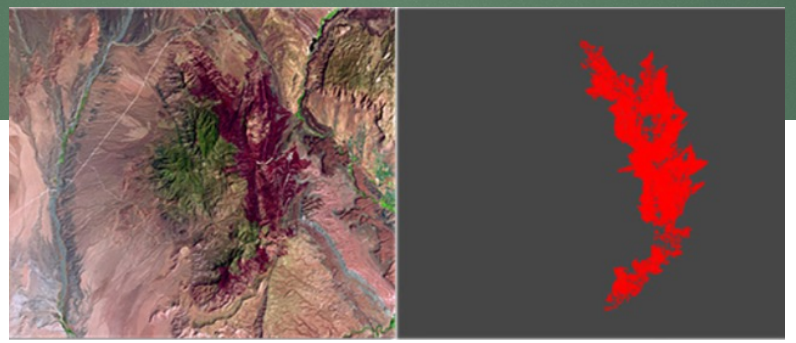
Burned Area Mapping in Alaska

by improving the 2020 Machine Learning Algorithm established by United States Geological Survey for Conterminous US

Siu-Yin Lee



University of Colorado **Denver**

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

Project Background

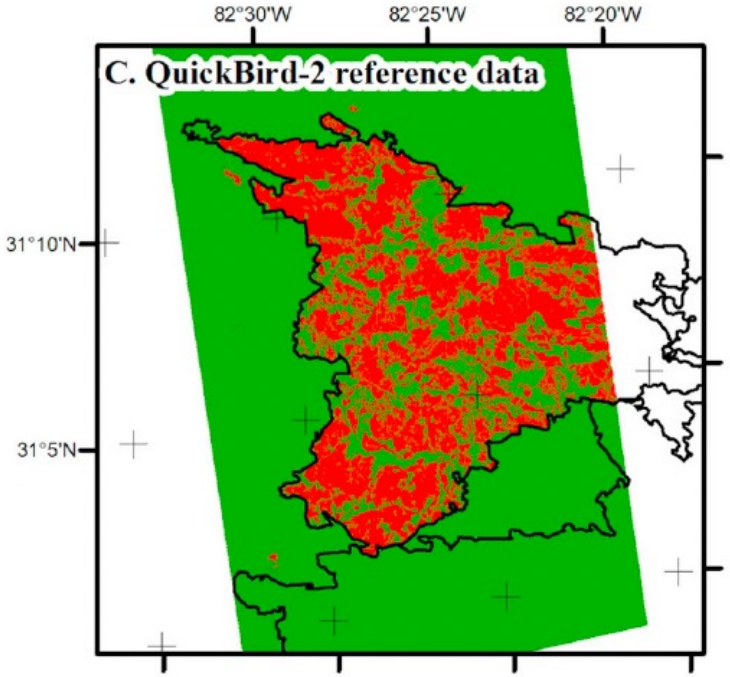
- **The importance of comprehensive fire records**
  - Fires impact our ecosystems
  - A reliable and consistent record of fire is critical to study their impact and pattern
- **Current issues regarding fire records**
  - Few agencies track fire consistently over time and space.
  - Incomplete fire database, inefficient and subjective mapping methods make the study of fire challenging
- **Current efforts in burned area archive**
  - USGS has developed a ML algorithm using satellite data for Conterminous US (CONUS) since 2017. Algorithm was revised in 2020 and has benefitted many fire research communities.
  - Alaska, the next biggest US territory, is now the next step for USGS's fire mapping archive. This project is inspired by, and partially belongs to USGS's burned area product expansion

Project Motivation

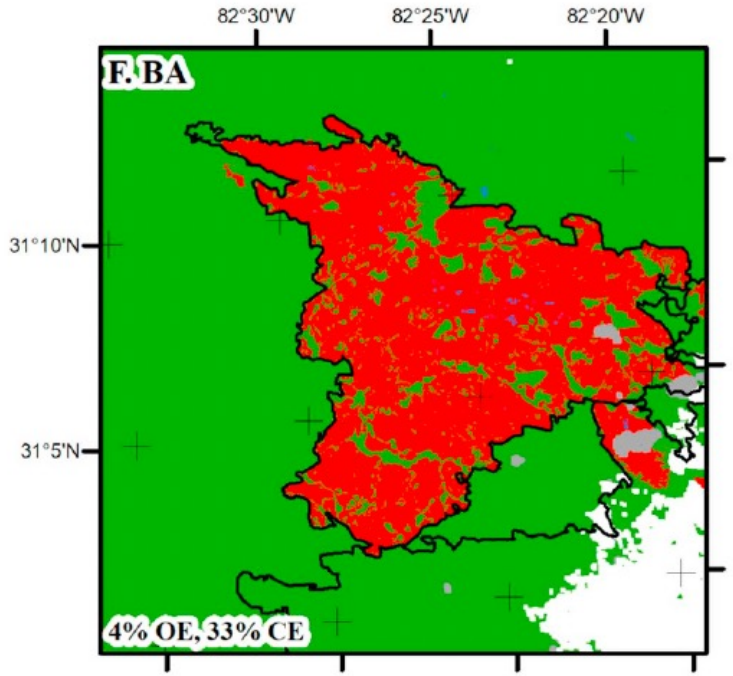
- **Train a machine learning model specifically using Alaska data**
  - Alaska has unique climate characteristics compared to CONUS. Eg. Temperature, sunlight
- **Modify the 2020 USGS algorithm to lower error rates**
  - The 2020 algorithm's error rates suggest room for improvement. (More on next slide)
- **Speed up algorithm training and prediction efficiency**
  - As satellite imagery technology evolves, the availability and volume of data are also rising tremendously.
    - Eg: ARD format data has increased data volume by 340%
    - The recent release of the Harmonized Landsat Sentinel-2 data will full global imagery every 2-3 days instead of 16 days.
  - With the upcoming high-volume data collections, we need more efficient machine learning algorithm

More on 2020 USGS Algorithm Error Rates

Ground Truth:
Burned Area in Florida in 2007



2020 CONUS Algorithm:
Mapping Example



When validating the Algorithm with high-resolution dataset, in average:

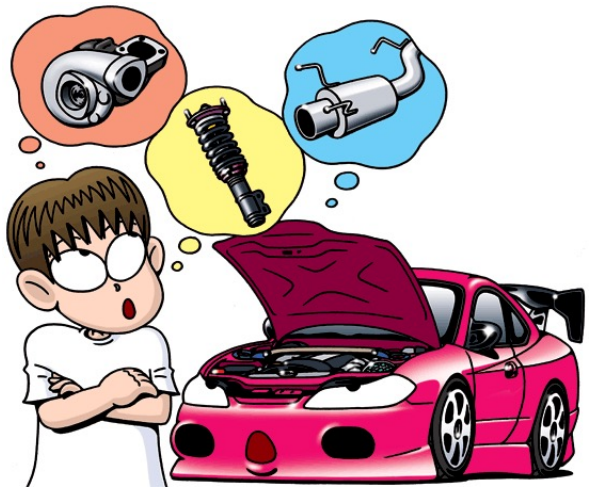
Omission Error:
= 1 - Recall
= 19%

Commission Error:
= 1 - Precision
= 41%

Research Question

Can we find a more **accurate** and **efficient** way to classify burned area in Alaska than the 2020 USGS algorithm?

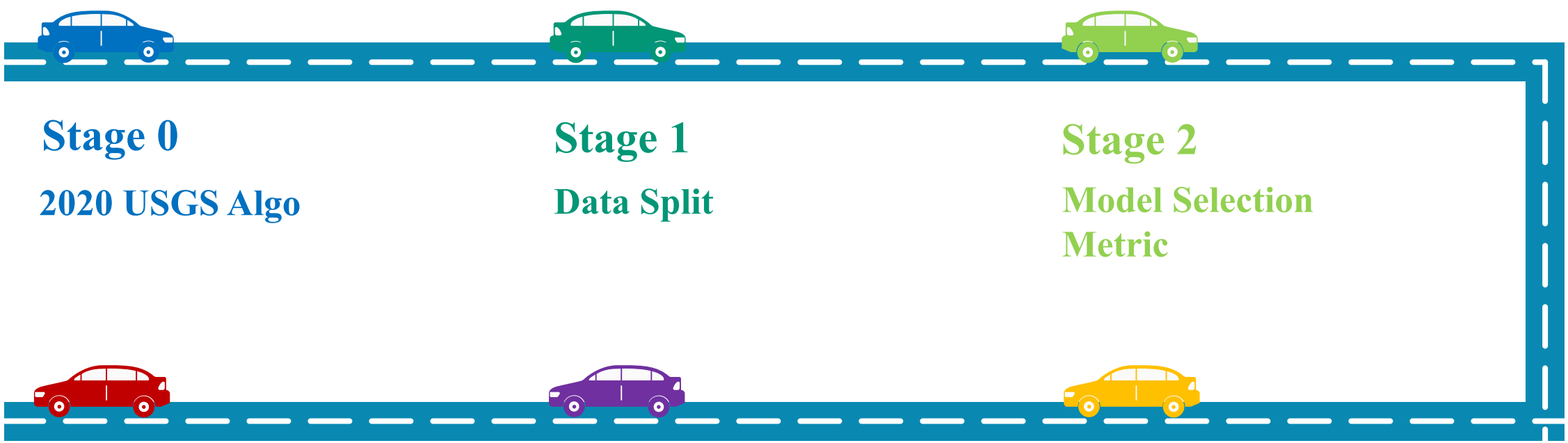
Project Methodology



Like a car tuning process:

- Identify potential issues in the 2020 algorithm that may cause the high error rates and offer suggestions for improvement.
- Modify certain components of the model and see if the modification improves performance..

Modification Roadmap



Stage 0

2020 USGS Algo

Stage 1

Data Split

Stage 2

Model Selection
Metric

Stage 5

Hyperparameter Tuning

Stage 4

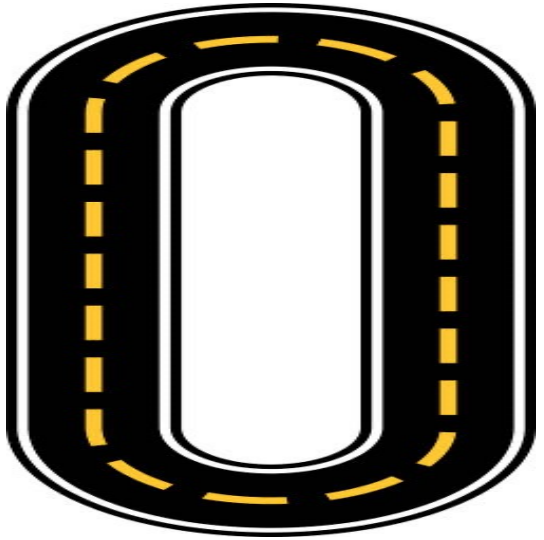
Feature Selection

Stage 3

Classifier

At each stage, I will modify only one component of the model

Then, we will do a test drive and compare performance before and after each step of changes. The changes are cumulative and will be carried through to the later stages.



