Remote Sensing for Forest Recovery: Technical Report

Benjamin Frizzell Zanan Pech Mavis Wong Hui Tang
Piotr Tompalski Alexi Rodríguez-Arelis

2025-06-18

Table of contents

1 Introduction

Afforestation is essential for capturing carbon, enhancing biodiversity, and improving forest resilience to climate change. It also supports human well-being by providing green spaces for nature-based activities, which can improve mental health and reduce the risks of wildfires and floods in communities. Recognizing its importance, the Canadian government launched the 2 Billion Trees program to provide financial support for organizations to plant trees over ten years across Canadian provinces. However, monitoring the success of large-scale afforestation initiatives remains a critical and complex challenge, particularly during the early stages of growth. Young trees often produce weak spectral signals due to their sparse canopies, making them difficult to detect using traditional remote sensing methods (University of British Columbia Master of Data Science Program 2025). As part of the 2 Billion Trees program—which aims to plant two billion trees across Canada by 2031—Natural Resources Canada must track survival rates across hundreds of ecologically diverse and often remote planting sites (Natural Resources Canada 2021).

In this study, we aim to investigate two main research questions:

- Can satellite-derived vegetation indices and site-level data be used to accurately predict tree survival over time in large-scale afforestation programs?
- Which modelling approach is most effective, and how long after planting is needed before accurate survival predictions can be made?

To address these questions, this study leverages satellite-derived vegetation indices and site-level data to train machine learning models. By evaluating multiple modelling approaches—including logistic regression, random forests, and deep learning models, namely recurrent neural network architectures—we aim to determine which techniques provide the most accurate predictions of tree survival rates to support the mission of sustainable forest management and addressing climate change.

2 Data & Pre-processing

2.1 Data Description

The dataset used in this study is a combination of field-measured survival rates for more than 2,500 afforested sites and remote sensing data obtained from the Harmonized Landsat Sentinel-2 (HLS) project (**hls?**).

The field-measured data and satellite data are of different resolution. While the survival rates data was recorded at site-level, the satellite data was recorded at a higher resolution, where each afforested site is divided into one or more 30m x 30m pixels. Each row in the dataset represents a pixel-level satellite observation at a given time, linked with its corresponding site-level features.

Table 2 shows the pixel-level features and Table 1 shows the site-level features in the original dataset.

Table 1: Summary of site-level features. The site-level features provide spatial, temporal and ecological information associated with each afforested site, including the site ID, area, previous land use, afforestation information, species type, and our target: field-measured survival rates from Year 1 to Year 7.

Category	Column Name	Description
Identifier	`ID`	Site ID
Spatial	`Area_ha`	Area of the Site (hectares)
	`prevUse`	Previous Land Use of the Site
Temporal	`PlantDt`	Planting Date
	`Season`	Planting Year
	`AsssD_1` to `AssD_7`	Date of Field Survival Assessment (Years 1 to 7)
Ecological	`SpcsCmp`	Species Composition of Site
	`Type`	Species Type (Conifer, Deciduous, Mixed)
	`Planted`	Number of Trees Planted (Initial Field Record)
	`NmbrPlO`	Number of Trees Originally Planted
	`NmbrPlR`	Number of Trees Replanted
	`NmbrPlT`	Total Number of Trees Planted (`NmbrPlO` + `NmbrPlR`)
Target	`SrvvR_1` to `SrvvR_7`	Field Measured Survival Rate (Years 1 to 7)

Table 2: Summary of pixel-level features. The pixel-level features include the pixel ID, the capture date of the satellite data, and our primary predictor: the spectral indices.

Category	Column Name	Description
Identifier	`PixelID`	Pixel ID
Temporal	`ImgDate`	Image Date of the Remote Sensing Data
	`Year`	Image Year of the Remote Sensing Data
	`DOY`	Image Day of Year of the Remote Sensing Data
Spectral Indicies	`NDVI`, `SAVI`, `MSAVI`, `EVI`, `EVI2`, `NDWI`, `NBR`, `TCB`, `TCG`, `TCW`	See Table 3 for details.

