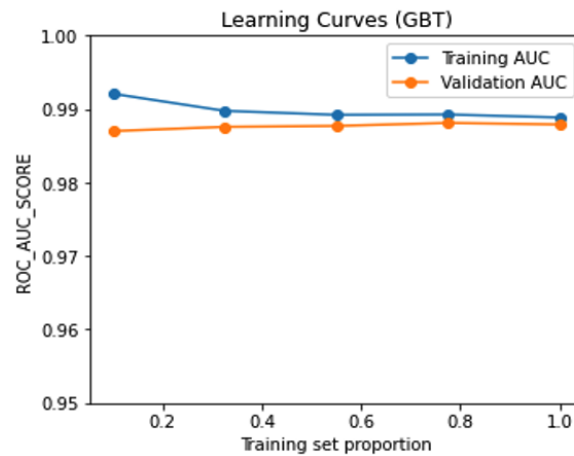
7.1.4. Algorithm Modification regarding Data Leakage: cross-validation set

	Aspect	Step 0 Algorithm		Aspect	Step 1 Algorithm
1	Data Split	50% Train Set 50% Test Set		1	60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	AUC Score		2	AUC Score
3	Classifier	Gradient Boost		3	Gradient Boost
4	Feature Selection	Forward Selection		4	Forward Selection
5	Hyper-parameter	Only 12 combinations tested		5	Only 12 combinations tested

As the first stage of our modification process, in order to evaluate final model performance on unseen data, an extra split is done on the training data. Instead of splitting the data into 50% Train Set and 50% Test Set, the data is now split into 60% Train Set, 20% Test Set and 20% Cross-Validation Set, so that we can use the extra Cross-Validation set to evaluate the model after feature selection. The diagram below indicates the change in the process:



Even though there is no fixed rule for optimal data split proportion in Machine Learning, the new split proportion 60-20-20 is one of the commonly-suggested proportion for a 3-way-split on large dataset. (Bressler & Tannor, 2021) (Baheti, 2022) In addition, at the initial phase of data exploration, Learning Curves were plotted to test out different Train-Test splits' performance. From the graph below, we can see that the performance scores of the train and test set start to converge and do not show much difference beyond a train-set proportion of 60%. It indicates that at this proportion, there is a good balance between the data sizes for learning and evaluation. Therefore, for our modification, training set was set at 60%, with the remaining split evenly between test set and cross-validation set.



7.1.5. Why 5-fold Cross-Validation?

With an extra portion of the data held up for final evaluation, our modification has already fixed the Data Leakage issue discussed. The implementation of 5-fold cross validation is simply for having a more robust evaluation on the performance of the model by taking performance variance of different evaluation datasets into account. By using 5-fold cross validation, we have the opportunity to evaluate the model 5 times and observe the mean scores and standard deviations. Therefore, the evaluation is more reliable than using one single evaluation dataset. Like the data split proportion, there is no fixed rule for the number of K in K-fold cross validation. For large dataset, 5 is a common choice as it is not too computationally exhaustive. Moreover, empirically, $k=5$ or 10 have been shown to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance. (James, Witten, Hastie, & Tibshirani, An Introduction to Statistical Learning, 2017)

7.1.6. Stage 1 Modification Results

The below table shows the benchmark results of this stage's modification:

	Step 0 Algorithm 50 Train– 50 Test	Step 1 Algorithm 60 Train – 20 Test – 20 CV
Hyperparameter	Max Depth: 7 Learning Rate: 0.1	Max Depth: 7 Learning Rate: 0.1
TNR = $TN/(TN+FP)$	99.72%	99.7%
TPR = $TP/(TP+FN)$	79.92%	79.51%
FNR (Omission Error) = $FN/(TP+FN)$	20.08%	20.49%
FPR = $FP/(TN+FP)$	0.28%	0.3%
Commission Error = $FP/(TP + FP)$	5.72%	6.3%
# of Feature selected	11	10
Full Pipeline Process Time	3D 9H 27M 23S	2D 6H 31M 32S

Surprisingly, the original 2020 USGS algorithm and the modified algorithm at this stage chose the same model combination (learning_rate = 0.1 and max_depth=7) but the performance on the final validation set is slightly worse for the modified algorithm. As shown in the result table, compared to stage 0, TNR and TPR of stage 1 have both slightly decreased, while error rates have increased.

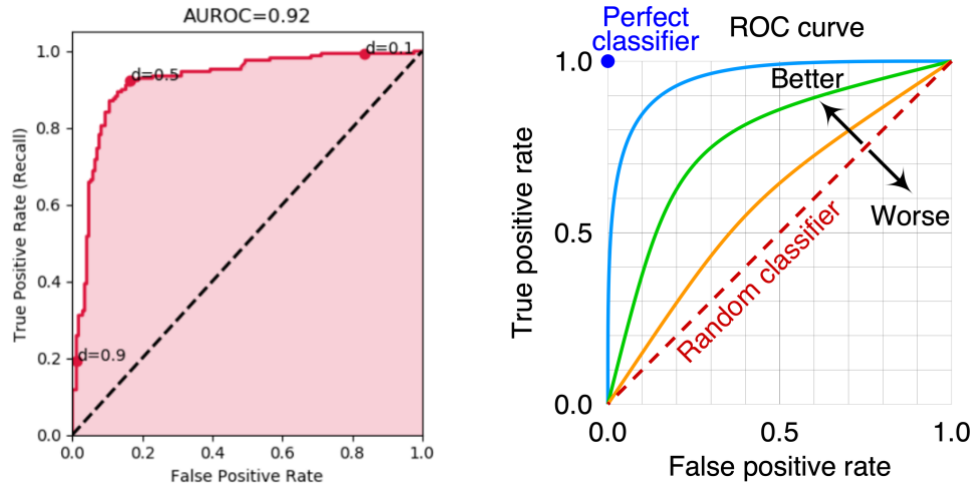
Reflecting on the result, even though the introduction of the CV split gave us the opportunity to evaluate our model in a more robust manner, with this modification, the test set size, which was used to perform feature selection has been significantly reduced from 50% of the data to 20% of the data. This resulted in a reduction in the number of selected features from 11 to 10. The change in the set of features may be the reason for the slight decline of performance.

Stage 2: Change Model Selection Metric

7.2. Stage 2: Change Model Selection Metric

In the 2020 USGS algorithm, AUC Score was used as a model selection metric. Features and models were selected if they yielded the highest AUC score. At this stage of modification, we will experiment switching the model selection metric from AUC to Average Precision (AP) Score. Explanations will be given below.

7.2.1. What is AUC Score?



AUC Score (also known as AUC-ROC) is the Area-Under-Curve for the Receiver Operator Characteristic (ROC) curve. The ROC curve is a probability curve plotted with True Positive Rate (TPR) against False Positive Rate (FPR). It summarizes the performance of a binary classifier and measures the trade-off between TPR and FPR at different classification decision thresholds. (Flatley, 2021) This method was originally developed for operators of military radar receivers starting in 1941, hence the name. (Wikipedia, n.d.) The left (Flatley, 2021) and right (Wikipedia, n.d.) graphs above are two examples that can help us understand AUC Scores.

On the left graph above, various classification decision thresholds are plotted on the red ROC curve. When different classification thresholds are chosen, the number of positive and negative classes predicted by the classifier will change and so will the TPR and FPR. To see how TPR and FPR changes at different thresholds, we rely on the ROC curve. (Flatley, 2021)

Examining the trade-off between TPR and FPR is not the only use of the ROC curve, more often, researchers use the area under the curve (AUC) to evaluate different models' performance. A High AUC score indicates that a model has a high True Positive Rate while being able to keep False Positive Rate low globally across different decision thresholds. For example, consider a random classifier with an AUC score of 0.5, which is indicated by the red diagonal line on right graph above. At a FPR of 0.5, the random classifier's corresponding TPR is also 0.5. In contrast, if we look at a "better" performing classifier, which is indicated by the blue line on the right graph, at a FPR of 0.5, this

better classifier gives us a TPR of 1. Therefore, a classifier with a ROC curve expanding to the top left corner, which also indicates a bigger area under curve and thus a higher AUC score, is preferred. (Draeos, Measuring Performance: AUC(AUROC), 2019)

7.2.2. What is the downside of AUC score?

AUC score is a widely-used to assess models in Machine Learning, but for dataset that is highly imbalanced, AUC score can be misleading. (Brownlee, 2020)

When AUC Score is used for model evaluation, intuitively speaking, it means that our goal is to search for a model that maximizes TPR and minimizes FPR. To illustrate the potential issue, let's first examine how are TPR and FPR calculated: (Brownlee, 2020)

$$\text{TPR} = \text{TP} / (\text{TP} + \text{FN})$$

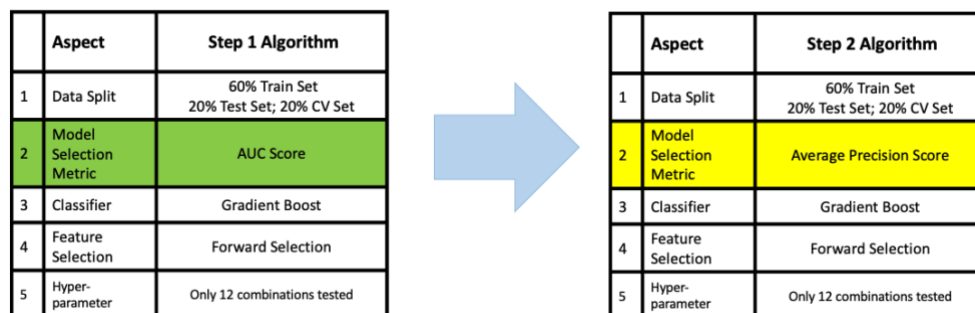
$$\text{FPR} = \text{FP} / (\text{FP} + \text{TN})$$

We can see from the second equation above that, if we want a low FPR, for given a fixed number of False Positive, we can do so by increasing the number of True Negative.

In our burned area project, the positive class refers to the burned area class and negative refers to the unburned area class. As one can imagine, as fire is a rare event, the majority of our data is unburned, and thus, the dataset is hugely imbalanced. Among all our training data points, 95% is unburned (negative) and only 5% is burned (positive). If we use AUC score to evaluate model, as mentioned above, so long as our model can classify a huge number of unburned points as unburned, the model has a low FPR and will appear to be a “good” model.

However, in the application of burned area mapping, even though we would like low errors in general, we are more concerned about the burned area (the positive, minority class) as that is the ‘signal’ we would like the classifier to pick up. When there is a severe class imbalance in dataset, the use of AUC score for model evaluation can be overly optimistic. (Brownlee, 2020)

7.2.3. Algorithm Modification regarding Selection Metric: Average Precision



In order to train and select a model that takes class imbalance into account, at this stage, our algorithm is modified to use Average Precision (AP) score as feature and model selection metric. The above diagram illustrates the modification details.

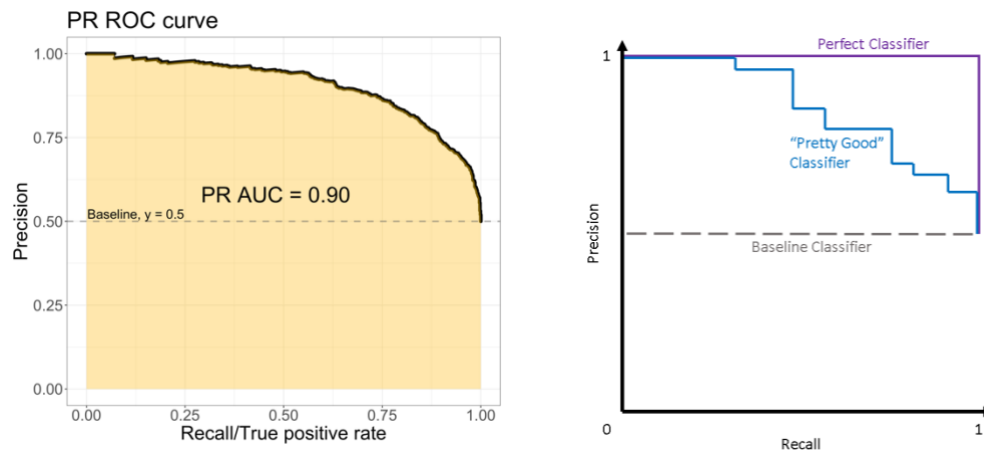
7.2.4. Why Average Precision?