2020 USGS Algorithm Details

	Aspect	2020 CONUS 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	Hyperparameter	Only 12 combinations tested

Hyper-parameter	Values tested
N_estimator	1000
Learning_rate	0.01, 0.05, 0.1
Max_depth	1, 3, 5, 7

Modification Overview

Stage	Aspect	2020 USGS	Modification
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

Alaska Dataset for this project

- 886,792 datapoints
- Every data point = a pixel from the collection of Alaska satellite images that was sampled and processed by the USGS science team.
- 94.6% are unburned datapoints and 5.4% are burned datapoints
- The fire date range of these samples ranges from 2000-2017
- **Response Variable:** Fire (binary), 1 = burned, 0 = unburned. Label comes from MTBS data and thus are mostly generated by the MTBS analysts' visual inspections.
- **Predictor Variables:** 133 Satellite Spectral Indices (Continuous). They indicate land surface conditions, reference conditions and change metrics of the environment.

HPC Clusters and Validation Data

- HPC Clusters

All runs in this project utilizes the SHAS partition on the Summit supercomputer, a HPC system supported by NSF, CU-Boulder and Colorado State University

The SHAS partition consists of 24 cores/nodes, 4.84 GB Ram/core

- Validation Data set

High Resolution and Separate Validation Dataset for Alaska was not available or accessible at the time of this project.

For final model evaluation, 20% of our dataset is held up as final validation set to benchmark performance

Now, let our journey begin…

Step 1: Fix Data Leakage

Step 1: Fix Data Leakage

Goal of Predictive model:

Train a model to make accurate prediction on unseen data. To evaluate a model's generalization ability on unseen dataset, a common technique is to hold up a portion of the data for final evaluation.

What is Data Leakage?

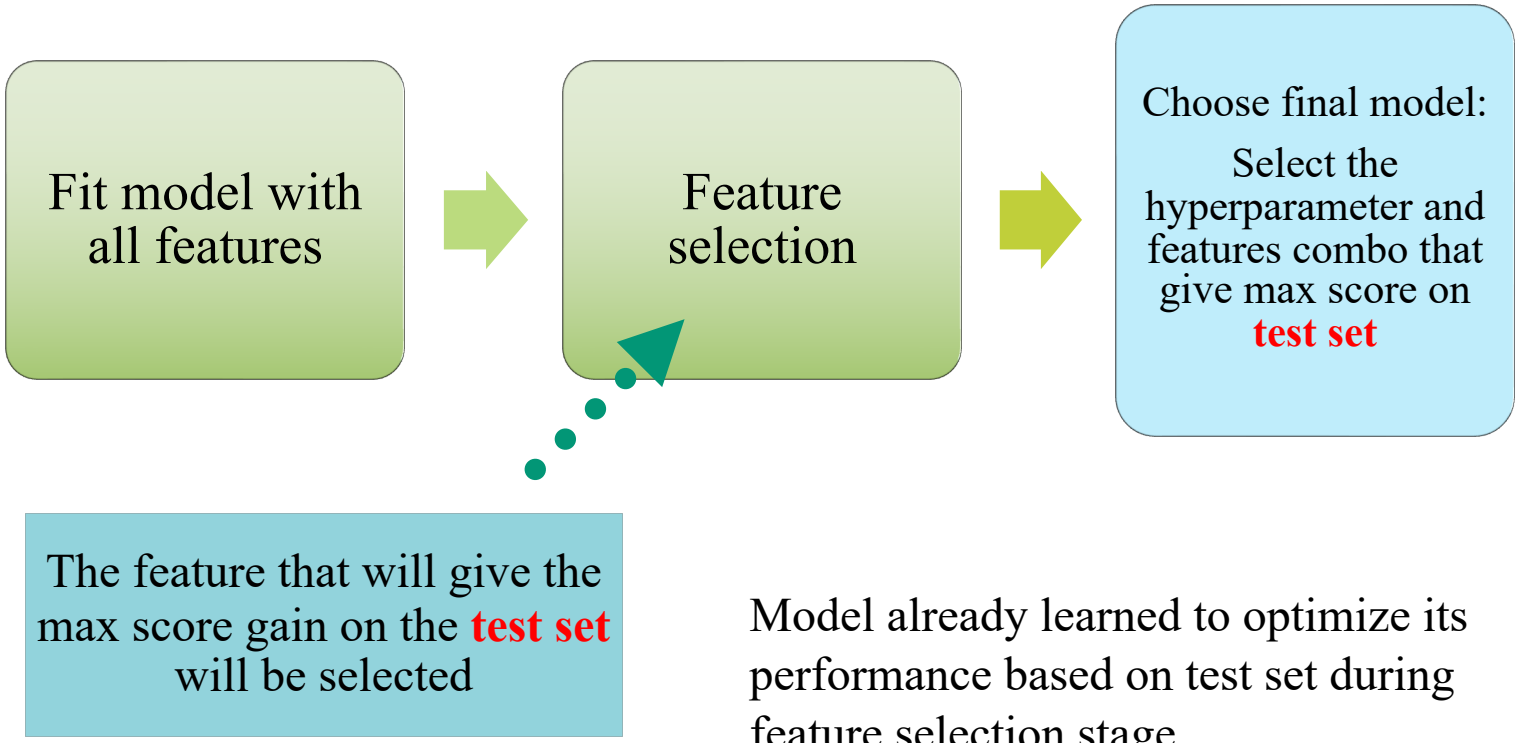
Data leakage happens when your model, during the training process, is exposed to information it is not supposed to know. It often happens unintentionally and is facilitated by the data preparation process.

Why is Data Leakage a problem?

- We end up not being able to evaluate the model's TRUE performance on unseen data.
- Very often, we overestimate the model's performance until we realize the sad truth about the model's real capability during deployment.
- We may end up choosing suboptimal solution that could have been outperformed by a leakage-free model

Where is the Data Leakage in the 2020 USGS Model?

For each of the 12 hyperparameter combinations:



Model already learned to optimize its performance based on test set during feature selection stage.
No unseen data used to evaluate final performance

STEP 1: FIX DATA LEAKAGE

Step 1 Modification

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

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

STEP 1: FIX DATA LEAKAGE

Step 1 Modification

	Use Train Set	Use Test Set	Choose Model
2020 USGS Algorithm 50% Train Set: 50% Test Set	Fit model with all features	Feature selection	Select the hyperparameter and features combo that gives max score on **test set**
Modification 60% Train Set 20% Test Set; 20% CV Set	Fit model with all features	Feature selection	Select the hyperparameter and features combo that gives max avg score on **5-fold CV Set**

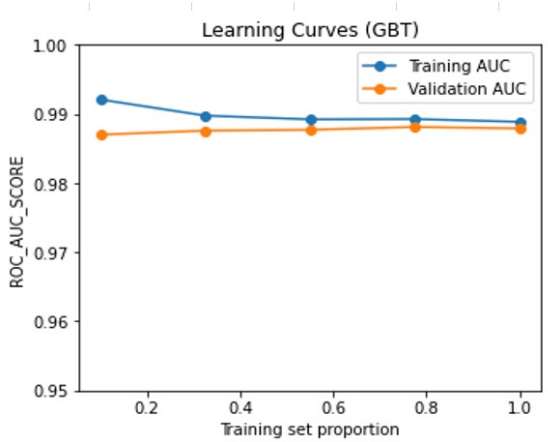
STEP 1: FIX DATA LEAKAGE

Why 5-fold CV?

- For more robust evaluation on the performance of the model by taking variance of different evaluation datasets into account
- For large dataset, 5 is a common choice as it is not too computationally exhaustive
- Empirically, 5-fold or 10-fold have been shown to yield test error estimates that suffer neither excessively high bias nor very high variance

Why 60-20-20?

- There is no fixed rule. We want a balance between the data sizes for learning and evaluation. 60-20-20 is one of the common 3-way split choice.
- During EDA, learning curve shows that when Trainset = 0.6, the test and train sets' score start to converge and stabilize.



STEP 1: FIX DATA LEAKAGE

Step One Test Run Result

	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

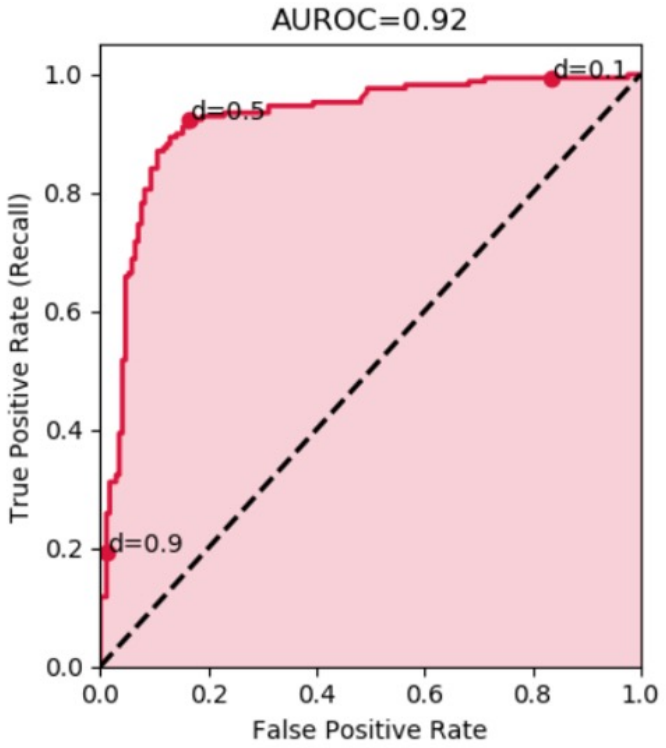
Step 2: Change Model Selection Metric

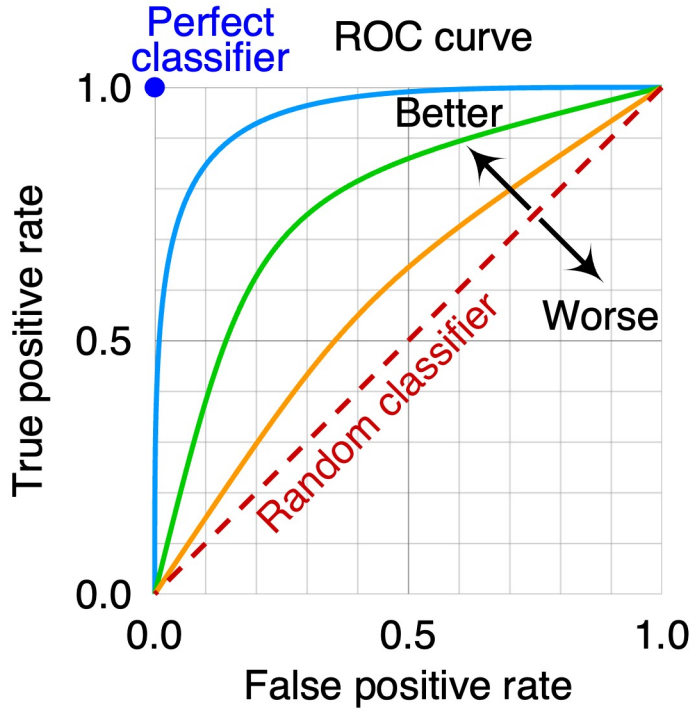
Step 2: Change Model Selection Metric

What is AUC Score?

AUC Score is the Area-Under-Curve for the ROC curve.