Table 3: Description of spectral indices available in the dataset.(**zeng2022optical?**; **tasselled?**; **NDVI?**; **EVI2?**; **NBR?**)

Type	Index	Description
Ve getation Index	Normalized Difference Vegetation Index (NDVI)	Measures vegetation greenness and health by comparing near-infrared (NIR) and red reflectance.
	Soil-Adjusted Vegetation Index (SAVI)	Adjusted NDVI that reduces background soil influence and corrects for soil brightness.
	Modified Soil-Adjusted Vegetation Index (MSAVI)	Improved SAVI that minimises soil background influence.
	Enhanced Vegetation Index (EVI)	Measures vegetation greenness using blue, red, and NIR bands to correct for atmospheric and canopy background influences.
	Two-band Enhanced Vegetation Index (EVI2)	Similar to EVI, but only uses red and NIR bands.
	Tasseled Cap Greenness (TCG)	Measures vegetation greenness using a tasselled cap transformation of spectral bands.
Water Index	Normalized Difference Water Index (NDWI)	Measures moisture content by comparing NIR and shortwave infrared (SWIR) reflectance.
	Tasseled Cap Wetness (TCW)	Measures soil and vegetation moisture using a tasselled cap transformation of spectral bands.
Fire Index	Normalized Burn Ratio (NBR)	Identify burned areas and measure burn severity using NIR and SWIR bands.
Surface Ref lectance	Tasseled Cap Brightness (TCB)	Measures soil brightness using a tasselled cap transformation of spectral bands.

2.2 Data Cleaning

After careful inspection of the raw dataset, we perform extensive data cleaning to address data quality issues. The preprocessing steps were outlined below:

1. **Data Loading**

The original dataset was in RDS format, a native data format for R. As we are using Python as our main programming language, we converted the data into a Apache Parquet file format for compatibility.

2. **Records Removal**

   - **Replanted Sites**

     According to the survey records, some of the afforested sites have been replanted. To avoid introducing complex survival dynamics, we removed all records from the replanted sites.

   - **Out-of-Range Values**

     - **Spectral Indices** : With the exception of the Tasseled Cap indices (`TCB`, `TCG` and `TCW`), all spectral indices (e.g. `NDVI`, `NBR`, `EVI` …) should lie within the range [-1, 1]. All records that are out-of-range were removed from the dataset.
     - **Survival Rates** : The field-measured survival rates should be within 0 to 100%. Records that falls outside this range were considered invalid and removed.

   - **Missing Spectral Data**

     The missingness plot (Figure 1) suggest that some spectral data were missing from the dataset. These rows were removed from the data.

   - **Pre-Plantation Satellite Data**

     While the satellite records dates back to 2013, many sites were planted in 2014 or later. To avoid introducing noise, these satellite records captured before planting were removed, as pre-plantation site conditions are not relevant when modeling afforestation survival rates.

3. **Data Engineering**

   Since vegetation indices were mainly used for monitoring vegetation growth and vegetation health, we envision the afforestation density may be a useful feature to add to our data. By normalizing tree counts (`Planted`) across site sizes (`Area_ha`), the new feature `Density` (number of trees per hectare) can provide a more informative representation of underlying site conditions.

4. **Imputing Species Type**

   As observed in Figure 1, many records were missing from the species type (`Type`) column. These missing values can be imputed based on the species composition (`SpcsCmp`) column. According to the Forestry Glossary from Natural Resources Canada, a forest is classified as a mixed stand forest if less than 80% of trees are of a single species. Using this threshold, sites were labeled as `Conifer` if the proportion of softwood species exceeds 80%, `Deciduous` if hardwood species exceeds 80% and `Mixed` otherwise.

5. **Column Removal**

- `PlantDt` : This column was dropped since the majority of values in the column were missing (Figure 1).
- `NmblR`, `NmblT`, `NmblO`: These columns capture the site replanting information. Since all replanted site records were excluded earlier, they are no longer useful, and were removed from the dataset.
- `prevUse`: Exploratory data analysis showed that over 80% of the sites are previously agricultural lands. Due to such severe class imbalance, this column has limited predictive power and was removed from the data.
- `SpcsCmp` : The survey data was collected from two data sources, resulting in inconsistencies in the data format of this column. The majority of the data does not have any detailed species composition, recording only the proportion of hardwood vs softwood trees. As such, this column was only used for the imputation of the species type (`Type`) and dropped afterwards to avoid redundancy.
- `Year` : Both `Year` and `DOY` can be derived from `ImgDate`. To avoid redundancy, `Year` was dropped, retaining only `DOY` for seasonality tracking in RNN modeling.
- `Area_ha`, `Planted` : These two columns were dropped after deriving the new feature `Density`.