Average Precision (AP), similar to AUC Score, measures the area under a curve. Yet, instead of measuring area under the ROC curve like AUC score, AP measures the area under the Precision-Recall (PR) curve. The two components of AP, precision and recall, are calculated as:

$$\text{Recall} = \text{TPR} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

The following pictures are examples of the PR curves and their area under curve (Chou, 2020) (Steen, 2020):



For a balanced dataset, a random classifier has an AP of 0.5, same as AUC. For imbalanced dataset, the baseline random classifier will have an $AP = \frac{\# \text{ positive class}}{\# \text{ total datapoints}}$ (Draelos, 2019). A skilled classifier, on the other hand, has a high precision and a high recall globally across different decision thresholds. Therefore, a model that has a PR curve closer to the top right corner is preferred.

The difference between AUC and AP solely lies in the replacement of AUC's FPR component with AP's Precision component. Below recaps the two components of AUC (left) and AP (right):

$$\begin{aligned} \text{TPR} &= \text{TP} / (\text{TP} + \text{FN}) \\ \text{FPR} &= \text{FP} / (\text{FP} + \text{TN}) \end{aligned}$$

$$\begin{aligned} \text{Recall} &= \text{TPR} = \text{TP} / (\text{TP} + \text{FN}) \\ \text{Precision} &= \text{TP} / (\text{TP} + \text{FP}) \end{aligned}$$

As shown in the contrast between the two pairs of formula, AP does not include TN in its calculation. It means that AP is not concerned about True Negative (ie classifying unburned as unburned). While AUC's FPR component tries to maximize TN for a fixed number of FP, AP's Precision component tries to minimize FP for a fixed number of TP.

Therefore, for an imbalanced dataset like ours, AP is more aligned with our goal of classifying the positive class, the burned area.

A Side Note:

AP uses rectangular approximation to calculate the area under the PR curve. According to Machine Learning library sklearn's documentation, AP is calculated as: (SKlearn developer, 2022)

$$AP = \sum_n (R_n - R_{n-1}) * P_n$$

Where P_n and R_n are the precision and recall at the N-th threshold.

Besides the use of rectangular approximation like AP, other common methods to calculate the area under PR curve are lower trapezoid estimator and interpolated median estimator. (Draeos, 2019) Depending on the calculation method used, the metric that measures the area under the PR curve may have different names (eg. AUCPR).

7.2.5. Stage 2 Modification Results

	Step 1 Algorithm Using AUC Score	Step 2 Algorithm Using Average Precision
Hyperparameter	Max Depth: 7 Learning Rate: 0.1	Max Depth: 7 Learning Rate: 0.05
TNR = TN/(TN+FP)	99.69%	99.76%
TPR = TP/(TP+FN)	79.51%	80.03%
FNR (Omission Error) = FN/(TP+FN)	20.49%	19.97%
FPR = FP/(TN+FP)	0.3%	0.24%
Commission Error = FP/(TP + FP)	6.3%	5.03%
# of Feature selected	10	14
Full Pipeline Process Time	2D 6H 31M 32S	5D 8H 24M 22S

At this stage of modification, the change of model metric from AUC to Average Precision has slightly improved TNR and TPR and decreased all error rates as indicated in the above table. The best selected model's learning rate has changed from 0.1 to 0.05 and the number of features selected has increased from 10 to 14.

The computational time of this stage has increased from 2 Days 6 Hours to 5 Days 8 Hours. It is a significant increase of computational time. After examining the results of other hyperparameter combinations at this stage, it is suspected that the long computational time was due to this particular hyperparameter combination of learning_rate and max_depth. At this stage, when max_depth of 7 and learning_rate of 0.1 were used, the same setting as Stage 1's model, the computational time was only 1 Day 18 Hours.

Stage 3: Classifier Modification

7.3. Stage 3: Adopt a newer form of the Classifier

The 2020 USGS paper used Gradient Boosted Trees as their algorithm classifier. At this stage of our project, we will change the classifier to XGBoost, an advanced version of Gradient boosting as our experimental model for benchmarking results. The rationale of this choice will be elaborated in the following subsections.

7.3.1. What is Gradient Boosted Trees

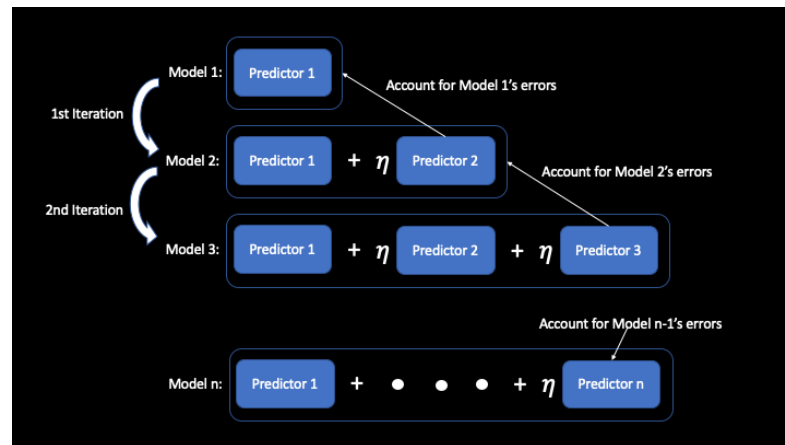
Gradient Boosting is a popular Machine Learning algorithm whose generalized form was first formulated by Jerome Friedman in 1999 (Chaturvedi, 2021). The crux of Gradient Boosting can be understood as follows (Tan, 2020):

Motivation: “If we can account for our model’s errors, we will be able to improve our model’s performance.”

Question: “But how do we know the error made by our model for any given input?”

Answer: “We can train a new predictor to predict the errors made by the original model.”

The following picture illustrates the process structure of Gradient Boost (Tan, 2020). When Decision Trees, the most common Gradient Boost base learner, is used, each of the blue predictor box refers to a Tree. The Greek letter, eta: η , indicates the small weight given to each tree, which is also known as the learning rate of the model. At each iteration, a new predictor (a.k.a. weak learner) is introduced into the ensemble to account for the previous model’s error.



As the name of Gradient Boosted Trees suggests, this algorithm can be understood by 3 key elements: Trees, Boosting and Gradient (Great Learning Team, 2020).

Tree: Gradient Boosted Trees uses Decision Tree as its base model. The basic idea of Decision Tree is that it uses different features and feature values as thresholds to make splits to divide the data. For each split, Decision Tree chooses the feature and values that

can reduce error or misclassification rate the most (commonly used metrics: entropy or Gini Impurity). The Tree keeps on splitting the data until no more improvement can be brought by further split, or a certain stopping criterion is met (Starmer, 2021).

Boosting: Boosting essentially means creating a model by combining many small Trees (a.k.a. an ensemble), where each subsequent small tree aims at correcting the errors of the previous tree. Each of the small tree is a weak learner itself as it does not make prediction well on its own, but with an ensemble of them correcting errors one by one, the model is given the opportunity to gradually learn during training and become a stronger predictor as a whole (Wade, 2020).

Gradient: The key characteristics of Gradient Boost is that it minimizes the loss function of the model by performing gradient descent with its weak learners (Great Learning Team, 2020). Gradient descent is a first-order iterative optimization algorithm extensively used in Machine Learning. However, unlike many machine learning algorithms (e.g. Neural Networks or Regression), which operate gradient descent in the parameter space to search for θ (the parameters), Gradient Boost operates gradient descent in the functional space to search for a function, which is the ensemble predictor/weak learner to be added into the model (Chung, 2019) (Hug, 2019).

To briefly illustrate the similarity of Gradient Boost and gradient descent mathematically (Hug, 2019), we consider the loss function:

$$\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2, \text{ where } \hat{y}_i = X_i^T \theta$$

In gradient descent:

We start with a random θ , and optimize \mathcal{L} with respect to θ by iteratively updating θ using the following update rule:

$$\theta^{(m+1)} = \theta^{(m)} - \text{learning_rate} * \frac{\partial \mathcal{L}}{\partial \theta^{(m)}} \quad \text{--- equation 1}$$

Let $\text{learning_rate} * \frac{\partial \mathcal{L}}{\partial \theta^{(m)}} = b_m$

Then $\theta^{(m+1)} = \theta^{(m)} - b_m$
 $\theta^{(m+2)} = \theta^{(m+1)} - b_{m+1} = \theta^{(m)} - b_m - b_{m+1}$
 \dots

The final optimal hyperparameter we get, $\theta^{(optimal)}$ can therefore be expressed as

$$\theta^{(optimal)} = \sum_{m=1}^{n_iter} b_m$$

In Gradient Boost:

We start with a random prediction \hat{y}_i , and optimize \mathcal{L} with respect to \hat{y}_i by iteratively updating \hat{y}_i using update rule:

$$\hat{y}_i^{(m+1)} = \hat{y}_i^{(m)} - \text{learning_rate} * \frac{\partial \mathcal{L}}{\partial \hat{y}_i^{(m)}} \quad \text{--- equation 2}$$

As you can see, equation 1 and 2 are the same, except the substitution of θ with \hat{y}_i .

However, one key note is that:

As we want the model to predict unseen samples, in the Gradient Boost algorithm, we will not use the actual values of the gradient of training samples to update \hat{y}_i . Instead, we will train a base learner (commonly a Decision Tree) to predict the gradient descent step during each iteration. The algorithm saves all the base estimators and use them altogether to output predictions for unseen samples.

Therefore, the final optimal prediction we get: $\hat{y}_i^{(optimal)}$ can be expressed as

$$\hat{y}_i^{(optimal)} = \sum_{m=1}^{n_iter} h_m(x_i)$$

where h_m represents the base learner at iteration m. Each one of the h_m is not directly predicting the target y_i , but the gradients with input feature x_i .

This is how Gradient Boost operates gradient descent in a functional space.

The mechanism described above also differentiates Gradient Boost from another popular boosting algorithm, Adaptive Boosting (Adaboost). Adaboost learns through iterative sample weight adjustments and bootstrap resampling (Starmer, 2021). It raises the weights of misclassified sample instances and lowers the weights of correctly classified samples. On the other hand, Gradient Boost does not care at all about the predictions that are already correct. Gradient Boost simply models each new tree entirely around the residual errors of the previous predictor (Wade, 2020).

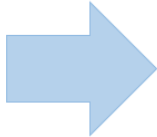
7.3.2. What is the downside of Gradient Boosted Trees?

Gradient Boosted Trees is a highly flexible and powerful algorithm for building predictive models. Unlike many other algorithms, it does not even require data scaling. However, the major downsides of Gradient Boosted Trees are (Kurama, 2020):

1. Due to its tendency to keep minimizing errors, overfitting can easily occur by overemphasizing outliers.
2. With the huge number of features and threshold to consider at each split, it is very computationally expensive – both time and memory exhaustive!

7.3.3. Algorithm Modification regarding Classifier: XGBoost

To address the shortcomings of Gradient Boosted Trees, at this stage of our study, we will make our modification by changing our classifier to XGBoost.

	Aspect	Step 2 Algorithm		Aspect	Step 3 Algorithm
1	Data Split	60% Train Set 20% Test Set; 20% CV Set		1	Data Split 60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	Average Precision Score		2	Model Selection Metric Average Precision Score
3	Classifier	Gradient Boost		3	Classifier XG Boost
4	Feature Selection	Forward Selection		4	Feature Selection Forward Selection
5	Hyper-parameter	Only 12 combinations tested		5	Hyper-parameter Only 12 combinations tested

XGBoost literally stands for “Extreme Gradient Boosting” and is an algorithm built upon the foundation of Gradient Boosting in 2014 (XGBoost Developers, 2022) (XGBoost, n.d.). As its API structure and settings are very similar to Gradient Boost, the shift from Gradient Boost to XGBoost only requires minor edits in the code.

But more than convenience, the main reason that XGBoost is adopted in our modification is its superb performance and efficiency. XGBoost has been the winning algorithm used in many Kaggle competitions and has a large community support (Adebayo, 2020). For structured or tabular datasets, XGBoost was often found to outperform even deep learning algorithm like neural nets (Machine Learning Mastery, 2016).