ROC (Receiver Operator Characteristic) curve is a probability curve plotted with True Positive Rate against False Positive Rate



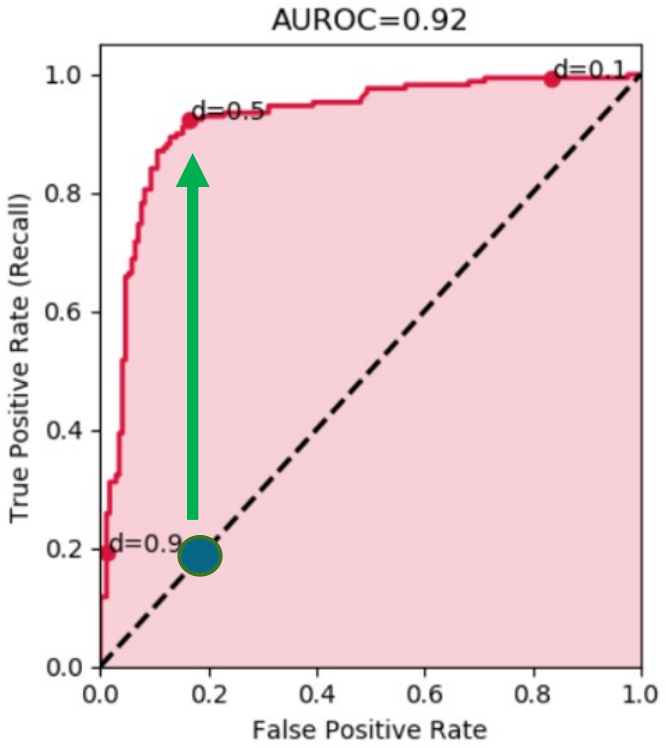


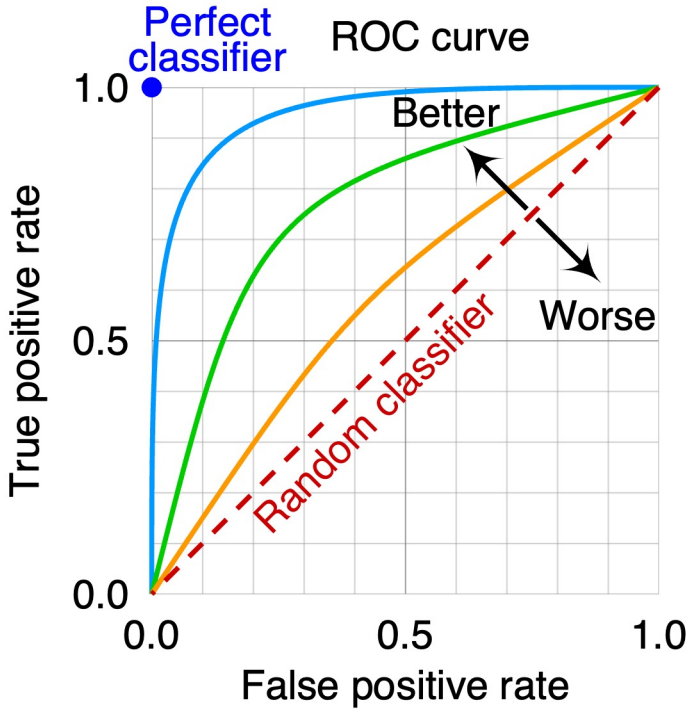
Step 2: Change Model Selection Metric

What is AUC Score? Continued

AUC measures the trade off between true positive rate (TPR) and false positive rate (FPR) at different classification decision thresholds (d in graph).

A High AUC score indicates that your model has a high True Positive Rate while being able to keep False Positive Rate low globally across different decision thresholds.





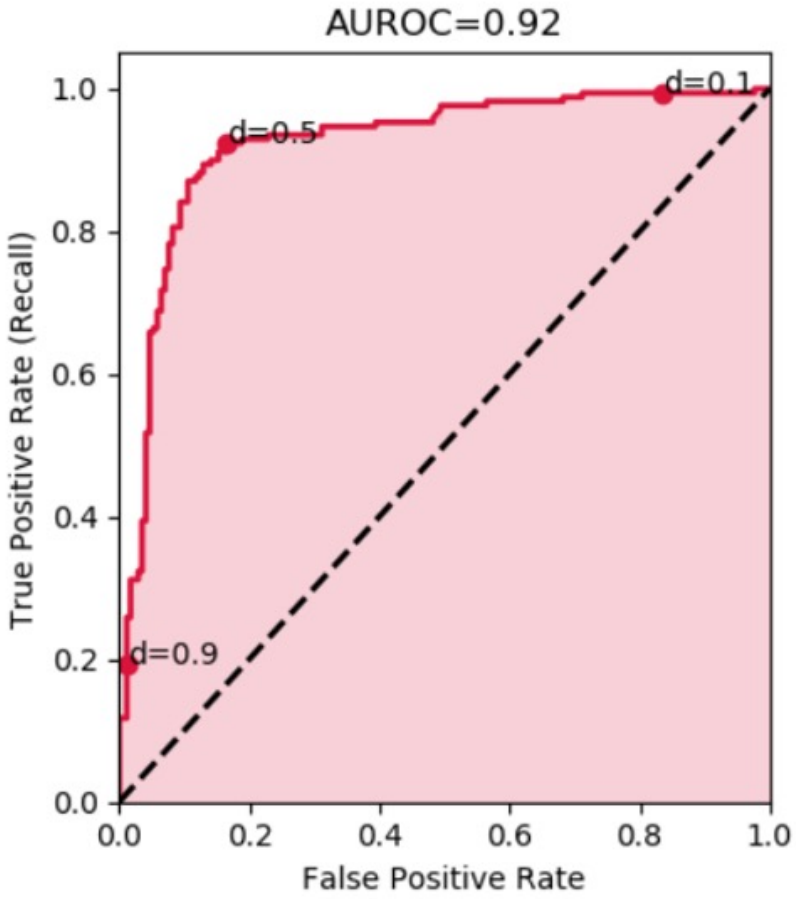
Why is using the AUC Score a problem?

In the 2020 USGS Algorithm, we chose features and hyperparameter combination that maximizes AUC Score.

It means we wanted to **maximize TPR** and **minimize FPR**.

How are TPR and FPR calculated?

$$TPR = TP / (TP + FN)$$
$$FPR = FP / (FP + TN)$$

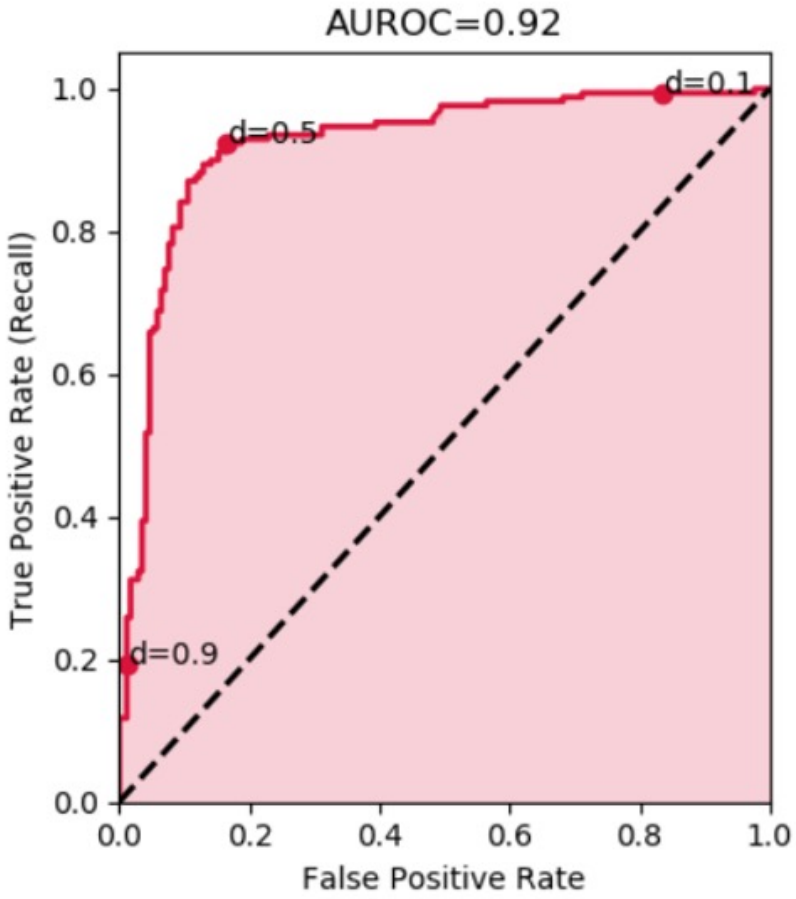


Why is using the AUC Score a problem?

How are TPR and FPR calculated?

TPR = TP / (TP + FN) ↓ FN ↑ TPR

FPR = FP / (FP + TN) ↑ TN ↓ FPR

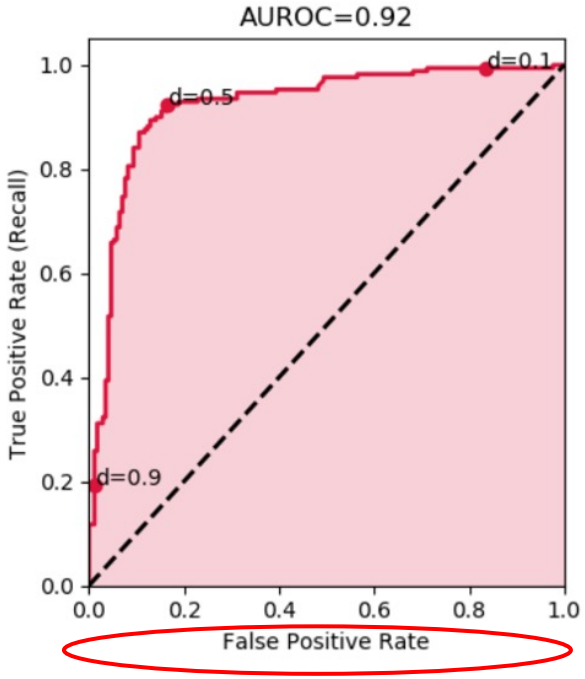


Why is using the AUC Score a problem?

How are TPR and FPR calculated?

$$TPR = TP / (TP + FN)$$ ↓ FN ↑ TPR

$$FPR = FP / (FP + TN)$$ ↑ TN ↓ FPR



The problem is:

Positive = burned area ; Negative = unburned area
The data is hugely imbalanced. 95% of data point is unburned.

If we use AUC, so long as our model can classify a huge number of unburned points as unburned, the model will then have a low FPR and will appear to be a “good” model, which can be deceiving as the model can be quite bad at classifying the positive class.
What we care more is the burned area, the signal. We don’t really care that much about high TN
When there is a severe class imbalance in dataset, the use of AUC score for model evaluation can be overly optimistic.