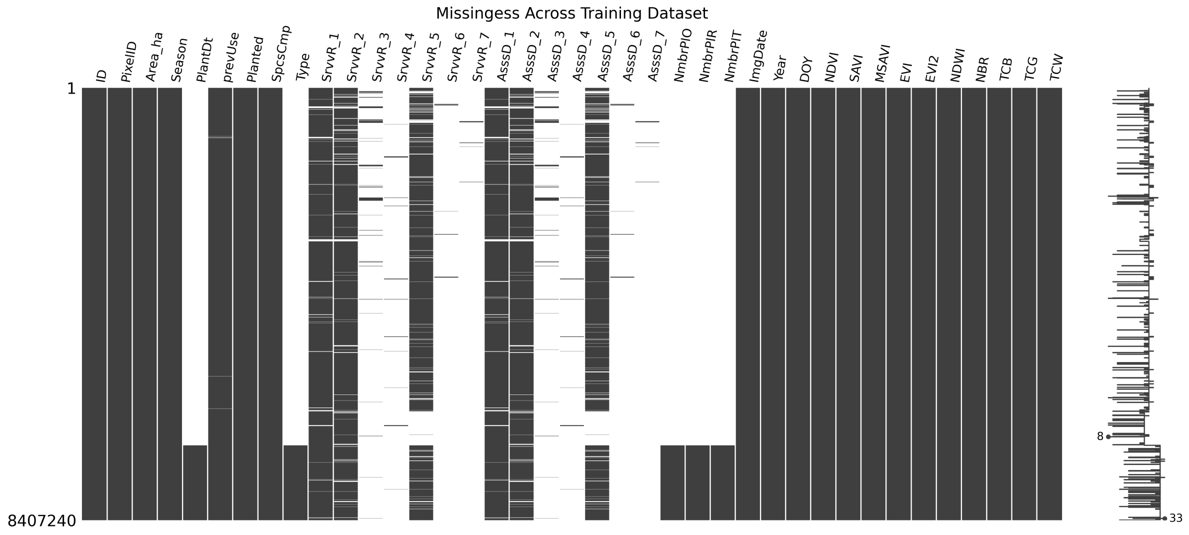

Figure 1

2.3 Train Test Split

We performed a 70:30 split on the processed data to create a training dataset and a test dataset. Instead of using a traditional random train test split, we were splitting the data by site, ensuring that each site appears only in either the training set or the test set. Since the survey data are measured at site-level, all pixels from a given site will share the same survival rate records. This strategy avoids data leakage, ensures that the test data remains unseen during training. It also preserves the temporal structure of the satellite data, which is crucial for RNN models, allowing them to train on the complete time series for each site in the training data.

2.4 Phase 1: Data Preparation for Classical Models

After data cleaning, we performed data transformation to prepare the cleaned data for classical model training. The data transformation steps were outlined below :

1. **Pivoting survival records**

   Since survey records have varying survey frequency, we pivot the data to combine the survival rates columns (`SrvvR_1` to `SrvvR_7`) into a single column (`target`), and the survey dates columns (`AsssD_1` to `AsssD_7`) into a survey date column (`SrvvR_Date`). We added an `Age` column to keep track of the tree's age at the time of the survey (number of years since plantation).

2. **Satellite-Survey record matching**

   Both the survival rates data and satellite data are recorded at irregular time intervals. While the survival rate surveys were conducted annually, most sites only have 3 years of survival rate records (Year 1, 2, 5). On the other hand, satellite data are obtained much more frequently. Since the Harmonized Landsat Sentinel-2 satellite circles the Earth every 16 days, the satellite records have an average time interval of around 16 days. With each pixel having hundreds of satellite records and only 3 survival rate records, we needed a way to match the survival records with the satellite records.

   Due to seasonality in the satellite records, we cannot simply take an annual average of the vegetation indices. Instead, we computed the average signal within a ± 16 days time window of the survey date. We chose a ± 16 day window specifically to match the repeat cycle of the Landsat satellite, ensuring at least 1 satellite record returned for most survey records.

3. **Binary Target Mapping**

   We approached the problem as a classification problem. To do this, we map the `target` (survival rates) into binary classes `Low(0)`/ `High(1)` survival rates. The target is classified as having a low survival rate if it falls below the threshold; and high survival rate otherwise. Since we do not have a defined classification threshold for high and low survival rates, we tried multiple thresholds (50%, 60%, 70% and 80%) and obtained results for each threshold for comparison.