The details of XGBoost will be discussed in the following section.

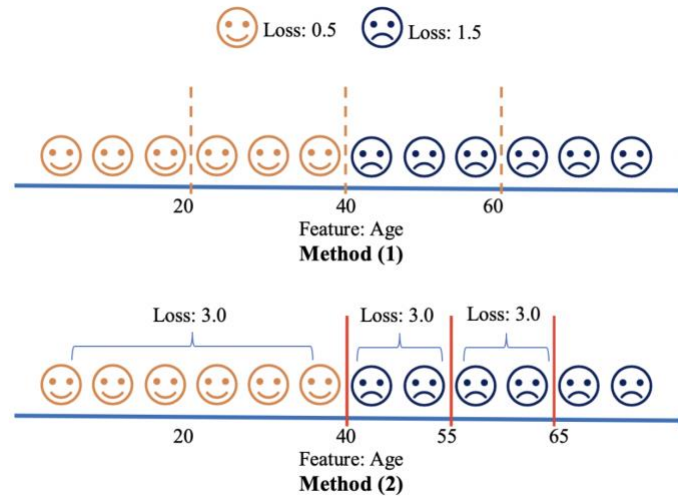
7.3.4. Why XGBoost?

The “Extreme” in XGBoost actually means “pushing computational limits” to the extreme (Wade, 2020). Here, we will briefly discuss the distinct design features that help XGBoost perform its magic, with a special focus on how they address the shortcomings of overfitting and computational efficiency of Gradient Boosted Trees.

To address the high computational cost for determining tree splits in Gradient Boosted Trees, XGBoost presents an approximate split-finding algorithm called Weighted Quantile Sketch for huge dataset and makes use of parallelization when looking for optimal splits.

The idea of Weighted Quantile Sketch is that it makes a histogram for each feature and use the bins’ boundaries as candidate split points. Therefore, the algorithm saves time by considering far fewer split candidates as now the data are put into bins. In addition, to make sure that the split candidate points are well-chosen and can help improve model performance (i.e., reduce loss), heavier weights are given to data points that are not well predicted yet/have higher loss. Then, in the histogram, instead of having the same number of data points in each bin, Weighted Quantile Sketch puts the same total weights in each bin. By doing so, the algorithm will consider more split candidates and conduct more

detailed search in areas where the model needs improvement (Tan, 2020). The diagram below (Wang, 2020) illustrates the difference between a typical Quantile split (Method 1 in diagram) and the Weighted Quantile sketch (Method 2 in diagram):



Other than Weighted Quantile Sketch, XGBoost makes use of parallelization to accelerate the split search. In general, parallelization is hard to be implemented in Boosting model because each weak learner added depends on the previous ensemble's errors. Thus, it is a serial operation. Nonetheless, XGBoost makes parallelization happen – not on the ensemble/tree level, but on the feature level. For example, when searching for the optimal split point for a leaf, each core can be calculating the loss of each feature's split points. The final split point for that leaf can then be decided after making comparison among all the calculated losses (Tan, 2020).

To address the tendency to overfit in Gradient Boosted Trees, XGBoost has many regularization hyperparameters that can help control the growth of a tree, but the key distinction between Gradient Boost and XGBoost is their objective function. XGBoost has regularization parameters built into its objective function. Please note that there is a difference between hyperparameters and parameters: the former refer to settings that cannot be learned by the algorithm, while the latter refer to values that are learned via training. Below is the objective function of XGBoost (Machine Learning Mastery, 2020):

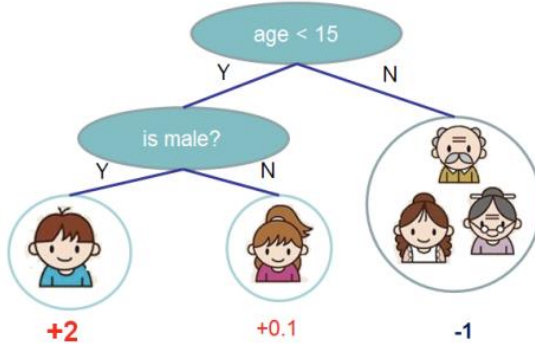
$$Obj = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

where the first term is the training loss and the second term is the regularization. As XGBoost learns to minimize loss that encourages growth of the tree, the regularization controls the complexity of the tree. Now we will focus on the discussion of the regularization term Omega:

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

The complexity of a tree $\Omega(f_t)$ is calculated with two parts. The first part is a γ -weighted T, which indicates the number of leaves in the tree. The second part is a λ -weighted L2 norm of the leaf weight. In the equation, Gamma: γ and Lambda: λ are user-defined

hyperparameters. The following illustrates the calculation of $\Omega(f_t)$ (Machine Learning Mastery, 2020):



For a tree with 3 leaves whose weight is 2, 0.1 and -1 respectively, Ω will be:

$$\Omega = \gamma 3 + \frac{1}{2} \lambda (2^2 + 0.1^2 + (-1)^2)$$

Other than being a regularized version of Gradient Boost, XGBoost employs a functional gradient descent up to the second-order derivative. This makes it more precise in finding the next best learner. It also uses Taylor expansion to approximate the loss, increasing flexibility of objective function choices and speed up calculation (Chung, 2019). We will look at the objective function again but focus on how Taylor expansion is applied to get the final form of XGBoost's objective function.

$$Obj = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

As we know that boosted trees is an additive algorithm which sums up the predictions of all trees to get final prediction, the prediction given by the final model t will thus be:

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

With this understanding, we can rewrite the objective function as:

$$Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Depending on the loss function choice, optimizing the loss can be complicated when the form of the function (e.g. logistic loss) is not friendly for derivation. In order to increase flexibility in objective function choices, XGBoost uses Taylor Approximation up to the second-order:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$$

Applying Taylor on the loss part of the Objective Function:

$$Obj^{(t)} \approx \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$\text{Where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \text{ and } h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$

After removing all the constants, the Objective Function for model t will be:

$$Obj^{(t)} \approx \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Important Notes:

(1) Noting that the objective function now only depends on g_i and h_i , therefore XGBoost can support custom loss functions: it simply needs to use one solver that takes g_i and h_i as inputs to optimize every loss function (XGBoost Developers, 2022). This is how XGBoost uses Taylor expansion to allow flexibility in loss function choices.

(2) Note that for the current round: model t , the values of g_i and h_i remain the same as they are simply the first and second order derivatives of the loss at previous iteration with respect to predictions. It means that g_i and h_i can be computed before the search starts at this current round and can be multiplied into the objective function by simple plug-in. This largely reduces computing cost when the tree is evaluating different tree splits and is how XGBoost uses Taylor expansion to accelerate the speed (Chung, 2019) (Samudrala, 2018).

Besides what has been mentioned above, XGBoost has other design features that make it significantly faster: sparsity aware split-finding, cache-aware access and block compression and sharding etc.. These designs are mostly about improving data reading time, computational memory and storage (Wade, 2020). As these topics are mostly related to the field of computer science, they will not be discussed in details in this paper. (Sparsity aware split-finding is also related to how XGBoost handles missing data.)

7.3.5. Stage 3 Modification Results

	Step 2 Algorithm Gradient Boosted Tree (Best)	Step 2 Algorithm Gradient Boosted Tree (Same Hyperparameter)	Step 3 Algorithm XGBoost
Hyperparameter	Max Depth: 7 Learning Rate: 0.05	Max Depth: 7 Learning Rate: 0.1	Max Depth: 7 Learning Rate: 0.1
TNR = TN/(TN+FP)	99.76%	99.71%	99.78%
TPR = TP/(TP+FN)	80.02%	79.34%	84.75%
FNR (Omission Error) = FN/(TP+FN)	19.97%	20.66%	15.25%
FPR = FP/(TN+FP)	0.24%	0.29%	0.22%
Commission Error = FP/(TP + FP)	5.03%	6.09%	4.43%
Average Precision	93.9%	92%	96.1%
# of Feature selected	14	10	21
Full Pipeline Process Time	5D 8H 24M 22S	1D 18H 14M 46S	1D 9H 56M 13S

The above result table displays the benchmark of this stage of modification. The first column is the best performing model during Stage 2 using Gradient Boosted Tree. The second column is provided to compare Gradient Boost and XGBoost performance using the same hyperparameter setting. The third column is the best performing model during Stage 3.

Please note that:

At this stage, XGBoost's n_job parallelization option has **not** been utilized. Even though all runs were conducted using 24 cores in the computing node, the cores were occupied by multiprocessing in the forward feature selection portion in both Stage 2 and Stage 3. Therefore, there is no “cheating” involved by XGBoost by utilizing multi-core parallelization at this stage.

Compared to Stage 2's best performing model (1st column), XGBoost has demonstrated an increase of Average Precision by 2.2% and an incredible decrease of processing time from 5D 8H 24M to 1D 9H 56M despite an increase of selected feature size from 14 to 21 features.

Compared to Stage 2's model with the same hyperparameter setting, XGBoost has significantly improved Average Precision by 4.1% and decreased processing time from 1D 18H 14M to 1D 9H 56M despite a doubling of selected feature size from 10 to 21 features.

From this stage's benchmark, we see the superior performance of XGBoost over Gradient Boosted Tree.

Stage 3.5: Introduce Additional Features

Readers of this paper may be curious why there is a Stage 3.5 between Stage 3 and Stage 4 and wonder if it is a mistake. The answer is: No, it's not a mistake, but an unexpected idea.

Context for this added stage:

When author worked at USGS, author was asked to train a model for Alaska without using the raw band features: Band 1 – Band 7. The request was most likely made because some of the remaining 126 features already used these raw band features in their formula calculation. See Table 2 of USGS's 2020 paper (Hawbaker T. J., et al., 2020). Also, in the 2020 paper result, these raw band features were not selected by the original algorithm. Therefore, they were deemed not necessary in our task of burned area classification. Even though the present project (this report) had already been started without including the raw band features, when author was looking at some of the satellite images on ArcMap, the need to revisit this decision was clear based on the following reflection:

“How were the burned/unburned labels in our training dataset generated? Weren't they produced via human visual inspection? If that is the case, the features related to the colors of these images perhaps would be useful for the algorithm to learn and predict these labels”

In fact, the raw band features were the ones most related to the images' color display (see below table (USGS, n.d.)), meaning their impact should be evaluated by adding them into the model.

Band 1	Band 2	Band 3	Band 4	Band 5	Band 6	Band 7
Coastal Aerosol	Blue	Green	Red	Near Infrared (NIR)	Shortwave Infrared 1 (SWIR 1)	Shortwave Infrared 2 (SWIR 2)

Result for Stage 3.5:

	Step 3 Algorithm XGBoost	Step 3 Algorithm XGBoost Add Raw Bands
Hyperparameter	Max Depth: 7 Learning Rate: 0.1	Max Depth: 7 Learning Rate: 0.1
TNR = $TN/(TN+FP)$	99.78%	99.80%
TPR = $TP/(TP+FN)$	84.75%	88.96%
FNR (Omission Error) = $FN/(TP+FN)$	15.25%	11.04%
FPR = $FP/(TN+FP)$	0.22%	0.20%
Commission Error = $FP/(TP + FP)$	4.43%	3.76%
Average Precision	96.1%	97.4%
# of Feature selected	21	15
Full Pipeline Process Time	1D 9H 56M 13S	20H 35M 32S

Compared to Stage 3, which does not use Raw Band features, their addition improved Average Precision Score by 1.3% and reduced Omission Error by 4.21%. Also, 3 out of 7 of the Raw Band: Band 6, Band 7 and Band 1 were selected by the model in the set of 15 total features. Therefore, these features were included for the remainder of the analysis for this report.

Stage 4: Feature Selection Modification

7.4. Stage 4: Select more robust features

Feature selection is a significant procedure in machine learning model training. The main purpose is to improve prediction accuracy, reduce model complexity and increase model interpretability by removing irrelevant features or noise in the model. (James, Witten, Hastie, & Tibshirani, An Introduction to Statistical Learning with Applications in R, 2017) In the 2020 USGS paper, stepwise forward selection was implemented as their feature selection method. In this stage, we will experiment using Boruta and Feature Importance thresholding to benchmark the results.