STEP 2: CHANGE MODEL SELECTION METRIC

Step 2 Modification

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

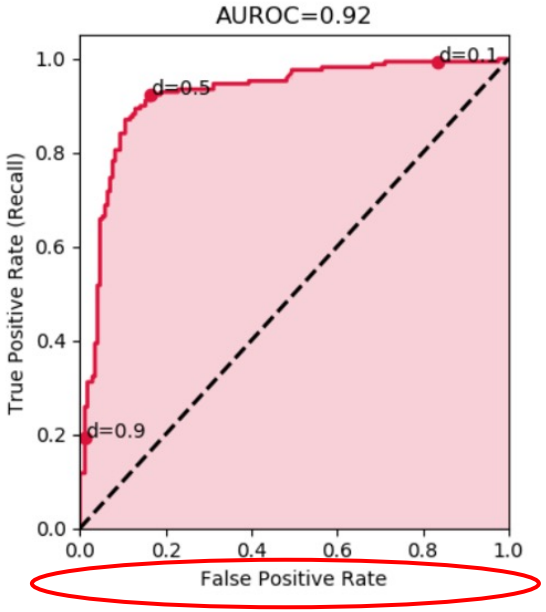
	Aspect	Step 2 Algorithm
1	Data Split	60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	Average Precision Score
3	Classifier	Gradient Boost
4	Feature Selection	Forward Selection
5	Hyper-parameter	Only 12 combinations tested

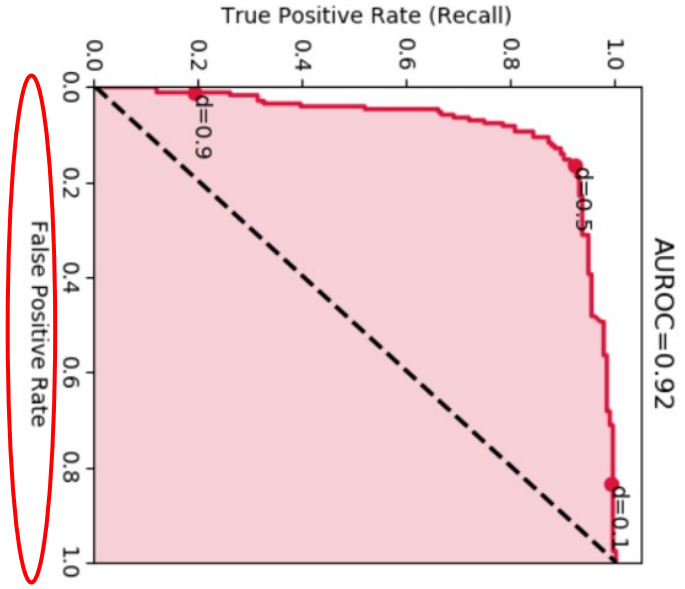
STEP 2: CHANGE MODEL SELECTION METRIC

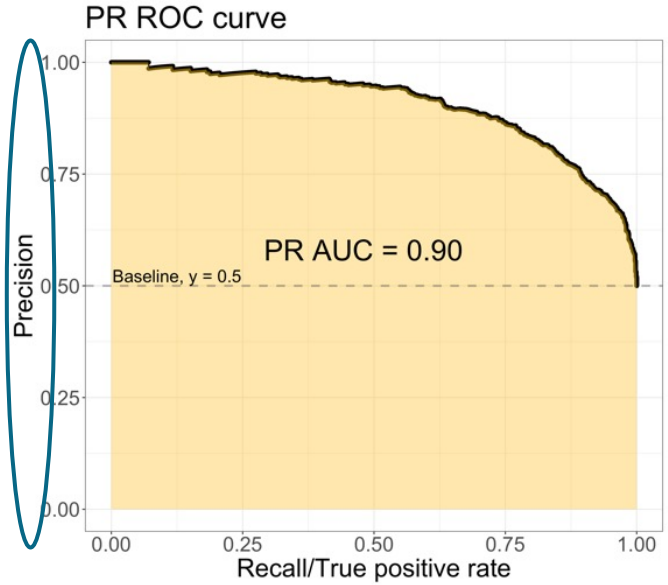
Why Average Precision?

Average Precision is similar to AUC
It measures the area under the curve for the Precision Recall Curve

Precision = TP/ (TP+FP)

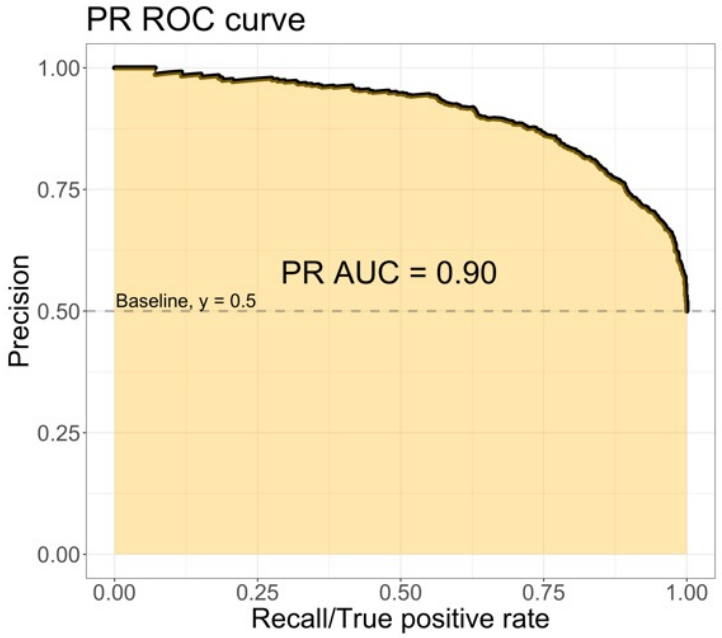


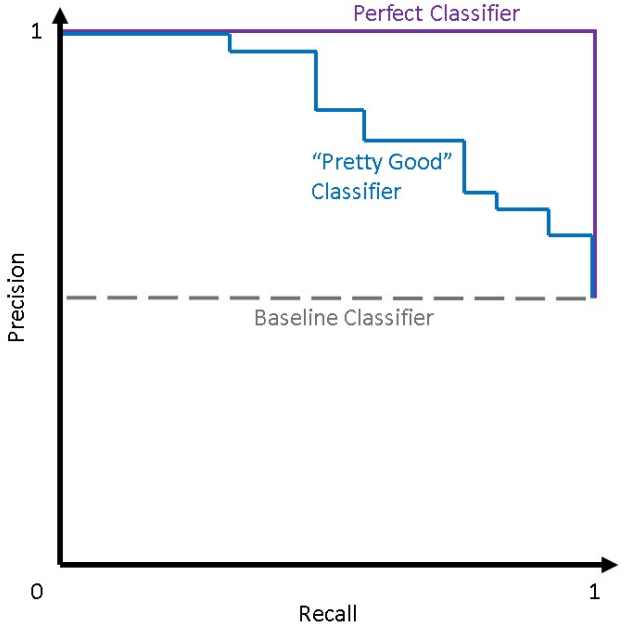




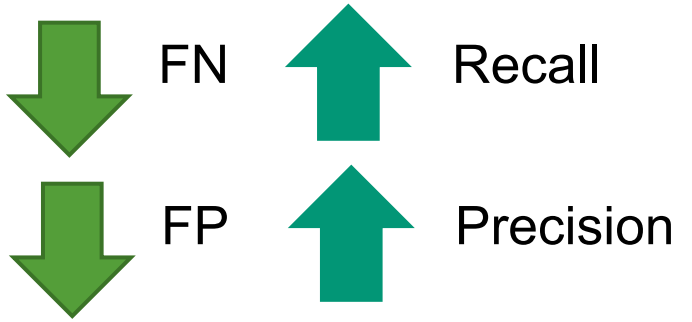
STEP 2: CHANGE MODEL SELECTION METRIC

Why Average Precision?



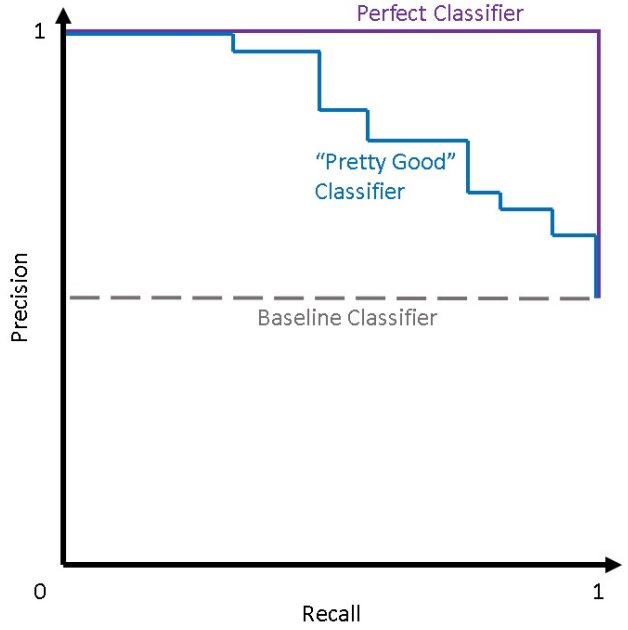


Precision = TP/ (TP+FP)
Recall = TPR = TP/ (TP+FN)



STEP 2: CHANGE MODEL SELECTION METRIC

More about Average Precision



For a random classifier, AP = $\frac{\#\ positive\ class}{\#\ total\ datapoints}$

For balanced dataset, random classifier's AP = 0.5, like AUC

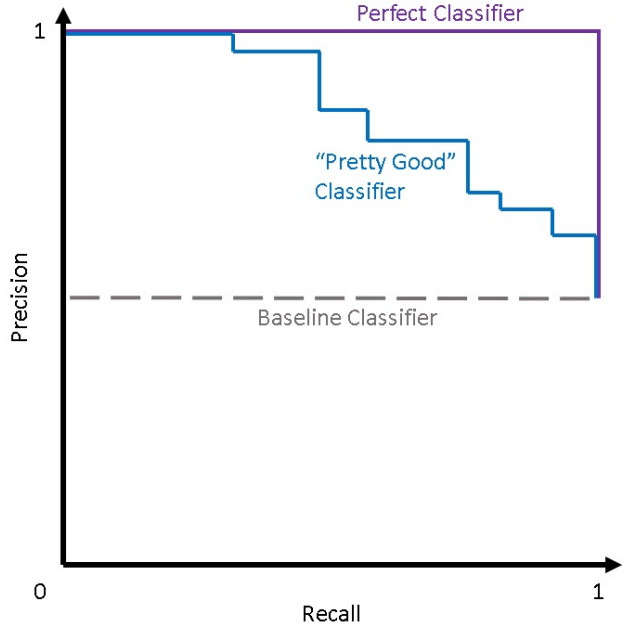
For imbalanced dataset, a random classifier's AUC score is always 0.5 regardless of class proportion. But for AP, if the AP score is bigger than P, the proportion of positive class, we know that the classifier is useful.

Why is a random classifier's AUC = 0.5, and AP = positive class proportion?

We can simply work out the math by assuming p as proportion of positive class and q as the probability of randomly classifying something as positive. If we plug p and q into the TPR, FPR, Recall, Precision formula, we will see it.