4. **One-hot-encoding of `Type`**

   While Random Forest and Gradient Boosting models have native support for handling categorical features, logistic regression models can only handle numeric features. To maintain consistency, OneHotEncoding was applied to the `Type` column for all classical models.

5. **Standard Scaling**

   Since the logistic regression model is also sensitive to the scale of the data, StandardScaler was also applied to the numeric features before fitting the logistic regression model.

2.5 Phase 2: Data Preparation for RNN models

During the second phase of our project, we worked on RNN models which are designed to capture sequential changes. RNN models require a different data format, processing the satellite records as a time series of spectral indices instead of individual observations. Here is an outline of the preprocessing techniques used to prepare our data for RNN modeling.

1. **Validation Split**

   When training the RNN model, in addition to a test set, we need a validation set to evaluate model performance during model training. As such, we did a 50:50 split on the test data to obtain a validation set.

2. **Data Engineering**

   To better capture the time dependencies in the satellite data, we procure two new features to the satellite data.

   - `Log transformed time_delta`: The `time_delta` records the difference between the image date and the survey date. We use this to capture the irregularities in the time steps of the satellite records. This column also helps the model prioritise the recent data over the old data. We conduct a log transformation on the `time_delta` to normalise it since its values can go up to thousands.
   - `negative cosine transformation DOY`: We use a cosine transformation of `DOY` to capture the seasonality of the spectral indices. We chose a negative cosine transformation specifically as it mimics the fluctuation patterns of all the spectral indices (except for `TCB`), which peaks during summer and drops in winter.

3. **Data Normalisation**

   Since RNN models are sensitive to the scale of the data, we need to normalise the data to avoid vanishing or exploding gradient. As most of the spectral indices are bounded between [-1, 1], only the `TCB`, `TCW`, `TCG` and `Density` columns were normalised. To avoid data leakage, the summary statistics (mean and standard deviation) was computed using only the training data. This statistics is then used to normalising the train data, test data and validation data.

4. **One-hot-encoding of `Type`**

   Since RNN models can only handle numeric data, we use one-hot encoding to transform the species type column into the `Type_Deciduous` and `Type_Conifer` columns. Since the species types are mutually exclusive, the `Type_Mixed` column was dropped to remove linear dependencies between type columns and reduce redundancy.

5. **Sequence Generation**

   We split the survey records and satellite records into separate data frames: The look-up table containing the site-level survey records and the image records table containing the pixel-level satellite data. Similar to what we did for the classical models, we pivot the survey records so all survey records and survey dates are combined into respective columns.

   For each row in the lookup table, we searched the image table for all records with match `ID`, and `PixelID` and selected all satellite records up until the survey date. This would be the sequence data we use for training our RNN model. We saved the sequence for each survival record as an individual parquet file. The file name was saved in the look-up table to allow easy access during model training. The rows with no sequences available (e.g. survival records before 2013, when the first satellite record was obtained) were removed.

6. **RNN Dataset and Dataloader**

   Depending on the age of the site, the sequence length for each survival record varies. For example, for a year 7 survival record, the sequence can contain up to 7 years of satellite records (which were recorded every 16 days in theory).

   To feed the sequence data into the RNN model, the sequence within the same batch needs to have the same sequence length. In pytorch, by default, the dataset is shuffled randomly before each epoch to improve generalization. However, with such a large variation in sequence length, random shuffling will result in excessive padding for short sequences.

   To reduce the amount of padding needed to optimise memory usage while still introducing randomness to the data, we created a custom Pytorch dataset for passing the sequence data to the RNN model. This custom dataset class has an associated method that shuffles the dataset within their Age group. The idea is that samples of the same age are more likely to have a similar sequence length. By shuffling within their age group, we were able to introduce randomness to the training data, while minimizing the padding lengths.

7. **Target mapping**

   Since training the RNN model is time-consuming, and we do not have a defined classification threshold, we decided to train a regression RNN model instead. This way, we do not need to train a separate RNN model for each threshold value. Therefore, we did not map the target values to binary classes in the data preprocessing state. The mapping was done before model evaluation after the model had been trained.