7.4.1. What is forward feature selection?

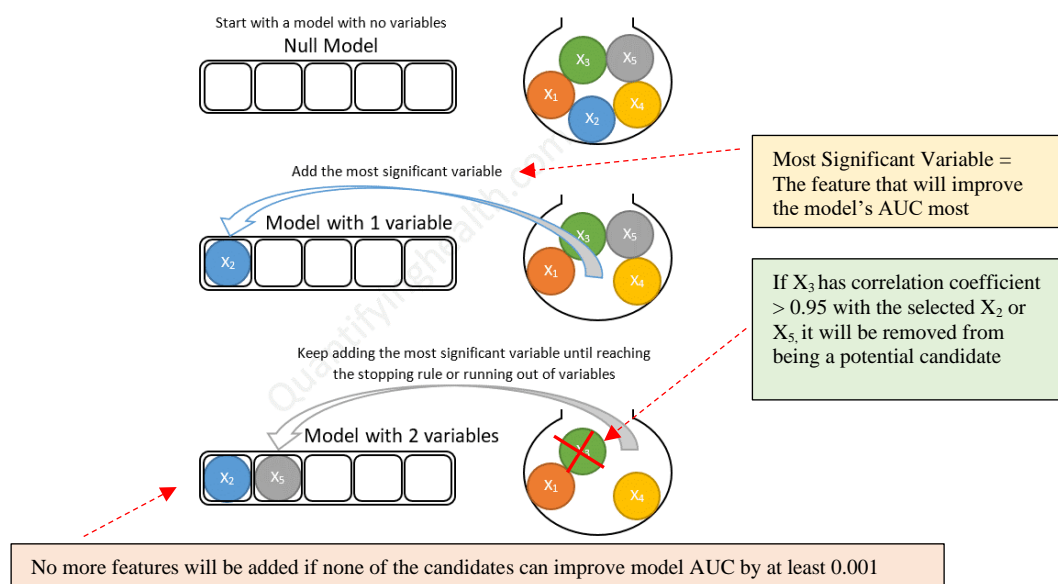
In essence, forward feature selection is a stepwise subset selection method “in which we start with having no feature in the model...[and] we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.” (sauravkaushik8, 2016)

Specifically, the forward selection process adopted by USGS in 2020 is described as the following:

“Our routine sequentially tested a suite of potential predictors and selected the single predictor that increased the GBRM’s AUC value most. During each step, remaining predictors to test were removed if they had a 0.95 or greater correlation with any of the selected predictors. This process continued until the change in AUC was <0.001 .” (Hawbaker T. J., et al., 2020)

Below is a graphical representation of the procedure. (Choueiry, n.d.) Side remarks are added to describe the 2020 USGS implementation specifics.

Forward stepwise selection example with 5 variables:



7.4.2. What is the downside of forward feature selection?

Compared to the ‘best subset’ selection method, which searches for the best subset by testing all possible feature combinations in the entire feature space, stepwise forward selection is a relatively, computationally-efficient method.

To understand this in concrete terms: in our case, with 133 features, for an exhaustive search to find its best subset, we will need to fit 2^{133} (2^p) models, which equals to an astronomical number of more than 1.08×10^{40} models. In contrast, when stepwise feature selection was used in stage 3, with 15 features selected, we approximately* only needed to fit $\sum_{k=0}^{15} (133 - k) = 2008$ models. (James, Witten, Hastie, & Tibshirani, An Introduction to Statistical Learning with Applications in R, 2017)

**In the formula, k can be understood as “when the k^{th} feature has been selected”. It is an approximation as the formula does not account for the correlation-drop USGS implemented.*

However, forward selection has a few main downsides:

1. Given its greedy nature, every step in the additive process is based on the current state of the already selected features. For example, in the scenario where we have a set of 3 predictors (X_1, X_2, X_3), if X_1 is chosen at Step 1, the algorithm can only add either X_2 or X_3 during Step 2. There is no way for the algorithm to choose the combination of (X_2, X_3) even if this outperforms (X_1, X_2) and (X_1, X_3). (James, Witten, Hastie, & Tibshirani, An Introduction to Statistical Learning with Applications in R, 2017)
2. As features are selected based on the “greatest additional improvement” criteria, features are competing among themselves to be selected. Even if X_2 and X_3 both have great predictive power on the response, only one of them can be chosen at each step. And once one is chosen, the other one may never make it into the selection basket due to the greedy nature mentioned in point 1.
3. The stopping threshold or criteria in the selection process blinds us from seeing the performance change or potential gain beyond the stopping spot. For example, in stage 3, 15 features were selected as none of the potential 16th feature could improve the Average Precision score by 0.001. However, hypothetically-speaking, it is possible that the 17th or 20th feature would be able to bring us substantial improvement that would be worth increasing the size of the subset further. Stepwise forward feature selection does not give us the chance to see and evaluate that possibility.

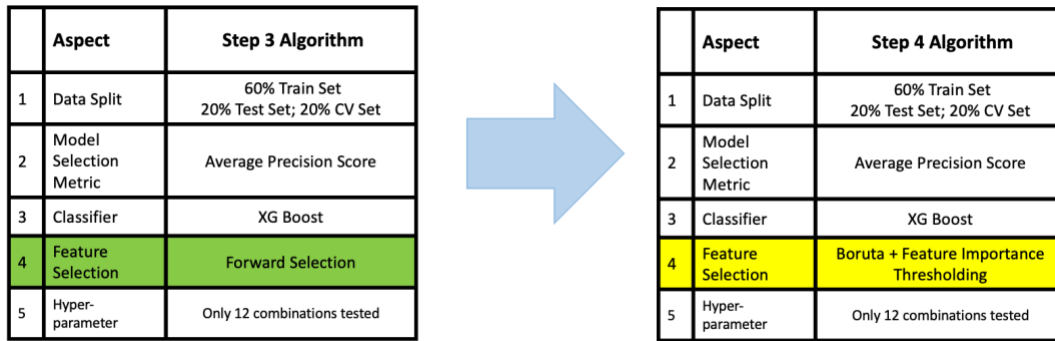
7.4.3. Algorithm Modification regarding Feature Selection – Part 1: Boruta and Part 2: feature importance thresholding

To address the mentioned shortcomings of stepwise forward feature selection, an alternative feature selection method, Boruta, was first implemented as the experimental model for this stage to benchmark against the stepwise forward feature selection method.

Correlation drop was also executed afterward using the same 0.95 threshold to make the benchmarks more comparable.

As the resulted subset of Boruta features was surprisingly big, feature importance thresholding was then implemented to further reduce the subset size. Details will be explained in the following subsections.

Below is the diagram that shows Stage 4's modification and benchmark details:




7.4.4. Why Boruta?

Boruta is a feature selection method that is statistically grounded and robust. Its key characteristics are that it makes use a permutation method and the binomial distribution to determine a feature's "usefulness". Boruta was originally invented by two Polish researchers at the University of Warsaw: Miron Bursa and Witold Rudnicki (Kordeczka, 2018)

Unlike stepwise forward feature selection, the method is not a greedy algorithm, and thus does not depend on the selection process's "current state". Moreover, Boruta does not make features compete among themselves. Instead, the features' competitors are the randomized versions of themselves, which are called "shadow features". (Mazzanti, 2020) Also, compared to stepwise forward feature selection, Boruta does not aim at finding a minimal subset of features, but finding all relevant features that are robust in predicting the response. Therefore, Boruta helps us better understand the true predictive power of the features. (Homola, 2015)

The following illustrates the mechanism of Boruta (Mazzanti, 2020):

Suppose we have 3 features: age, height and weight, and we want to use them to predict income.

	age	height	weight		income		age	height	weight	shadow_age	shadow_height	shadow_weight	
0	25	182	75	0	20		0	25	182	75	51	176	75
1	32	176	71	1	32		1	32	176	71	32	182	71
2	47	174	78	2	45		2	47	174	78	47	168	78
3	51	168	72	3	55		3	51	168	72	25	181	72
4	62	181	86	4	61		4	62	181	86	62	174	86

1. As the first step, Boruta will double the number of features by making a shadow copy of them.
2. Boruta will then use an estimator (XGBoost in our case) to fit these 6 features (original + shadow) on the response (income in this example).
3. Boruta calculates all 6 features' importance*.
4. On each trial run, if an original feature's importance is higher than the highest shadow feature's importance. It is considered as a "success".

**The meaning on feature importance will be explained in the Feature Importance Thresholding subsection below. For now, let's simply see it as a scoring on the features' contribution in prediction.*

The central idea of this mechanism is that: a feature is useful only if it's capable of doing better than the best randomized feature. (Mazzanti, 2020)

	age	height	weight	shadow_age	shadow_height	shadow_weight
feature importance %	39	19	8	11	14	9
hits	1	1	0	-	-	-

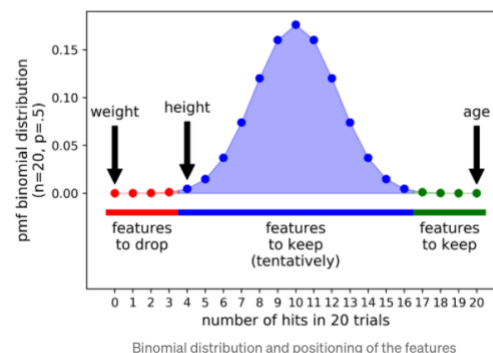
Outcome of one run

Suppose we did 20 trial runs with different versions of the randomized features (It means that at each trial, the features are shuffled differently to form the shadows):

	age	height	weight
hits (in 20 trials)	20	4	0

Outcome of 20 runs

Suppose "age" has higher ranking importance than all the randomized shadow features 20 out of 20 times, while "height" only 4/20 times and weight 0/20 times. How do we make the decision on which features to keep and which to drop? At this point, Boruta will use the binomial distribution to help us decide. Think of each run as an independent trial and the probability of getting a "success" randomly is 50%, like tossing a coin, we will then get a probability distribution for 20 trials like this:



If we set our probability threshold (alpha in statistics) at 0.01, meaning each side of the tails consists of 0.5% of the distribution, we will then get the red rejection region, green acceptance (strong features) region and blue (weak features) undecided region as shown

in the above graph. In this example, it means that Boruta suggests dropping weight, keeping age, and leaving height up to us because the binomial distribution indicates that it is very unlikely for “age” to outperform the shadow features 20 out of 20 times, and for “weight” to be outperformed by the shadow features 0 out of 20 times due to random chance.

7.4.5. Boruta Results

In stage 4 of our project, we originally have 133 features and implemented Boruta for feature selection using 100 trials and an alpha level of 0.01. Surprisingly, 126 out of 133 features are considered strong features by Boruta. Next, using the same 2020 USGS paper correlation threshold (0.95), we dropped features that are highly correlated with each other and ended up with 114 features.

Through the use of Boruta, we learn that our data set consists of many robust, strong features for classifying burned pixel. The majority of our features are not useless noise, with the only exceptions being the dropped “csi”, “csi_diff_3yr_over_mean”, “csi_difference_3yr”, “csi_mean_3yr”, “csi_sd_3yr”, “csi_z_score” and “vi46”.

Yet, the surprisingly large set calls for the need to further trim down the number of features for computation efficiency purposes.

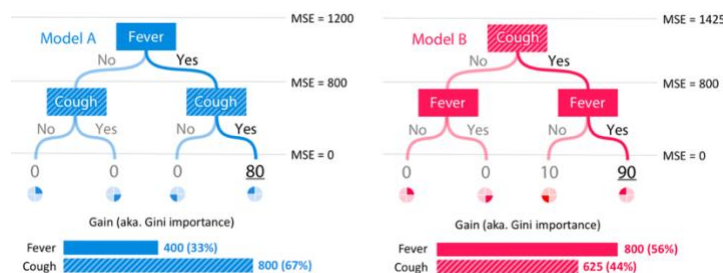
7.4.6. Feature importance thresholding

With the remaining 114 features we have after the Boruta and correlation drop procedures, we fit the features using XGBoost to check each feature’s importance in classifying burned area. After that, we examine how Average Precision score changes with each feature added into the model according to the order of feature importance to determine how many final features we should select.