STEP 2: CHANGE MODEL SELECTION METRIC

More about Average Precision



AP uses rectangular approximation to calculate the area under the PR curve

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

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. Depending on the calculation method used, the metric that measures the area under the PR curve may have different names (eg. AUCPR)

STEP 2: CHANGE MODEL SELECTION METRIC

Step Two Test Run Result

	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

Step 3: Change Classifier

Step 3: Change Classifier

What is Gradient Boosted Trees

- Tree
- Boosting
- Gradient

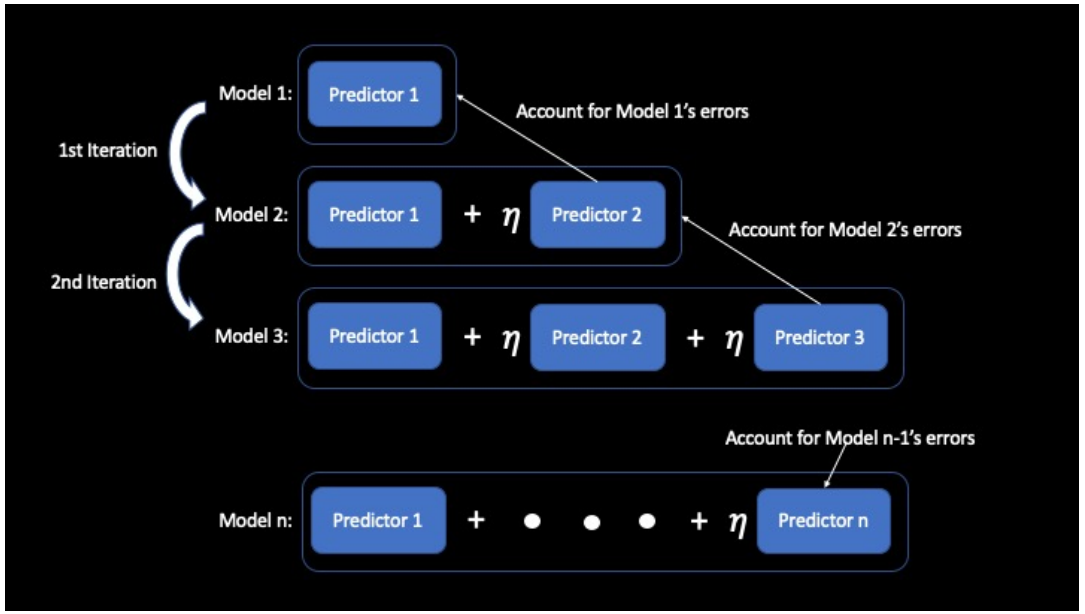
In the 2020 USGS algorithm, Gradient Boosted Trees was used as the model's classifier.

As the name of Gradient Boosted Trees suggests, this algorithm can be understood by 3 key elements: Trees, Boosting and Gradient

Step 3: Change Classifier

What is Gradient Boosting in general

- Popular ML algorithm whose generalized form was formulated by Jerome Friedman in 1999



Crux of Gradient Boosting:

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."

Step 3: Change Classifier

What is Gradient Boosted Trees

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 the most

The Tree keeps on splitting the data until no more improvement can be brought by further split, or a certain stopping criterion is met

Step 3: Change Classifier

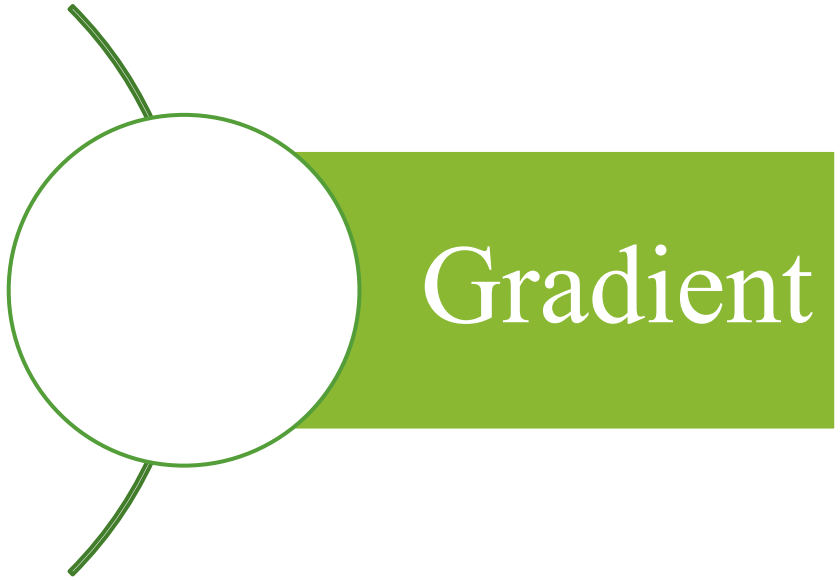
What is Gradient Boosted Trees

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

Step 3: Change Classifier

What is Gradient Boosted Trees



The key characteristics of Gradient Boost is that it minimizes the loss function of the model by performing gradient descent with its weak learners

Gradient descent is a first-order iterative optimization algorithm extensively used in Machine Learning.

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

To briefly illustrate the similarity of Gradient Boost and Gradient Descent mathematically, we consider the loss function:

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

<u>In Gradient Descent:</u>

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

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

Let $\quad learning\_rate * \frac{\partial \mathcal{L}}{\partial \theta^{(m)}} = b_m$

Then $\quad \theta^{(m+1)} = \theta^{(m)} - b_m$

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

$\dots$

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

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

<u>In Gradient Boost:</u>

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:

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

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.

Step 3: Change Classifier

What is the down side 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 :

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!

STEP 3: CHANGE CLASSIFIER

Step 3 Modification

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

	Aspect	Step 3 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	Forward Selection
5	Hyper-parameter	Only 12 combinations tested

STEP 3: CHANGE CLASSIFIER

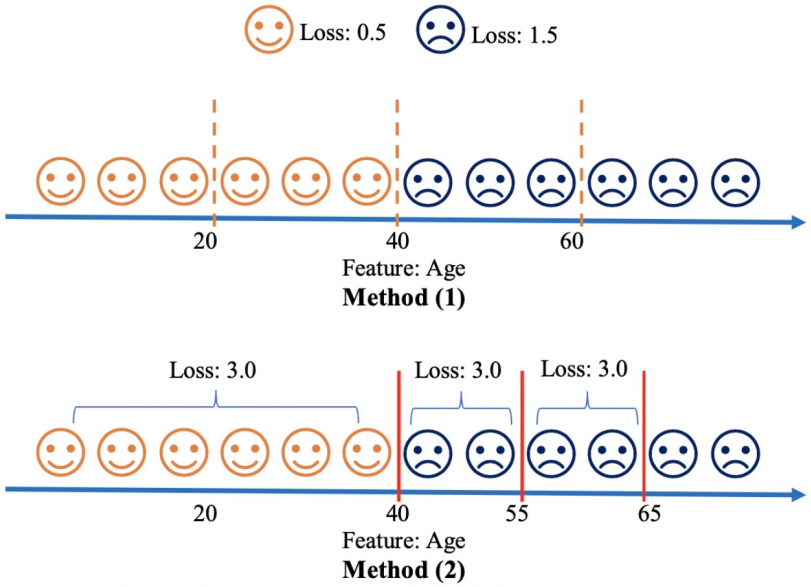
Why XGBoost?

- XGBoost = “Extreme Gradient Boosting” = “pushing computational limits” to the extreme
- Built upon the foundation of Gradient Boosting in 2014, initiated by Tianqi Chen
- API structure very similar to Gradient Boost, the shift from Gradient Boost to XGBoost only requires minor code edits
- Besides convenience, the main reason is its superb performance and efficiency.
- Winning algorithm used in many Kaggle competitions and has a large community support.
- For structured datasets, found to outperform even deep learning algorithm
- 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.**

STEP 3: CHANGE CLASSIFIER

Weighted Quantile Sketch and Parallelization

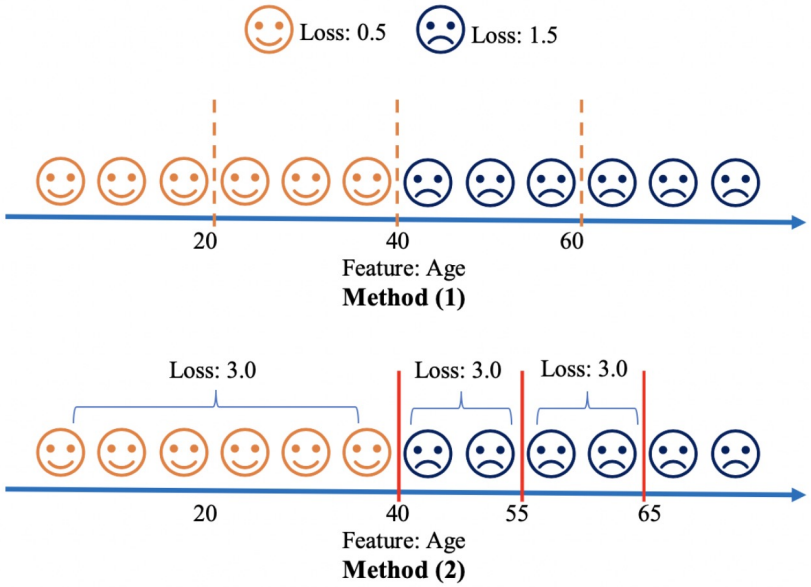
- **Weighted Quantile Sketch** makes a histogram for each feature and use the bins' boundaries as candidate split points.
- Saves time by considering far fewer split candidates as now the data are put into bins.
- Heavier weights are given to data points that are not well predicted yet/have higher loss.
- 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.
- Consider more split candidates and conduct more detailed search in areas where the model needs improvement



STEP 3: CHANGE CLASSIFIER

Weighted Quantile Sketch and Parallelization

- **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.
- XGBoost makes parallelization happen – not on the ensemble/tree level, but on the feature level.
- Eg: 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



STEP 3: CHANGE CLASSIFIER

Built-in Regularized Parameter for Objective function

To address the tendency to overfit in Gradient Boosted Trees. XGBoost has regularization parameters built into its objective function. Note: Hyperparameter ≠ Parameter

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

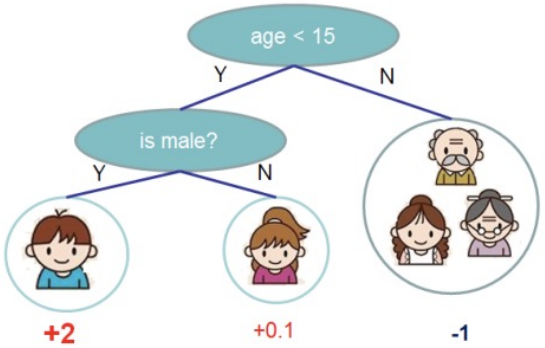
Minimize loss
encourages tree growth

Regularization controls
tree complexity

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

γ-weighted T
T = # of leaves

λ-weighted L2 norm
of the leaf weight



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

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

γ and λ are user-defined hyperparameters.

STEP 3: CHANGE CLASSIFIER

Taylor Expansion and 2nd Order derivative

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

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

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

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

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

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

where $g_i = \partial_{\hat{y}^{(t-1)}}\, l\left(y_i, \hat{y}_i^{(t-1)}\right)$ and $h_i = \partial^2{}_{\hat{y}^{(t-1)}}\, l\left(y_i, \hat{y}_i^{(t-1)}\right)$

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

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

STEP 3: CHANGE CLASSIFIER

Other Design Features that make XGBoost significantly faster

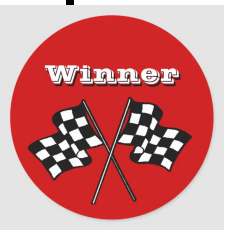
- Sparsity aware split-finding
- Cache-aware access
- Block compression and Sharding
- Etc..