3 Modelling Specifications

This phase of modeling began with three classical machine learning models: Logistic Regression (as a baseline), Random Forest, and Gradient Boosting Machines. To better capture the temporal structure of the remote sensing time series, two recurrent neural networks—GRU and LSTM—were subsequently developed. This section provides detailed descriptions of each model's architecture and the rationale for their selection. Methods for training, tuning, and evaluating model performance will also be thoroughly outlined.

3.1 Classial Modelling

3.1.1 Logistic Regression

Logistic regression is a generalized linear model widely used for binary classification tasks, valued for its simplicity and interpretability. It models the **log-odds** of the probability, or the **logit** that a given record belongs to class 1 as a linear combination of the input features (Wikipedia contributors 2025):

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta^\top \mathbf{x}_i, \quad i = 1, \dots, n \tag{1}$$

Here,

- n denotes the sample size,
- $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iD}]$ is the D-dimensional feature vector for the ith observation (e.g., site-level features and aggregated vegetation indices),
- p is the probability that the target label y_i corresponds to the high survival class: $p = P(y_i = 1 \mid \mathbf{x}_i)$

The coefficient vector $\beta = [\beta_1, \beta_2, \dots, \beta_D]$ represents the influence of the features on each prediction. The jth entry of β corresponds to the change in the log-odds associated with a one-unit increase in the jth feature, holding all other features constant.

An optimal estimate of β is determined by minimizing the **cross-entropy loss**:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)\right], \tag{2}$$

Where $\hat{p}_i$ is the estimated class probability obtained from the inverse of Equation 1, which can be shown to be the **sigmoid function**:

$$\hat{p}_i = \sigma(\beta_0 + \beta^\top \mathbf{x}_i) = \frac{1}{1 + \exp\left(-(\beta_0 + \beta^\top \mathbf{x}_i)\right)} \tag{3}$$

These probabilistic predictions can be converted to binary class labels by applying a specified decision threshold, typically 0.5. Model performance across different thresholds can be evaluated using ROC and precision–recall (PR) curves, which are discussed in Section 3.6.

Overall, logistic regression provides an interpretable, statistically grounded baseline and serves as a proxy for the classical statistical modeling used prior to this analysis. To demonstrate the value of more sophisticated machine learning models in predicting survival rates, any subsequent models should achieve performance that exceeds that of logistic regression.

3.1.2 Tree-Based Modelling

Many high-performing machine learning models are composed of simple, rule-based structures known as decision trees. These models make predictions by recursively partitioning the input dataset into distinct regions based on selected features and threshold values. An example of a decision tree is shown in Figure 2.

Each internal node in the tree represents a decision based on a specific feature and a corresponding threshold, and each leaf node corresponds to a unique subset of the data, defined by the path of decision rules leading to it. In binary classification, the majority label of samples in a leaf node is used as the prediction, but for regression, the mean of the target within the leaf node is given. Feature-threshold pairs are selected using a greedy algorithm: starting from the root node, the tree is grown iteratively by choosing the split that most effectively reduces the value of a given loss function. The cross-entropy loss defined in Equation 2 is commonly used for binary classification tasks; however, Gini impurity is another frequently used criterion (Pedregosa et al. 2011). Alternatively, regression loss functions such as MSE can be used for Regression Tree tasks. Tree construction halts either when a leaf node contains only one class (resulting in zero loss for that subset) or when a predefined stopping criterion, such as the maximum depth, is met. See Section 3.3.2 for guidance on selecting an appropriate maximum

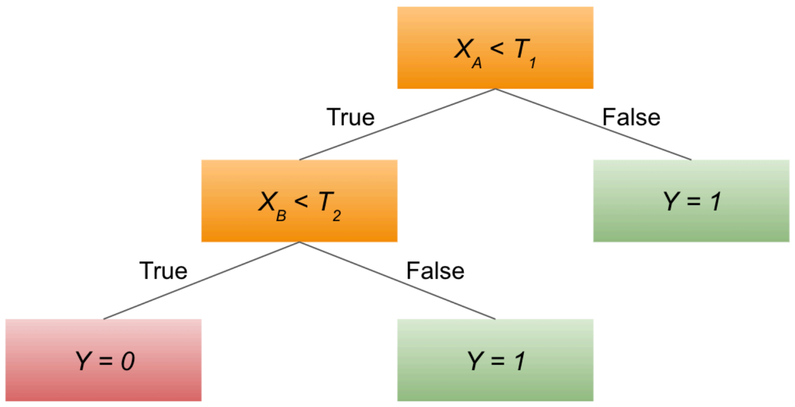

Figure 2: A simple example of a Decision Tree with a depth of 2. The predictions Y are made by comparing selected features X_A and X_B via comparison with threshold values T_1 and T_2.

tree depth; choosing a higher max depth generally leads to a greater number of decisons and an overall more complex model.