7.4.6.1. What is Feature Importance?

There are different variations of feature importance calculation. The one we used was the widely-used “Gain” (aka Gini Importance), which calculates the average training loss reduction gained when using a feature for splits (Lundberg, 2018).

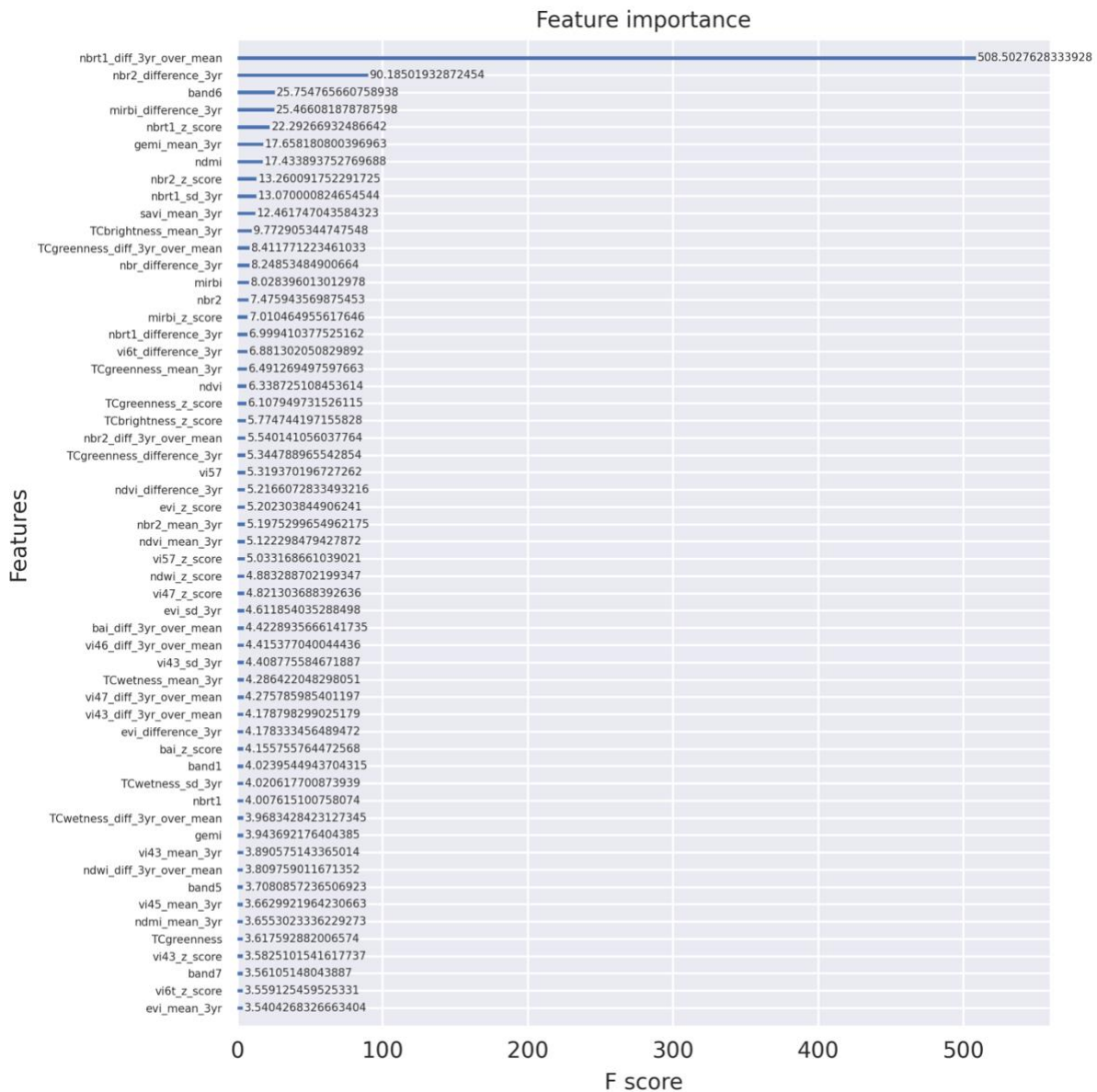
Below is a simplified illustration of the Gain calculation in a single tree (Lundberg, 2018). Here, two models (A and B) are using 2 features, Cough and Fever, to predict one’s health risk score:



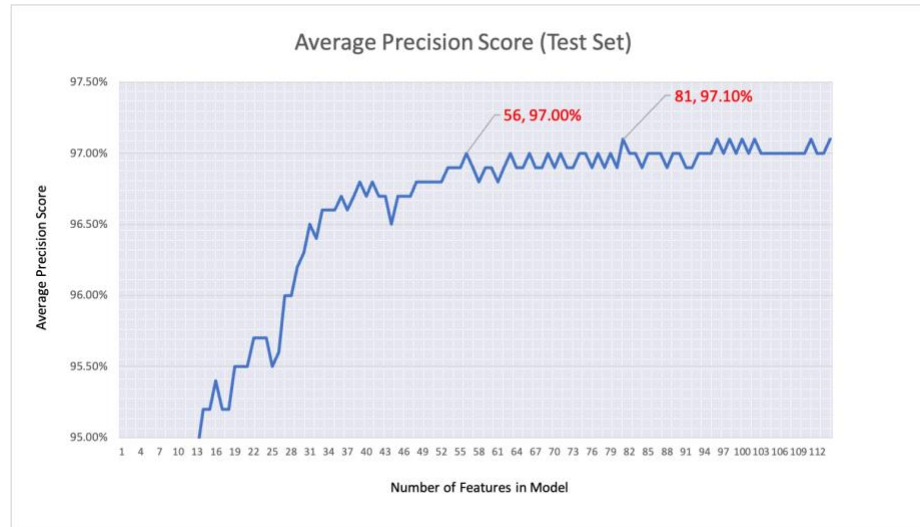
In Model A, Fever is used as the first split and reduces Mean Squared Error (MSE) by 400 (1200 – 800). Cough is then used as the second split and reduces MSE by 800. Therefore, Fever's Gain is 400 while Cough's is 800. If we normalize the values, Fever's Gain is 0.33 and Cough's is 0.67. Model B's calculation follows the same logic. (Lundberg, 2018)

7.4.6.2. How to use Feature Importance to select features?

In our project, we calculated the Gain feature importance of all 114 remaining features after the Boruta and correlation drop procedure. Below is the plot of the top 56 features' importance:



Given the evaluated feature importance, we can use a plot to observe how our test set score changes with each feature added into the model. The order of feature addition is determined by the feature importance ranking. Below displays the score change plot:



As we can see, the score fluctuates with a rising trend until about the 56 features from the previous figure are added into the model. To balance the goals of maximizing prediction performance and computational efficiency, 56 features were selected as our final feature subset.

7.4.6.3. Feature Importance Thresholding vs Forward Selection

It is reasonable to perceive the Feature Importance Thresholding method as very similar to stepwise forward selection. In a sense, they are similar because both methods examine the change of score by adding each feature. Nonetheless, there are some subtle, but important, differences:

In the stepwise forward selection method, the order of feature addition is determined by the next feature that can improve the Average Precision score in the test set the most, given the current state of the selected model. The benefit is that the method is score performance oriented. The downside is that the robustness of the feature may be questioned as it is unclear why it brings such improvement. On the other hand, the feature importance thresholding method adds each feature according to the feature importance ranking in the full-set model fit. A feature is added because overall, averaging all the tree nodes in the ensemble, the feature selected is the next feature that most improves accuracy of its splits. It measures the feature's effects on prediction across all tree nodes. Therefore, feature importance provides user the full picture of how each feature contributes to prediction (as shown in the feature importance bar chart) and the performance is backed by a clearer justification.

Also, since stepwise forward selection stops the selection once the improvement is smaller than the preset threshold, a user has no idea what is possible beyond a certain stopping point. On the other hand, as shown in the feature importance thresholding graph, the user is presented with the score change with each feature added. Therefore, a user can make a more informed decision about how they would like to balance between feature set size and score performance according to their needs.

In regard to computational time, as mentioned before, for stepwise forward selection, the total number of models needed to be fit is $\sum_{k=0}^s (p - k)$, where p is the total number of available features, s is the number of final selected features and k is the number of features already selected. On the other hand, for feature importance thresholding, the total number of models needed to be fit simply equals p , the total number of available features. In our case, with $p=133$ and $s=15$, feature importance thresholding reduced the number of model fits from 2008 to 133. Therefore, feature importance thresholding is much more efficient.

7.4.7. Stage 4 Modification Results

	Step 3.5 Algorithm Using Forward Selection	Step 4 Algorithm Using Boruta + Feat Imp
Hyperparameter	Max Depth: 7 Learning Rate: 0.1	Max Depth: 7 Learning Rate: 0.1
TNR = $TN/(TN+FP)$	99.80%	99.82%
TPR = $TP/(TP+FN)$	88.96%	89.42%
FNR (Omission Error) = $FN/(TP+FN)$	11.04%	10.58%
FPR = $FP/(TN+FP)$	0.20%	0.18%
Commission Error = $FP/(TP + FP)$	3.76%	3.42%
Average Precision	97.4%	97.7%
# of Feature selected	15	56
Full Pipeline Process Time	20H 35M 32S	5H 49M 41S

Note:

1. In the 2020 USGS paper, scene predictors have a higher priority in being selected in the feature selection procedure. As our main goal is to compare feature selection method, for simplification purpose, we did not implement this selection hierarchy in any of our baseline or experimental models. All features in this project were treated the same.

In this stage of modification on feature selection method, the experimental model of using Boruta and Feature Importance Thresholding have slightly increased TNR and TPR and slightly decreased all error rates as shown on the above table. Even though the feature subset chosen is substantially bigger, the total process time is significantly reduced. The speed-up is mainly explained by the reduced numbers of model fit needed. Another reason is that the 24-core (one node) originally used for multiprocessing in the feature selection process in the benchmark model is now freed up for XGBoost's model-fit parallelization.

Boruta and Feature Importance Thresholding provide the user a lot of insights on their features. Boruta aims at testing the relevance and robustness of the features in prediction.

Feature importance quantifies the contribution of the features in prediction. The thresholding mechanism allows a user to examine score changes as each feature is added into the model. However, neither Boruta or Feature Importance were built with the motivations to get the most compact subset. This distinction explains the larger feature set size in our result compared to stepwise forward selection.

Depending on user's priority, stepwise forward selection's compact set may seem more appealing. Nonetheless, it is important to bear in mind that the final selection size is dataset specific. If resources allow, for the sake of getting deeper understandings about one's features, Boruta and feature importance thresholding are highly recommended for feature assessment.

Stage 5: Hyperparameter Tuning Modification

7.5. Stage 5: A more informed way to tune hyperparameter

In the 2020 USGS paper, the hyperparameter tuning process was essentially a grid-search method. Even though the hyperparameter tuning process is bundled together with feature selection as a pipeline, it is basically a 3 x 4 grid consisting of 3 combinations of learning-rate and 4 combinations of max-depth.

In this stage, a “smarter” hyperparameter tuning method – Optuna Bayesian TPE is implemented to benchmark against the grid-search method.

	Aspect	Step 4 Algorithm		Aspect	Step 5 Algorithm
1	Data Split	60% Train Set 20% Test Set; 20% CV Set		1	60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	Average Precision Score		2	Average Precision Score
3	Classifier	XG Boost		3	XG Boost
4	Feature Selection	Boruta + Feature Importance Thresholding		4	Boruta + Feature Importance Thresholding
5	Hyper-parameter	Grid-Search Only 12 combinations tested		5	Optuna Bayesian TPE 200 trials

7.5.1. What is grid-search?

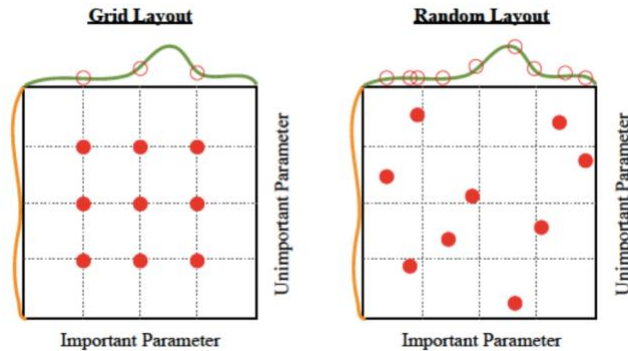
As the name suggests, grid search is a hyperparameter tuning method which evaluates model performance based on a user-defined grid. A grid is simply all the possible combinations of the user-defined values of each hyperparameter.