- These designs are mostly about improving data reading time, computational memory and storage

STEP 3: CHANGE CLASSIFIER

Step Three Test Run Result

	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



Step 3.5: Introduce Additional Features

Step 3.5: Introduce Additional Features

- Author was asked to trained a model for Alaska without using the raw band when working at USGS
- Most likely because some o the features already used these raw band features in their calculation and these raw band features were not selected by the 2020 algorithm.
- Author decided to revisit this decision while looking at the satellite images on ArcMap based on the following reflection though the project had already been started without the raw bands:

> "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"

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)

STEP 3.5: ADD RAW BANDS

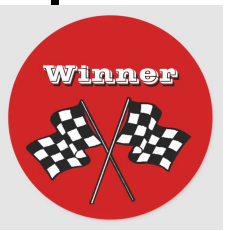
Step 3.5 Test Run Result

	Step 3 Algorithm XGBoost	**Step 3.5 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

3 out of 7 of the Raw Band:

Band 6, Band 7 and Band 1

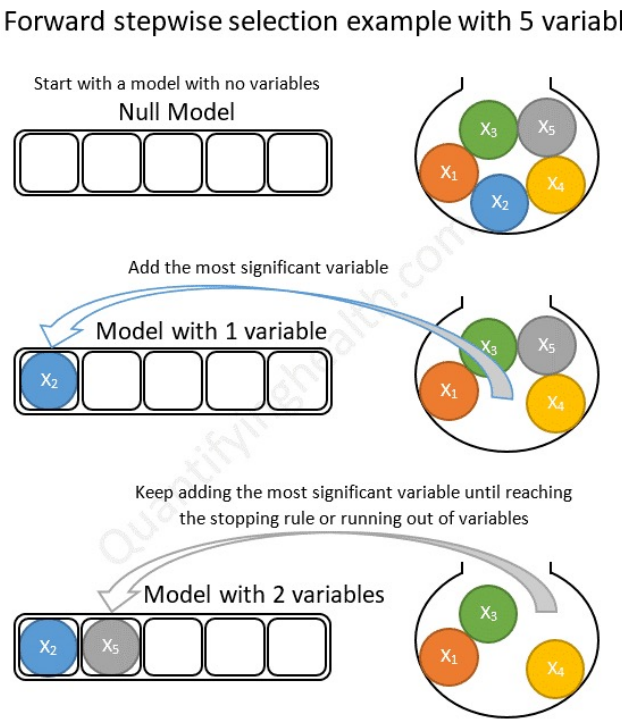
were selected by the model in the set of 15 total features



Step 4: Feature Selection Modification

Step 4: Feature Selection Modification

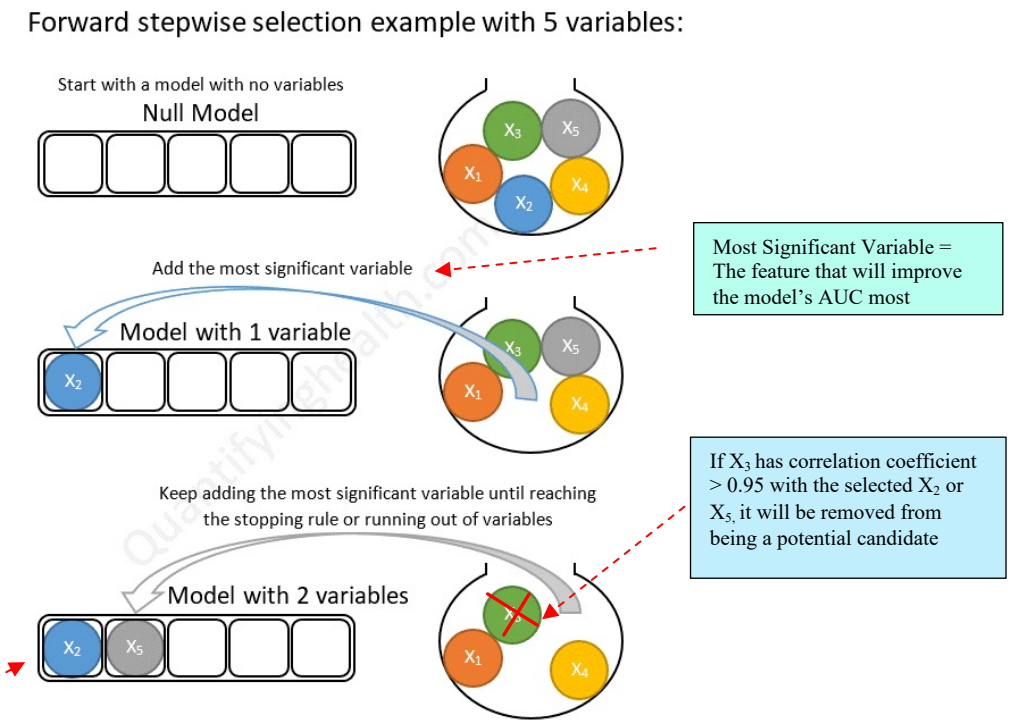
What is Stepwise Forward Selection



In the 2020 USGS algorithm, Stepwise Forward Selection was used as feature selection method.

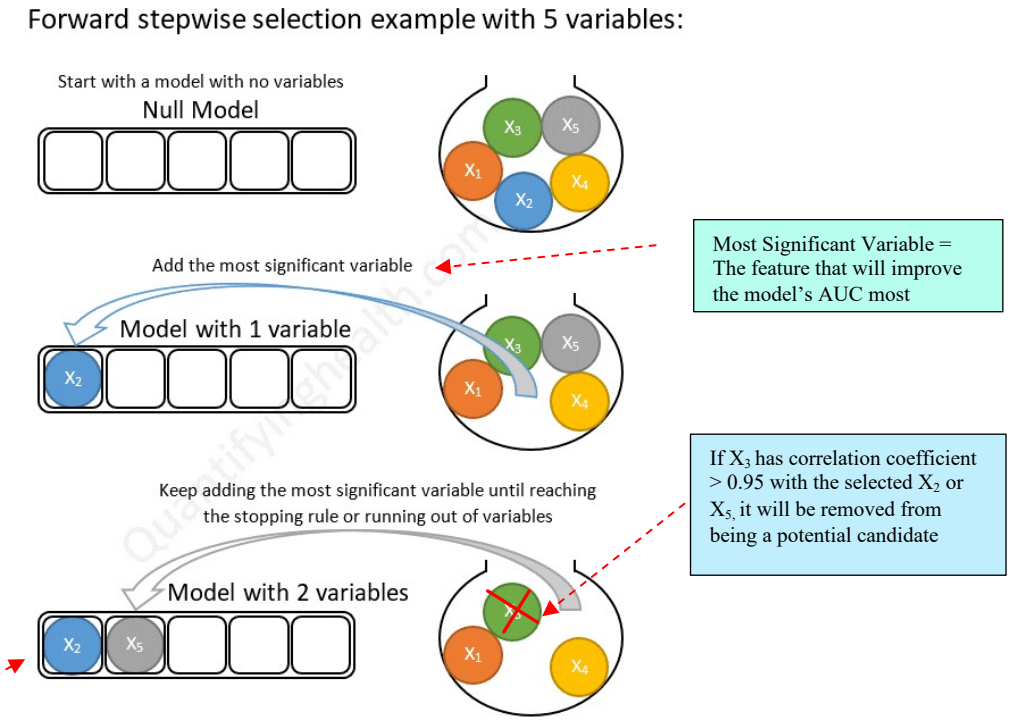
Step 4: Feature Selection Modification

What is Stepwise Forward Selection



Step 4: Feature Selection Modification

What is Stepwise Forward Selection



With 133 features:

Best Subset needs to fit 2^{133} (2^p) models
= about 1.08e+40 models

Forward Selection needs to fit $\sum_{k=0}^{15}(133 - k)$
= 2008 models
(with 15 features selected in stage 3)

Step 4: Feature Selection Modification

What is the downside of forward feature selection?

1. Greedy nature. We may miss out other better combination

2. Features are competing among themselves even though they may all have great predictive power on the response.

3. The stopping criteria may us miss potential gain beyond that stopping spot.

STEP 4: CHANGE FEATURE SELECTION METHOD

Step 4 Modification

	Aspect	Step 3 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	Forward Selection
5	Hyper-parameter	Only 12 combinations tested

	Aspect	Step 4 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	Only 12 combinations tested

STEP 4: CHANGE FEATURE SELECTION METHOD

Why Boruta?

- Statistically grounded and robust. Uses a permutation method and the binomial distribution to determine a feature's "usefulness".
- Not a greedy algorithm, and thus does not depend on the selection process's "current state".
- Does not make features compete among themselves. The competitors are the randomized versions of themselves, which are called "shadow features".
- Aims at finding all relevant features, not a minimal subset. We can better evaluate the predictive power of our features.

STEP 4: CHANGE FEATURE SELECTION METHOD

Illustration of Boruta's Mechanism

1

	age	height	weight
0	25	182	75
1	32	176	71
2	47	174	78
3	51	168	72
4	62	181	86

	income
0	20
1	32
2	45
3	55
4	61

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

2

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

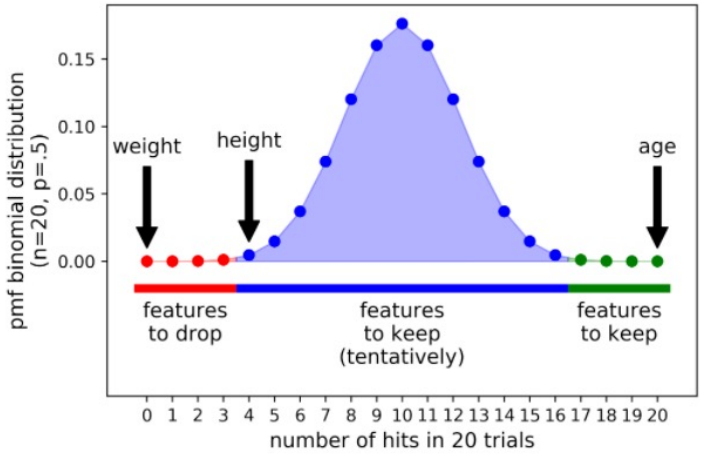
Outcome of one run

3

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

Outcome of 20 runs

4



Binomial distribution and positioning of the features

STEP 4: CHANGE FEATURE SELECTION METHOD

Step 4 Boruta Result

- 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.