3.1.3 Random Forest Classifier

The Random Forest model is an aggregate model composed of many decision trees, each trained on a bootstrapped subset of the training data and a randomly selected subset of the features. Typically, the maximum allowable depth for each tree in a Random Forest is quite high, resulting in individual trees that are often overfitted and exhibit high variance. However, this high variance is mitigated through aggregation: by combining the predictions of many diverse trees, the overall model can generalize effectively to unseen data. For binary classification tasks, the final prediction is determined by majority vote among the individual trees.

Although training Random Forests can be computationally intensive, each tree is trained independently, enabling efficient parallelization and scalability. Previous studies from Bergmüller and Vanderwel (2022), have demonstrated that Random Forests perform well when using vegetation indices to predict canopy tree mortality. Becuase of this, this model was selected as a candidate for the present analysis.

3.1.4 Gradient Boosting Classifier

The Gradient Boosting model is a popular model that exists in a collection of ‘boosting’ models, which -unlike Random Forests- consists of a sequence of underfit and biased ‘weak learner’ models which converge to a high-performing ‘strong learner’ model when combined (Zhou

2025). This model was selected as a candidate model due to fast implementation and strong performance across a wide variety of machine learning tasks (Chen and Guestrin 2016).

Convergence to a strong learner from a series of weak learners is performed by iteratively fitting a regression tree to the errors of the previous model estimate. To understand this, we firt define the per-sample the loss to be the negative of Equation 2 evaluated for a particular class prediction $\hat{p}_i$:

$$\ell_i(\hat{p}_i, y_i) = -\left[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)\right] \tag{4}$$

The model outputs raw logit predictions $f_i(\mathbf{x}_i)$, which can be converted to probabilistic predictions via the sigmoid function shown in Equation 3:

$$\hat{p}_i = \sigma(f_i(\mathbf{x}_i))$$

The errors associated to each prediction are quantified by the **gradient** g_i and **Hessian** h_i of the loss with respect to the model estimate:

$$g_i = \frac{\partial \ell_i}{\partial f(\mathbf{x}_i)} = \hat{p}_i - y_i \tag{5}$$

$$h_i = \frac{\partial^2 \ell_i}{\partial f(\mathbf{x}_i)^2} = \hat{p}_i(1 - \hat{p}_i) \tag{6}$$

3.1.4.1 Initialization

The model initializes with a constant prediction f_0 across all training sample, usually taken as the logit function (ie. the left-hand side of Equation 1) evaluated over the proportion of samples with label 1:

$$f_0 = \log\left(\frac{P(Y=1)}{1 - P(Y=1)}\right)$$

3.1.4.2 Update step

To update the model prediction after initialization, a regression tree is fitted with the gradients given by Equation 5 as the target predictor. Using Newton's method, the output for a particular leaf node j is given by the sum of g_i and h_i for all samples that reach that leaf node.

$$\omega_j^{(1)} = \frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i} \tag{7}$$

The overall model prediction is then updated:

$$f_1(\mathbf{x}_i) = f_0 + \eta \omega_{\mathbf{x}_i}^{(1)}$$

Where $\omega_{\mathbf{x}_i}$ denotes the leaf node that sample $\mathbf{x}_i$ is assigned.

Where η is a predefined **learning rate** which controls the degree to which each weak learner can make contributions to the overall model estimate. See Section 3.3.2 for further details.

This update process is repeated iteratively, producing a final estimate of the log-odds which can be converted to a class probability and class labels through the same process as that of the logistic regression model:

$$F(\mathbf{x}_i) = f_0 + \eta \sum_{k=1}^{K} \omega_{\mathbf{x}_i}^{(k)}$$

Where K is the total number of iterations of the algorithm.

3.2 Feature Selection Methods

To address collinearity among vegetation indices and evaluate the importance of both site-based and remote sensing features, we applied three feature selection methods:

3.2.1 Permutation Importance

We estimate each feature's importance by randomly shuffling its values across samples before training, then measuring the resulting change in the model's performance (F_1 score; see Section 3.6). This yields an interpretable, global importance metric. However, when predictors are highly correlated, it can misattribute importance—because different features may serve as proxies for one another—leading to misleading rankings (Pedregosa et al. 2011).