7.5.2. What is the downside of grid-search?

Grid-search is very simple to understand, execute, and parallelize, but there are two major downsides:

1. Grid-search is very computationally expensive. If one's model has many hyperparameters and one has no prior knowledge on which one to tune and what values to test, the computational resources needed increase exponentially with each hyperparameter added into the tuning process. For example, if one wants to test 10 values each for two hyperparameters A and B, one will need to test $10^2 = 100$ combinations. If one has 4 hyperparameters A, B, C and D, one will need to compute $10^4 = 10000$ combinations. Unless one has vast computational resources for large-scale parallelization, the tuning process for a big dataset can be horrifyingly long.
2. Even if one is willing to put in the computational resources, there is no guarantee that the search will yield good results. For hyperparameters that have a wide, continuous range, optimal combinations may lie between or beyond the user-defined grid. Very often, in practice, user needs to run multiple rounds testing different ranges to narrow the search space down. See illustration below that shows how

optimal solution is missed due to the restriction of a grid structure (Bajaj & ES, 2022):



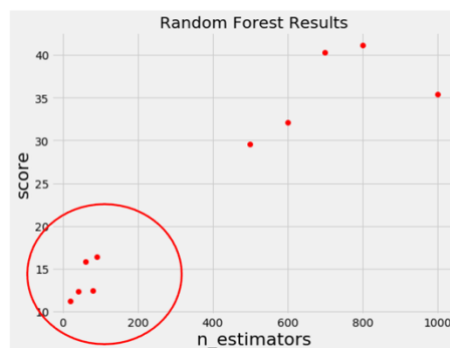
7.5.3. Algorithm Modification regarding Hyperparameter Tuning: TPE

The limitation of grid-search leads to the motivation to look for a “more intelligent” way to tune model hyperparameters. In Stage 5 of this study, Tree-Structured Parzen Estimator (TPE) is chosen as the experimental model for tuning hyperparameter. In particular, the Optuna library is used to execute the TPE method due to its relatively user-friendly interface and well-developed functionality in trial history tracking, parallelization and visualization, in comparison to popular libraries such as HyperOpt.

7.5.4. Why TPE?

Tree-Structured Parzen Estimator (TPE) is a type of Bayesian optimization methods that can help user implement “informed” searches via the use of past information.

To illustrate, consider a scenario where we need to minimize the loss function by testing different estimator sizes in a Random Forest. Based on the picture below with several initial runs’ results, using human visual judgment, we probably will think that smaller estimator sizes, like those in the red circle are more likely to yield lower loss. This is exactly the “intelligence” of Bayesian tuning method: unlike grid-search, instead of blindly running every single combination defined by the user without “learning” anything from each run’s result, or wasting resources by searching all over the place, it learns from past evaluation results to help user narrow down the search space to combinations that are likely to perform well (Koehrsen, 2018).



7.5.5. Bayesian Optimization in Brief

In essence, Bayesian hyperparameter optimization is the process of: “Build[ing] a probability model of the objective function to propose smarter choices for the next set of hyperparameters to evaluate.” (Koehrsen, 2018)

The probability model for the objective function is also called a “surrogate”, which is expressed by the probability of score (y) given hyperparameters (x):

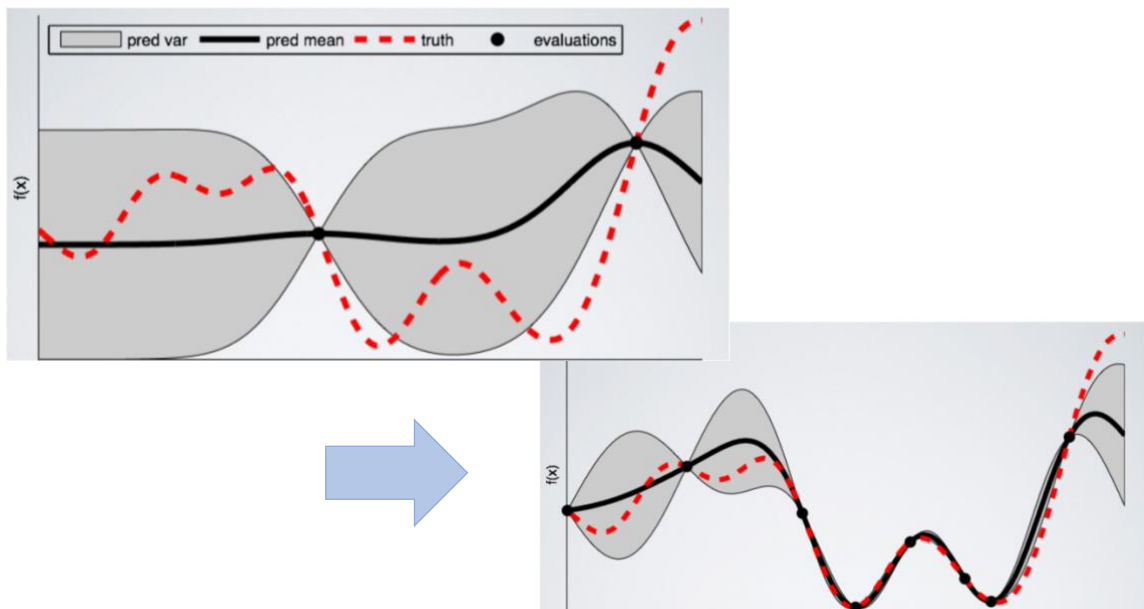
$$P(y | x) = P(\text{score} | \text{hyperparameters})$$

The optimization process can be understood as the following steps by making use of the surrogate and keeping track of past evaluation results (Koehrsen, 2018):

1. Build a surrogate probability model of the objective function
2. Find the hyperparameters that perform best on the surrogate
3. Apply these hyperparameters to the objective function
4. Update the surrogate model incorporating the new results
5. Repeat steps 2-4 until max iterations or time is reached

As the number of iterations of the above-described process increase, the surrogate approximation will become closer to the objective function. Therefore, by selecting hyperparameters that maximize/minimize the surrogate, it will likely maximize/minimize the objective function.

Below are the illustrations of the process. The black line is the surrogate model with uncertainty region in grey. The red line is the true objective function (Koehrsen, 2018). We see how the surrogate approximation becomes closer to the objective function from 2 evaluations (left figure) to 8 evaluations (right figure).



7.5.6. TPE Specifics in Brief

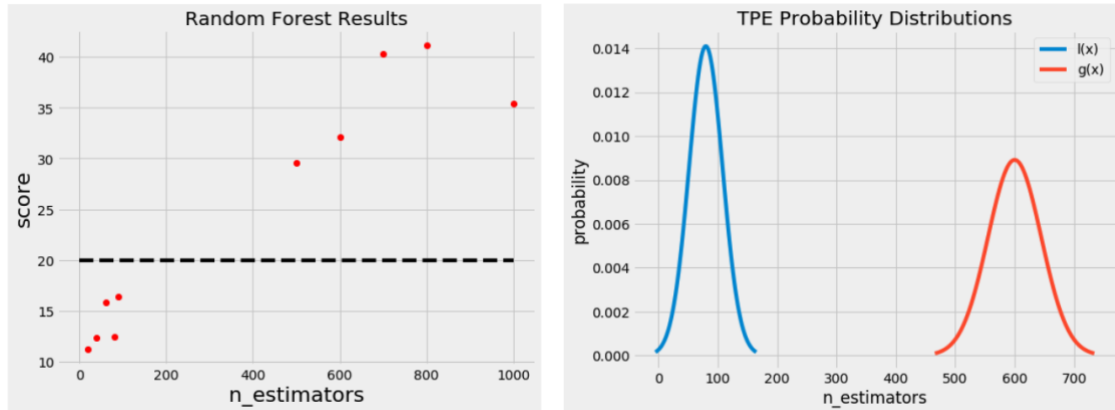
Different Bayesian hyperparameter optimization methods have different ways of building the surrogate probability model $P(y | x)$. Tree-Structured Parzen Estimator (TPE) builds its surrogate by applying the Bayes rule. Instead of modeling using $P(y | x)$, it uses $P(x | y)$, which is the probability of the hyperparameters given the score on the objective function. The flip is due to the Bayes Rule formula (Koehrsen, 2018):

$$P(y | x) = \frac{P(x | y) * P(y)}{p(x)}$$

Specifically, TPE models $P(x | y)$ by using a threshold y^* to make two different probability distributions $l(x)$ and $g(x)$ for the hyperparameters x . $l(x)$ is built upon the group of hyperparameters that yield objective scores lower than the threshold, and $g(x)$ is built upon the group of hyperparameters that yield objective scores greater than the threshold. Mathematically, it is expressed as:

$$P(x | y) = \begin{cases} l(x), & y < y^* \\ g(x), & y \geq y^* \end{cases}$$

Below left figure illustrates the threshold y^* with the dotted line. The right illustrates the two distributions $l(x)$ and $g(x)$ (Koehrsen, 2018).



Specifically, $l(x)$ and $g(x)$ are Gaussian Mixture Models (GMM). TPE fits and updates these two distributions on every trial, for every parameter (Optuna Developers, 2018).

Then, TPE draws sample x from $l(x)$, the distribution of the better performing group, and then evaluate the ratio $\frac{l(x)}{g(x)}$. The hyperparameter x that yields the maximum value of this ratio, which is associated with the greatest expected improvement, will be chosen to be evaluated on the objective function. As the number of trials increases, the surrogate function will become a closer approximation of the objective function, and therefore the hyperparameters that are chosen based on the evaluation of the surrogate are also likely to bring good results or score improvement on the objective function (Koehrsen, 2018).

7.5.4 Stage 5 Modification Results

	Step 4 Algorithm Using Grid-Search	Step 5 Algorithm Using Optuna TPE	Step 5 Algorithm Using Optuna TPE - Thres
Hyperparameter	Grid of 2 dimensions: 12 combinations	Space of 11 dimensions: 200 trials	Space of 11 dimensions: 200 trials
TNR = $TN/(TN+FP)$	99.82%	99.59%	99.82%
TPR = $TP/(TP+FN)$	89.42%	93.37%	89.66%
FNR (Omission Error) = $FN/(TP+FN)$	10.58%	6.63%	10.34%
FPR = $FP/(TN+FP)$	0.18%	0.41%	0.18%
Commission Error = $FP/(TP + FP)$	3.42%	7.22%	3.44%
Average Precision	97.7%	97.8%	97.8%
# of Feature	56	56	56
Avg Time per combination with cv (1 node)	9M 20S	3M 40S	3M 40S

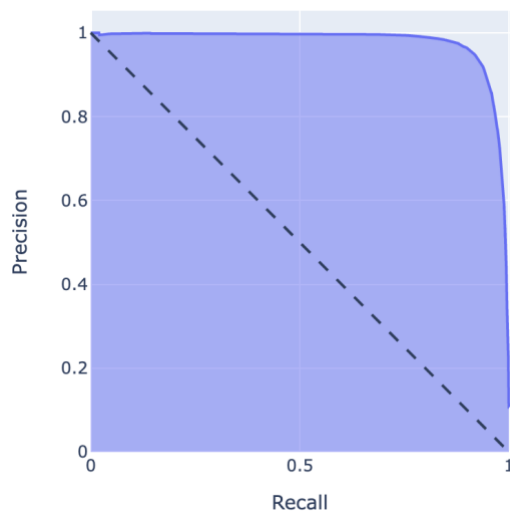
	Optuna Search Space	Final Hyperparameter
Reg_alpha	Float: [0, 10]	2.3087445982646853
Reg_lambda	Float: [0, 10]	9.185897991442204
Subsample	Float: [0.1, 1]	0.8376273311541326
Colsample_bytree	Float: [0.1, 1]	0.7333090667948312
Max_depth	Float: [2, 15]	10
Min_child_weight	Int: [2, 100]	8
Learning_rate	Float: [0, 1]	0.0571035315713899
Gamma	Float: [0, 10]	0.686509279863869
Grow_policy	["depthwise", "lossguide"]	"lossguide"
Max_delta_step	Float: [0, 10]	9.89518925762564
Scale_pos_weight	Float: [0, 20]	17.29253415864619

At this stage of modification, with 200 trials tuning 11 hyperparameters, Optuna TPE has slightly improved the model by 0.1% of Average Precision Score. In terms of computational time, Optuna evaluated each combination with only 3m 40s on average. Compared to Grid-search's average time of 9m 20s, it is 60% of time reduction.