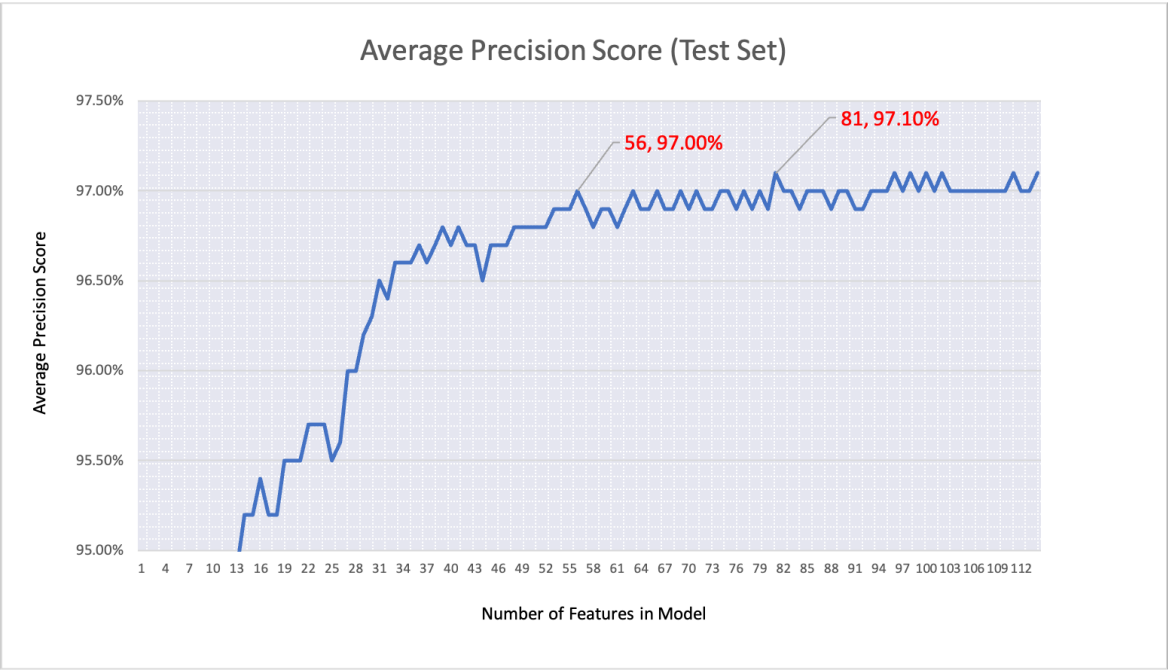
STEP 4: CHANGE FEATURE SELECTION METHOD

Feature Importance

We fit the 114 features using XGBoost to check each feature's importance in classifying burned area.

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

Examine how Average Precision score changes with each feature added into the model according to the order of feature importance



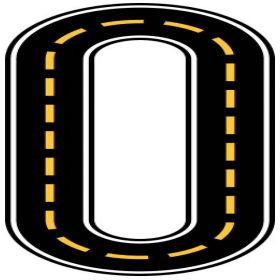
Feature Importance Thresholding vs Forward Selection

Feature Importance Thresholding method seem similar to stepwise forward selection, as both methods examine the change of score by adding each feature. Nonetheless, there are some subtle, but important, differences:

- The order of feature addition
  - FS: add feature that can improve score in the test set the most, given the current state
  - FI: add feature based on the feature's effects on prediction across all tree nodes.

- In regard to computational time, FS needs to fit $\sum_{k=0}^{s}(\mathrm{p}-\mathrm{k})$ models while FI only needs to fit P model. 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.

- Besides, with 133 model fit, we are presented with the full picture of how score changes for every single feature. Unlike FS, we are blinded beyond the stopping point.



STEP 4: CHANGE FEATURE SELECTION METHOD

Step 4 Test Run Result

	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

Step 5: Hyperparameter Tuning Modification

Step 5: Hyperparameter Tuning Modification

What is Grid Search

In the 2020 USGS algorithm, the hyperparameter tuning process was essentially a grid-search method. Even though the hyperparameter tuning process was 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.

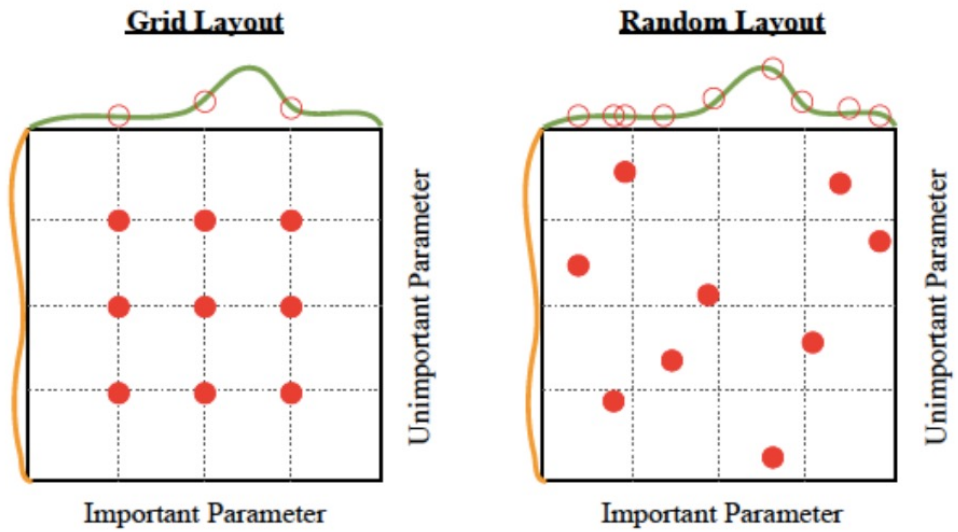
As the name suggests, grid search is a hyperparameter tuning method based on a user-defined grid. A grid is simply all the possible combinations of the user-defined values for each hyperparameter.

Step 5: Hyperparameter Tuning Modification

What is the downside of Grid Search

1. Grid-search is very computationally expensive. 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. Exponential Increase.

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.



STEP 5: CHANGE HYPERPARAMETER TUNING METHOD

Step 5 Modification

	Aspect	Step 4 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	Grid-Search Only 12 combinations tested

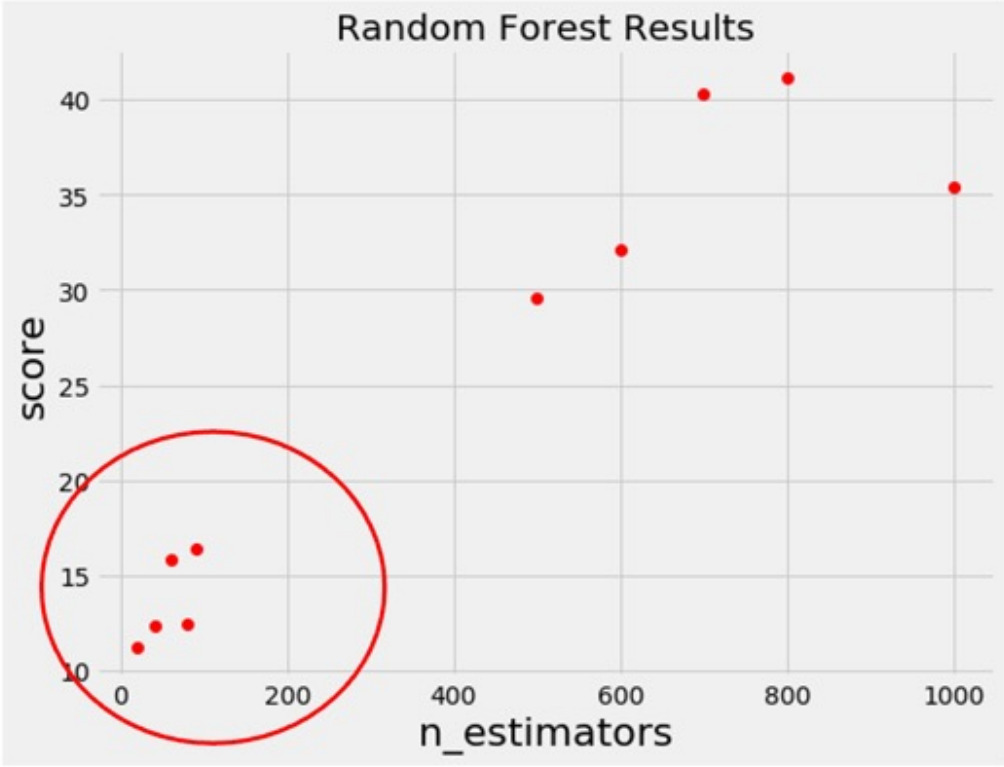
	Aspect	Step 5 Algorithm
1	Data Split	60% Train Set 20% Test Set; 20% CV Set
2	Model Selection Metric	Average Precision Score
3	Classifier	XG Boost
4	Feature Selection	Boruta + Feature Importance Thresholding
5	Hyper-parameter	Optuna Bayesian TPE 200 trials

STEP 5: HYPERPARAMETER TUNING MODIFICATION

Why Optuna 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.

The Optuna library is used to execute the TPE method due to its relatively user-friendly interface



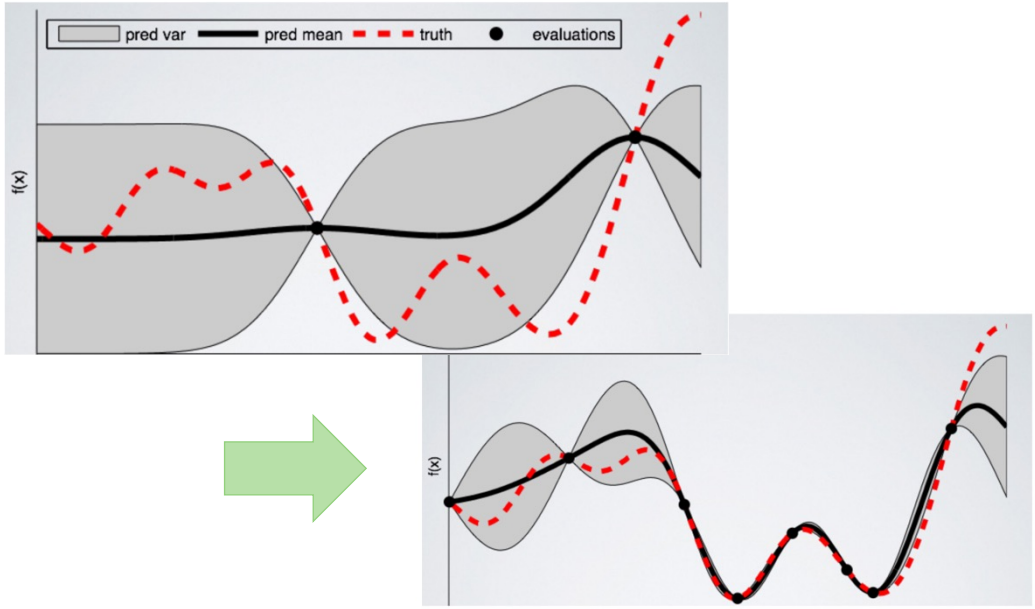
Bayesian Optimization in Brief

In essence, Bayesian hyperparameter optimization is the process of:
Building a probability model of the objective function to propose smarter choices for the next set of hyperparameters to evaluate.