3.2.2 SHAP Values

SHAP (SHapley Additive exPlanations) return per-prediction feature contributions based on Shapley values from cooperative game theory (Lundberg and Lee 2017). Concretely:

1. A baseline expectation is defined (e.g., the average model output when no features are known).

2. For each feature, all possible subsets of features including and excluding that feature are considered, and the marginal contribution is averaged.

3. Taking the mean absolute SHAP values across all samples yields a global importance ranking.

This method provides both local (per-prediction) and global interpretability. However, SHAP may tacitly distribute credit among highly correlated features—sometimes giving a near-zero or inflated value to one feature over another—depending on whether the model uses marginal or conditional expectations when computing the baseline.

3.2.3 Recursive Feature Elimination with Cross-Validation (RFECV)

Finally, RFECV is used to iteratively train the model and remove the least important features based on model-derived importance metrics (e.g., coefficients or feature gains). Each reduced feature subset was evaluated by its F_1 performance using cross-validation (see Section 3.3.3). This method directly handles correlated features by eliminating them if they do not contribute to the model performance, however it can be quite computationally exhaustive. Feature rankings based on how early features were removed are used as importance metrics.

3.3 Training and Tuning Classical Models

Most machine learning models involve a set of hyperparameters—values specified *a priori*—that govern model complexity and influence training behavior. Inappropriate hyperparameter choices can result in models that are either overly biased or unnecessarily complex, leading to poor generalization on unseen data. This section provides a detailed overview of the key hyperparameters for each candidate model in this analysis, along with the methodology used for their selection.

3.3.1 Regularization

In general, regularization involves a penalty to the loss function of that is proportional to the magnitude of the model parameters; stronger regularization leads to smaller parameters and more conservative predictions, which often aids in decreasing overfitting and variance. In Logistic Regression, this is implemented through an additional term in Equation 2:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right] + \lambda R(\beta) \tag{8}$$

Where λ controls the strength of regularization (larger values lead to stronger regularization), and $R(\beta)$ is some function of the model parameter magnitude. In L_1 regularization, $R(\beta) = \sum_j |\beta_j|$, and for L_2 regularization, $R(\beta) = \sum_j (\beta_j)^2$. L_1 tends to decrease parameter values to 0 in a linear fashion, whereas L_2 causes parameters to asymptotically decrease towards, but never exactly to 0.

In the context of Gradient Boosting with XGBoost, regularization is applied to the loss function in the form:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right] + \lambda\left(T + R(\beta)\right) \tag{9}$$

Where T is the number of leaves in the tree. Regularization is also applied to the weights directly, via modification of Equation 7 as implemented by Chen and Guestrin (2016):

$$\omega_j = \frac{\sum_{i\in j} g_i}{\sum_{i\in j} h_i + \lambda} \tag{10}$$

Generally, model performance varies logarithmically with λ, therefore it is advised that test values be sampled on a logarithmic scale when optimizing for performance.

3.3.2 Tree Hyperparameters

Nonparametric models such as Random Forest do not incorporate explicit regularization terms. Instead, they are controlled through structural hyperparameters that constrain model complexity. As discussed in Section 3.1.2, **maximum depth** is a key hyperparameter that limits the number of hierarchical decision rules in each tree, thereby directly affecting overfitting. Additional parameters—such as the **minimum number of samples per leaf**, the **cost-complexity pruning parameter** (α), and the **number of estimators** (trees)—can also be tuned to control generalization error (Pedregosa et al. 2011). However, to reduce computational cost and simplify the tuning process, only maximum depth and the number of estimators were optimized in this analysis.

3.3.3 Random Search Cross-Validation

Given a candidate model and a set of tunable hyperparameters, an optimization problem naturally arises: which hyperparameter configuration yields the best model performance? To address this, the present analysis employed random search cross-validation to tune hyperparameters. The process is illustrated in Figure 3.