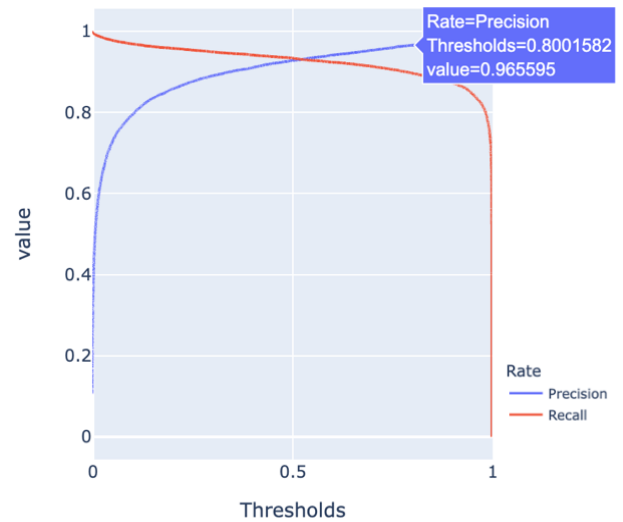
In the first table, 2nd column, we see that Optuna has significantly reduced Omission Error, but in exchange, increased Commission Error. To make comparison easier, we used Precision-Recall Curves (displayed below) to adjust the class prediction probability threshold to fix the trade-off between Precision and Recall. With a probability threshold for class 1 set at 0.8 for the Optuna final model, the Grid-search benchmark model and the Optuna model are fixed at the same precision rate, the result is then displayed on the 3rd column of the table.

Below are a few more plots for visualizing Optuna's tune results:

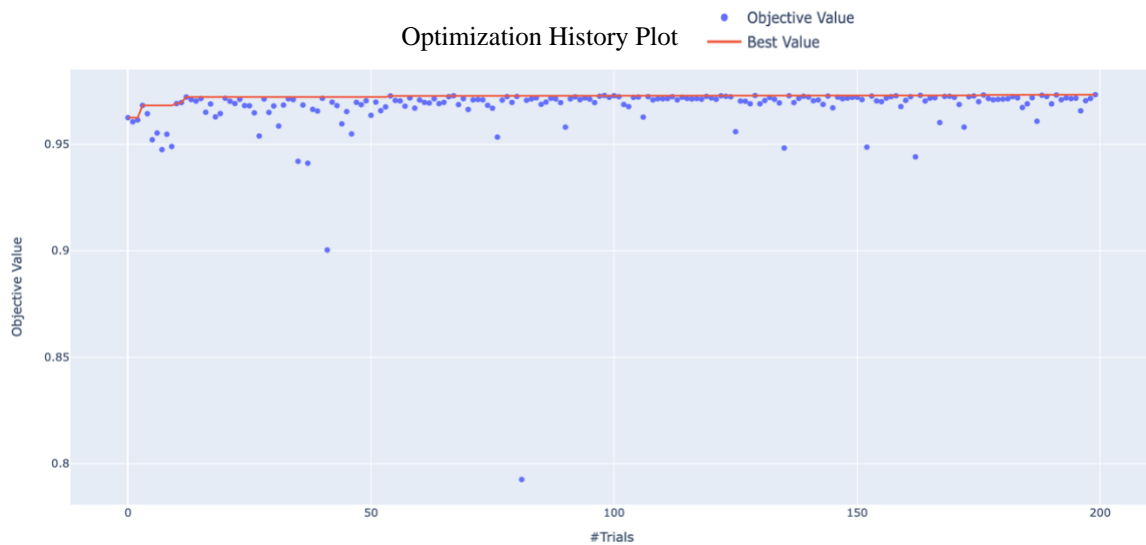
Precision-Recall Curve (AUC = 0.978)



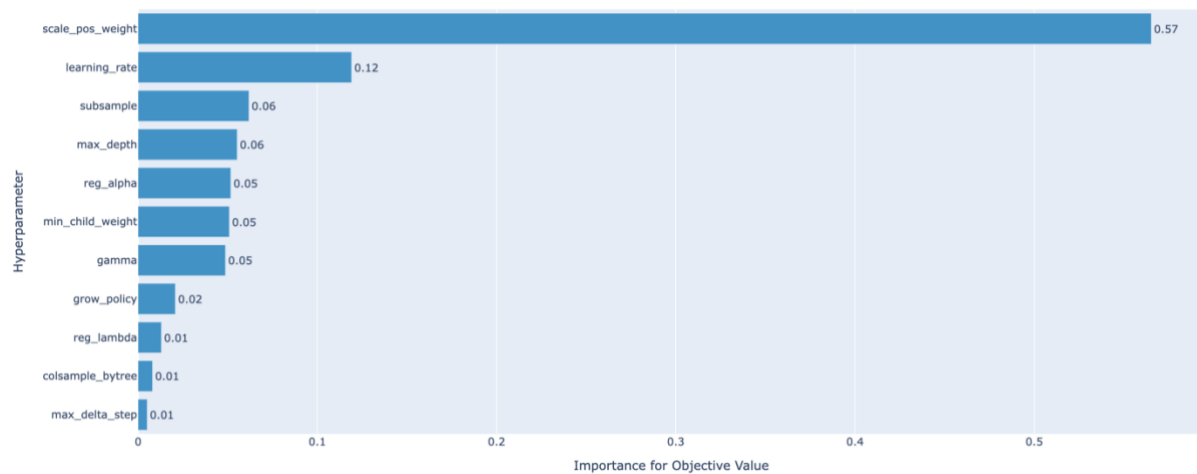
Precision-Recall at every threshold



Optimization History Plot



Hyperparameter Importance



From the result table, we see that Optuna TPE has brought only very slight prediction improvement to our model. Nonetheless, it will be injudicious to conclude that Optuna TPE and Grid-search are equal. Here are a few notes for consideration:

In terms of performance:

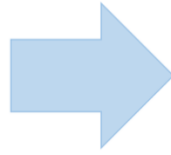
- As shown in Optuna's Optimization History Plot above, with a huge range of hyperparameters tested out in a 11-dimensional space, there are only very small variations in the objective score (Average Precision) from the beginning to the end of the tuning process. Only 9/200 (less than 5%) trials fall below the Average Precision score of 0.95. This possibly is an indicator that, given the data and selected features, our model performance is close to being maxed out and is relatively less sensitive to hyperparameter tuning, which possibly explains the small performance gain at this final stage of modification.
- As shown in the hyperparameter importance plot, Optuna has helped us identify many important hyperparameters that were not utilized in the grid-search model. The most important hyperparameter, `Scale_pos_weight`, which controls the balance of positive and negative weights and is useful for unbalanced classes, was not included at all in the grid (XGBoost Developer, 2022). This hyperparameter is shown to play a vital role in squeezing out the performance gain in our imbalanced dataset by Optuna.
- During our search, the option to take into account the dependence among the hyperparameters was activated. This function helps us get more optimized result faster, which Grid-search is not able to do.

In terms of efficiency:

- In our run, Optuna evaluated each combination using only 40% of the time grid-search needed.
- The search space Optuna was dealing with is tremendously bigger. It is 11 dimensions with many wide, continuous ranges. Without the intelligence of Bayesian optimization, it would be impossible to take a few shots in the dark to get the model optimized. In general, without prior knowledge, it will be unrealistic to expect a search of only 12 combinations (like in our grid-search model) in a 2-dimensional space to yield the best optimization result, unless one is extremely lucky.
- For hyperparameters like `Max_depth`, without the use of the surrogate function to evaluate the likelihood to yield high performance, it would take terribly long to tune high values of `Max_depth` when the dataset size is large.

8. Concluding Summary

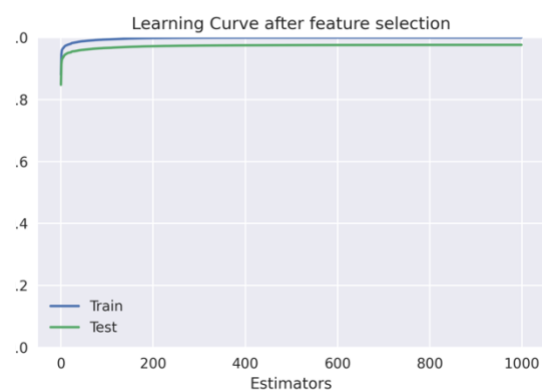
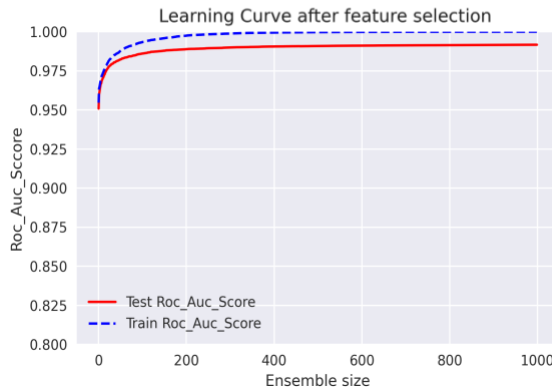
	Aspect	Original Algorithm
1	Data Split	50% Train Set: 50% Test Set
2	Model Selection Metric	AUC Score
3	Classifier	Gradient Boosted Trees
4	Feature Selection	Forward Selection
5	Hyper-parameter	Only 12 combinations tested



	Aspect	Final Algorithm
1	Data Split	60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	Average Precision Score
3	Classifier	XG Boost
4	Feature Selection	Boruta + Feature Importance Thresholding
5	Hyper-parameter	Optuna Bayesian TPE 11 dimensions - 200 trials

	Step 0 Original Model	Step 5 Final Model
TNR = $TN/(TN+FP)$	99.72%	99.82%
TPR = $TP/(TP+FN)$	79.92%	89.66%
FNR (Omission Error) = $FN/(TP+FN)$	20.08%	10.34%
FPR = $FP/(TN+FP)$	0.28%	0.18%
Commission Error = $FP/(TP + FP)$	5.72%	3.44%
Average Precision	92.3%	97.8%
# of Feature	11	56
Full Process Time (Train + Feat Select + Tune + Validate)	3D 18H 17M 39S	19 H 10M 45S

Original model (Left) and Final model (Right) Learning Curves



After a long journey modifying all 5 aspects of the original algorithm, the first two tables in this section have summarized all the modification details and results. Learning curves of the models are displayed to allow a check of (no) overfitting.

Overall, comparing our final model with the original model, we have:

In terms of predicting performance:

- Increased Model Average Precision Score by 5.5% (from 92.3% to 97.8%)
- Reduced Omission Error [TN/ (TP+FN)] by 9.74% (from 20.08% to 10.34%)
- Reduced Commission Error [FP/ (TP+FP)] by 2.28% (from 5.72% to 3.44%)

In terms of computational efficiency:

- Reduced full processing time (including Training + Feature Selection + Tuning + Validation)
by 79%, which is 71 Hours 7 Min (from 90H 17M* to 19H 10M**)

**In Stage 0, 12 models were run in parallel using 12 nodes, where each node computes one set of hyperparameters and goes through training, forward feature selection individually. The hyperparameter tuning part was tested in parallelly by 12 nodes; thus, the longest model processing time (from 3D18H17M39S) is used for measuring wall-clock time.*

***In Stage 4, Data went through Boruta and Feature Thresholding twice, one for outputting the threshold plot for examination (3H27M05S), one for running the selected number of features after threshold examination (3H29M17S). Then in Stage 5, the selected features go through Optuna hyperparameter tuning (12H14M23S), which totals to 19H10M45S*

In terms of computational resources:

- Reduced the use of computing resources from 12 nodes, with 24 cores each, to only 1 node with 24 cores.