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

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

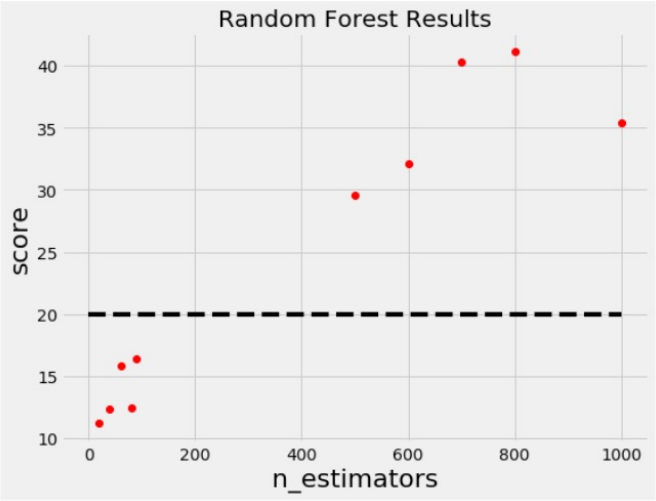
1. Build a surrogate probability model of the objective function
2. Find the hyperparameters that perform best on the surrogate
3. Apply these hyperparameters to the objective function
4. Update the surrogate model incorporating the new results
5. Repeat steps 2-4 until max iterations or time is reached



TPE in Brief

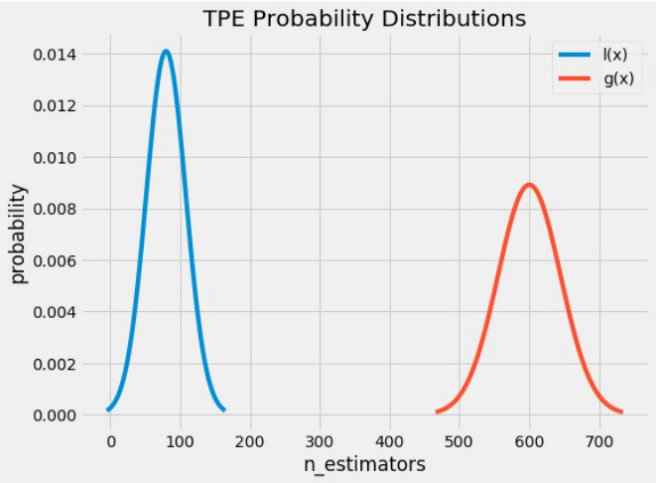
Different Bayesian hyperparameter optimization methods have different ways of building the surrogate probability model $P\ (y \mid x)$. Tree-Structured Parzen Estimator (TPE) builds its surrogate by applying the Bayes rule. Instead of modeling using $P\ (y \mid x)$, it uses $P\ (x \mid y)$, which is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

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

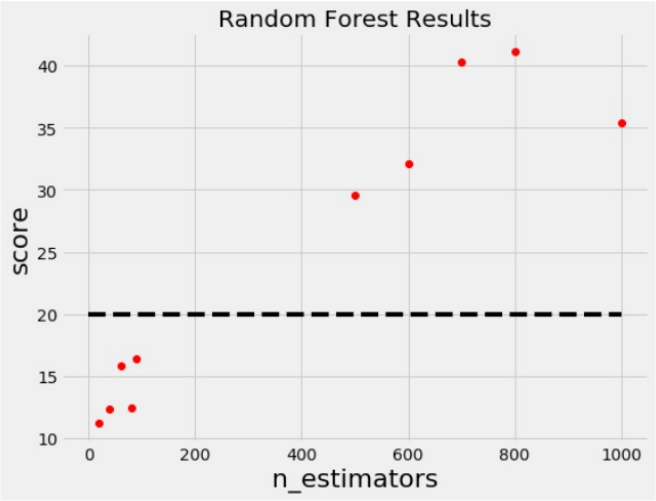


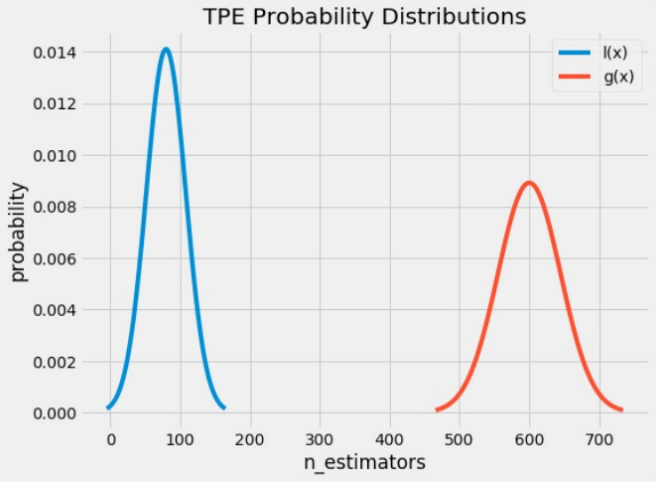
TPE models $P\ (x \mid y)$ by using a threshold y* to make two different probability distributions $l(x)$ and $g(x)$ for the hyperparameters x. $l(x)$ is built upon the group of hyperparameters that yield objective scores lower than the threshold, and $g(x)$ is built upon the group of hyperparameters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

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



TPE in Brief



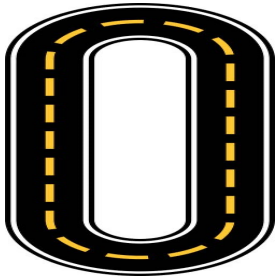


TPE updates $l(x)$ and $g(x)$ on every trial, for every parameter

It 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 evaluates 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 on the objective function



STEP 5: CHANGE HYPERPARAMETER TUNING METHOD

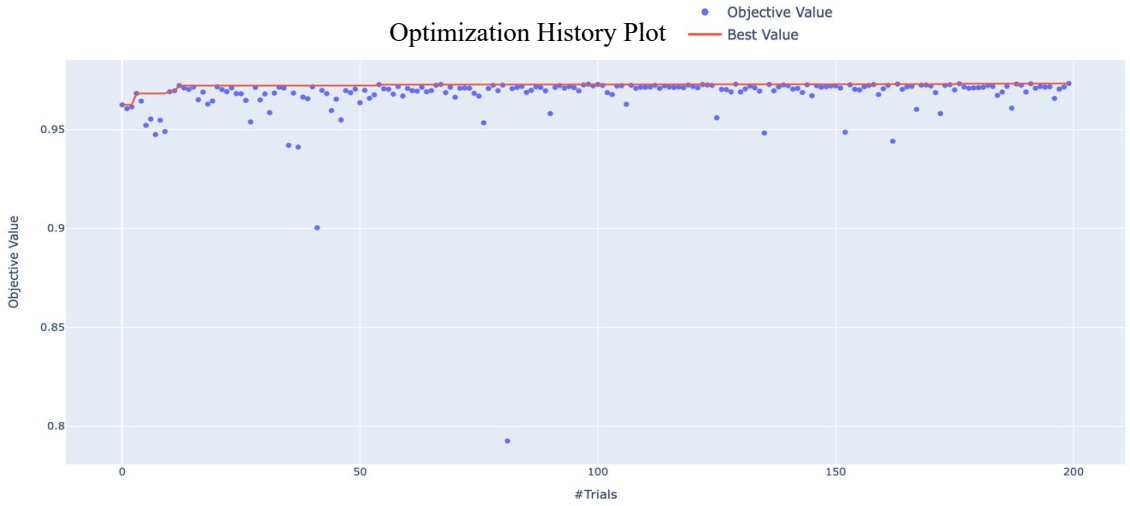
Step 5 Test Run Result

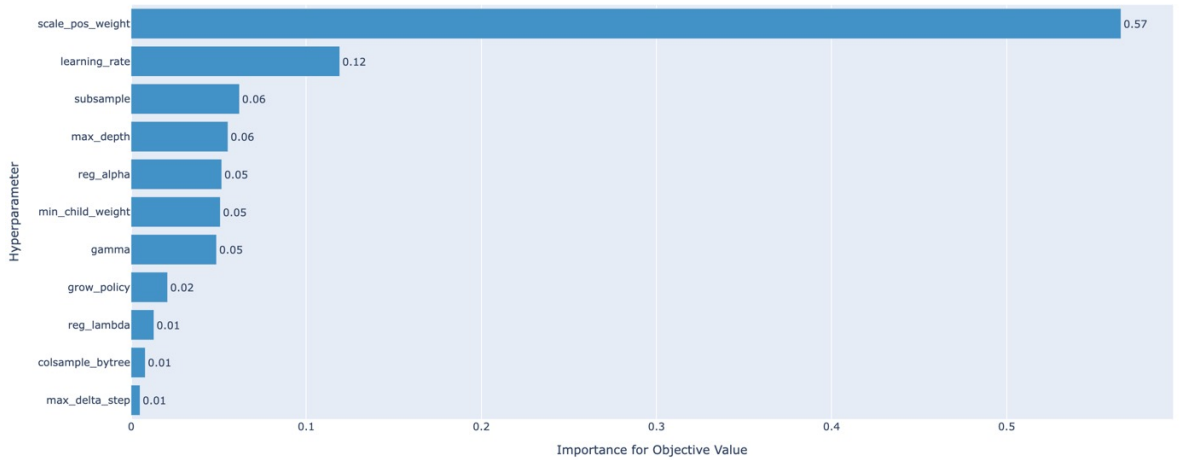
	Step 4 Algorithm Using Grid-Search	**Step 5 Algorithm Using Optuna TPE**	**Step 5 Algorithm Using Optuna TPE - Thres**
Hyperparameter	Grid of 2 dimensions: 12 combinations	Space of 11 dimensions: 200 trials	Space of 11 dimensions: 200 trials
TNR = TN/(TN+FP)	99.82%	99.59%	99.82%
TPR = TP/(TP+FN)	89.42%	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

STEP 5: CHANGE HYPERPARAMETER TUNING METHOD

Step 5 Test Run Result

	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



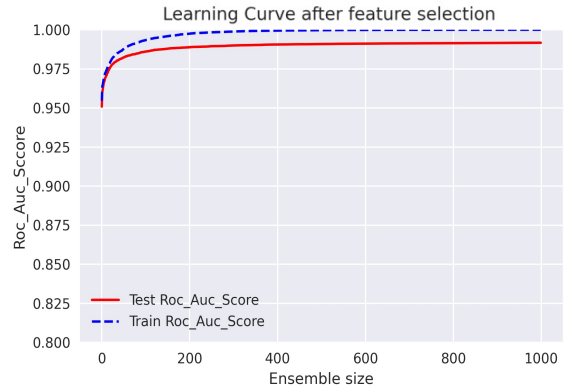


Concluding Summary

	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



Limitation and Future Suggestions

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

For further speed gain:

- Activate the use of GPU for XGBoost by simply adding “tree_method” as “gpu_hist” in the hyperparameter list. This requires CUDA-capable GPUs.
- Use multi-nodes for Optuna TPE for large scale hyperparameter tuning. Requires setting up a SQL database for trial history. Set up SQL database locally in the HPC clusters. Web-based SQL was tried out and not recommended due to latency.
- Implement pruning for Optuna TPE hyperparameter tuning. It is a mechanism to save time from evaluating unpromising trials. The downside is that, to integrate cross-validation with pruning, it may take some time to customize the codes.
- 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.

Limitation and Future Suggestions

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

For potential performance gain:

- 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 different feature selection methods and Bayesian Search in a pipeline. This allows the search for optimal number of features while searching for the optimal hyperparameters.

- Investigate an approach to define and identify outliers in Satellite data, as boosting methods in general are sensitive to outliers.

- Consider experimenting with resampling to further 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

The End

Thank You for listening!