9. Future Suggestions

For future research, if resources allow, researchers can consider the following aspects:

For further speed gain:

1. Activate the use of GPU for XGBoost by simply adding “tree_method” as “gpu_hist” in the hyperparameter list. This requires CUDA-capable GPUs.
2. Use multi-nodes for Optuna TPE hyperparameter tuning. This requires setting up a SQL database for storing trial history. Ideally, the SQL database should be setup locally in the HPC clusters. Web-based SQL was tried out and not recommended due to latency introduced with multi-node reading and writing simultaneously on the web.
3. Implement pruning for Optuna TPE hyperparameter tuning. It is a mechanism to trim off unpromising trials. In our exploration, when pruning is executed without cross-validation, Optuna TPE evaluated 200 trials in 1 single node within the lightning speed of 1H22M (instead of >12H) and found a hyperparameter set that yielded the highest search score.

The downside is that, to integrate cross-validation with pruning, it may take some time to customize the codes.

4. From the examination of the learning curves, we do not see signs of overfitting, but early-stopping can be considered if one would like to save time by not using a huge ensemble size and is satisfied with performance.

For potential performance gain:

1. Experiment feature selection and hyperparameter into a pipeline, similar to USGS' 2020 approach. A python package called Shap-hypetune is available for XGBoost to be used with Boruta/Recursive Feature Elimination/Recursive Feature Addition as feature selection and HyperOpt Bayesian Search as hyperparameter tuning. This allows the search for optimal number of features while searching for the optimal hyperparameters.
2. Investigate an approach to define and identify outliers in Satellite data, as boosting methods in general are sensitive to outliers.
3. Consider experimenting with resampling to address the issue of imbalanced data, though some found that resampling did not outperform the use of XGBoost hyperparameter scale_pos_weight in their tests (Wade, 2020).

10. Acknowledgement

First and foremost, I would like to thank Dr. Todd Hawbaker, my supervisor at USGS during my student contract, for allowing me to use the Alaska training dataset* for this project. Without his kindness, guidance and support at USGS, this project would not have been born.

Second, I would like to thank Research Computing Center at University of Colorado, Boulder and XSEDE for their computing resources** and wonderful help desk support.

Last but not least, I would like to thank the three awesome and talented professors on my committee: Dr. Yaning Liu (Chair), Dr. Erin Austin and Dr. Jan Mandel. I cannot thank them enough for their thoughtful feedbacks and encouragement.

**The dataset is a USGS property and is not available to public at the time of this paper. For future data release, please follow the USGS website and future publications.*

***All runs in this project was conducted on the Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University*

Works Cited

- Adebayo, S. (2020, November 16). *How the Kaggle Winners Algorithm XGBoost Algorithm works*. Retrieved from Dataaspirant: <https://dataaspirant.com/xgboost-algorithm/#t-1605502097067>
- Baheti, P. (2022, March 19). *The Train, Validation, and Test Sets: How to Split Your Machine Learning Data*. Retrieved from V7labs: <https://www.v7labs.com/blog/train-validation-test-set>
- Bajaj, A., & ES, S. (2022, March 4). *NeptuneBlog*. Retrieved from Hyperparameter Tuning in Python: a Complete Guide: <https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>
- Bressler, N., & Tannor, S. (2021, June 2). *Training, Validation and Test Sets: What are the differences?* Retrieved from DeepChecks: <https://deepchecks.com/training-validation-and-test-sets-what-are-the-differences/>
- Brownlee, J. (2017, July 24). *How Much Training Data is Required for Machine Learning*. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/much-training-data-required-machine-learning/>
- Brownlee, J. (2020, August 15). *Data Leakage in Machine Learning*. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/data-leakage-machine-learning/>
- Brownlee, J. (2020, January 6). *ROC Curves and Precision-Recall Curves for Imbalanced Classification*. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/>
- Chaturvedi, M. (2021, June 20). *Story of Gradient Boosting: How it Evolved Over Years*. Retrieved from AnalyticsIndiaMag: <https://analyticsindiamag.com/story-of-gradient-boosting-how-it-evolved-over-years/>
- Chou, S.-Y. (2020, April 25). *Compute the AUC of Precision-Recall Curve*. Retrieved from Github: <https://sinyi-chou.github.io/python-sklearn-precision-recall/>
- Choueiry, G. (n.d.). *Understand Forward and Backward Stepwise Regression*. Retrieved from Quantifying Health: <https://quantifyinghealth.com/stepwise-selection/>
- Chung, K. (2019, Dec 27). *Demystify Modern Gradient Boosting Trees From Theory to Hands-On Examples*. Retrieved from https://everdark.github.io/k9/notebooks/ml/gradient_boosting/gbt.nb.html#43_second-order_loss_approximation
- Draelos, R. (2019, February 23). *Measuring Performance: AUC(AUROC)*. Retrieved from Glass Box Machine Learning and Medicine: https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/?ref=morioh.com&utm_source=morioh.com
- Draelos, R. (2019, March 2). *Measuring Performance: AUCPR and Average Precision*. Retrieved from Glass Box Machine Learning and Medicine: <https://glassboxmedicine.com/2019/03/02/measuring-performance-auprc/>
- Flatley, M. (2021). *AUROC: Area Under the Receiver Operating Characteristic*. Retrieved from Morioh: <https://morioh.com/p/189aefce710f>
- Great Learning Team. (2020, June 6). *What is Gradient Boosting and how is it different from AdaBoost?* Retrieved from GreatLearning: <https://www.mygreatlearning.com/blog/gradient-boosting/#sh3>

- Hawbaker, T. J., Vanderhoof, M. K., Schmidt, G. L., Bael, Y.-J., Picotte, J. J., Takacs, J. D., . . . Dwyer, J. L. (2020). The Landsat Burned Area algorithm and products for the conterminous United States. *Remote Sensing of Environment*, 19.
- Hawbaker, T., Vanderhoof, M., Beal, Y.-J., Takacs, J., Schmidt, G., Falgout, J., . . . Dwyer, J. (2017). Mapping burned areas using dense time-series of Landsat data. *Remote Sensing of Environment*, 504-522.
- Homola, D. (2015, May 8). *BorutaPy*. Retrieved from DanielHomola: <https://danielhomola.com/feature%20selection/phd/borutapy-an-all-relevant-feature-selection-method/>
- Hug, N. (2019, June 1). *Understanding Gradient Boosting as a gradient descent*. Retrieved from NicolasHug: http://nicolas-hug.com/blog/gradient_boosting_descent
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning*. Springer.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer.
- Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2012, December 18). Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, pp. 1-21.
- Koehrsen, W. (2018, June 24). *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. Retrieved from Towards Data Science: <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>
- Kordeczka, A. (2018, March 13). Retrieved from Boruta - modern dimension reduction algorithm: http://rstudio-pubs-static.s3.amazonaws.com/369273_87ccc31e36c44bb886a5dfbf5865bb1c.html
- Kurama, V. (2020). *Gradient Boosting In Classification: Not a Black Box Anymore*. Retrieved from PaperspaceBlog: <https://blog.paperspace.com/gradient-boosting-for-classification/#:~:text=Let%20us%20look%20at%20some,be%20time%20and%20memory%20exhaustive.>
- Lundberg, S. (2018, April 17). *Interpretable Machine Learning with XGBoost*. Retrieved from Towards Data Science: <https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27>
- Machine Learning Mastery*. (2016, August 17). Retrieved from A Gentle Introduction to XGBoost for Applied Machine Learning: <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>
- Machine Learning Mastery. (2020, July 21). *XGBOOST Math Explained - Objective function derivation & Tree Growing | Step By Step*. Retrieved from YouTube: <https://www.youtube.com/watch?v=iBSMdFJ6Iqc>
- Mazzanti, S. (2020, Mar 17). *Boruta Explained Exactly How You Wished Someone Explained to You*. Retrieved from Towards Data Science: <https://towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a>
- Mendekar, V. (2021, Feb). *Machine Learning – it's all about assumptions*. Retrieved from KDnuggets: <https://www.kdnuggets.com/2021/02/machine-learning-assumptions.html#:~:text=XGBoost,-source&text=source-,Assumptions%3A,each%20variable%20has%20ordinal%20relation>

- MTBS. (n.d.). *Mapping Methods*. Retrieved from MTBS: <https://www.mtbs.gov/mapping-methods>
- MTBS. (n.d.). *Project Overview MTBS*. Retrieved from <https://www.mtbs.gov/project-overview>
- Optuna Developers. (2018). *Optuna.samplers.TPESampler*. Retrieved from Optuna: <https://optuna.readthedocs.io/en/stable/reference/generated/optuna.samplers.TPESampler.html#optuna.samplers.TPESampler>
- Samudrala, A. (2018, August). *Unveiling Mathematics Behind XGBoost*. Retrieved from KDnuggets: <https://www.kdnuggets.com/2018/08/unveiling-mathematics-behind-xgboost.html>
- sauravkaushik8. (2016, December 1). *Introduction to Feature Selection methods with an example (or how to select the right variables?)*. Retrieved from AnalyticsVidhya: <https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/>
- SKlearn developer. (2022). *Average Precision Score*. Retrieved from sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html
- Starmer, J. (2021, April 25). *Decision and Classification Trees, Clearly Explained*. Retrieved from YouTube: https://www.youtube.com/watch?v=_L39rN6gz7Y&t=0s
- Steen, D. (2020, September 19). *Precision-Recall Curves*. Retrieved from Medium: <https://medium.com/@douglaspssteen/precision-recall-curves-d32e5b290248>
- Tan, B. (2020, May 23). *The Intuition Behind Gradient Boosting & XGBoost*. Retrieved from TowardsDataScience: <https://towardsdatascience.com/the-intuition-behind-gradient-boosting-xgboost-6d5eac844920#d3a5>
- USGS. (n.d.). *Landsat Burned Area*. Retrieved from USGS: https://www.usgs.gov/core-science-systems/nli/landsat/landsat-burned-area?qt-science_support_page_related_con=0#qt-science_support_page_related_con
- USGS. (n.d.). *Landsat Collection 1 Level-3 Burned Area Science Product*. Retrieved from USGS: <https://www.usgs.gov/landsat-missions/landsat-collection-1-level-3-burned-area-science-product>
- USGS. (n.d.). *What are the band designations for the Landsat satellites?* Retrieved from USGS: <https://www.usgs.gov/faqs/what-are-band-designations-landsat-satellites>
- Vanderhoof, M. K., Brunner, N., Bael, Y.-J. G., & Hawbaker, T. J. (2017, July 20). Evaluation of the U.S. Geological Survey Landsat Burned Area Essential Climate Variable across the Conterminous U.S. Using Commercial High-Resolution Imagery. *Remote Sensing*, p. 743.
- Wade, C. (2020). *Hands-On Gradient Boosting with XGBoost and scikit-learn*. Birmingham, UK: Packt Publishing.
- Wang, B. (2020, April 26). *Gradient Tree Boosting: XGBoost vs. LightGBM vs. CatBoost (Part 2)*. Retrieved from Medium: <https://beverly-wang0005.medium.com/gradient-tree-boosting-xgboost-vs-lightgbm-vs-catboost-part-2-275525458968>
- Wikipedia. (n.d.). *Receiver Operating Characteristic*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- XGBoost. (n.d.). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/XGBoost>
- XGBoost Developer. (2022). *XGBoost Parameters*. Retrieved from XGBoost: <https://xgboost.readthedocs.io/en/stable/parameter.html>
- XGBoost Developers. (2021). *Python API*. Retrieved from dmlc XGBoost: https://xgboost.readthedocs.io/en/latest/python/python_api.html

- XGBoost Developers. (2022). *Introduction to Boosted Trees*. Retrieved from XGBoost: <https://xgboost.readthedocs.io/en/stable/tutorials/model.html>
- Zachdj. (2019, October 9). *What is the difference between gradient descent and gradient boosting? Are they interdependent on each other in any way ?* Retrieved from StackExchange: <https://datascience.stackexchange.com/questions/61501/what-is-the-difference-between-gradient-descent-and-gradient-boosting-are-they>
- Ziel, R. (2019, April). *Alaska's Fire Environment: Not An Average Place*. Retrieved from International Association of Wildland Fire: <https://www.iaawfonline.org/article/alaskas-fire-environment-not-an-average-place/>