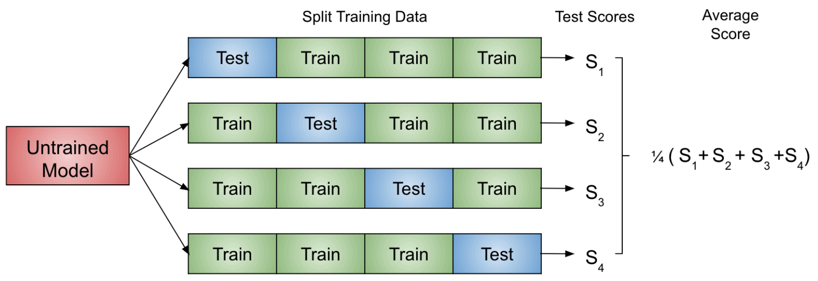

Figure 3: An example of four-fold cross-validation. A given model with a configuration of hyperparameters is trained four times, each time leaving out one subset of the data as a hold-out validation set. The model is evaluated on the hold-out fold, and the resulting scores are averaged. This process is repeated for multiple hyperparameter configurations. The configuration with the best average score is selected for further evaluation. Many scoring metrics exist depending on the use case and data characteristics; see Section 3.6 for details on the metrics used in this analysis.

Cross-validation mitigates the risk of overfitting by simulating model performance on unseen data through repeated training on subsets of the data while reserving a separate fold for validation. Averaging the resulting scores provides a more realistic estimate of generalization performance than fitting on the entire training set alone.

In random search, hyperparameter values are sampled from predefined distributions at each iteration, offering an efficient alternative to grid search, which exhaustively evaluates all combinations of specified values. Although grid search can be effective for low-dimensional hyperparameter spaces, it quickly becomes computationally prohibitive as the number of parameters increases. Accordingly, random search was chosen for its efficiency and scalability in this analysis.

3.4 Sequential Deep Learning Models

While the previously discussed models perform well across a range of supervised learning tasks and provide a strong performance baseline, they are limited by their assumption that each input instance is independent. This assumption is ill-suited to the sequential structure of the vegetation index data in this study, which exhibits temporal dynamics and potential spatial correlations between pixels within sites. To better model these dependencies, the final phase of the analysis employed sequential deep learning architectures based on Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models. Despite their increased complexity and computational demands, these models are efficiently implemented using modern deep learning libraries in Python, such as PyTorch (Paszke et al. 2019).

3.4.1 Recursive Neural Network (RNN)

The simplest deep learning model that supports sequential modelling is the RNN. Figure 4 outlines the architecture of this model.

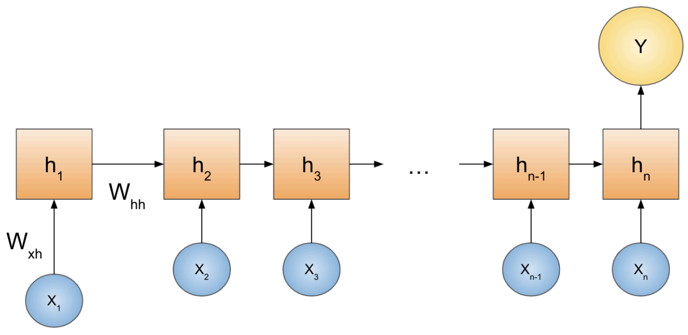

Figure 4: Basic architecture of a many-to-one RNN. Inputs from a sequence of vectors $(X_1, X_2, ..., X_n)$ are taken one-by-one, which updates a hidden state vector h_i according to a linear transformation with a weight matrix W_{xh}. As furher inputs are processed, the hidden state recursively updates according to the input as well as the previous hidden state through the weight matrix W_{hh}. A many-to-one RNN outputs one prediction Y after the final entry of the sequence is processed.

The key component of the RNN is the **hidden state**, which encodes the 'memory' of previous instances in the sequence. The transformation of the hidden state is governed by weight matrices W_{hh} and W_{xh}. Additionally, bias vectors b_{xh} and b_{hh} are included, and the linear transformation is passed through the hyperbolic tangent (tanh) function to introduce nonlinearity. Therefore, the hidden state h_t at time t in the sequence is updated given the previous hidden state h_{t-1} and current sequence entry x_t according to the transformation:

$$h_t = \tanh\left(x_t W_{xh}^T + b_{xh} + h_{t-1} W_{hh}^T + b_{hh}\right) \tag{11}$$

Although the RNN is capable of capturing short term dependencies in sequential data, long-term trends are difficult to capture due to issues of 'vanishing' and 'exploding' gradients during training (Pascanu, Mikolov, and Bengio 2013). See Section 3.5 for further details regarding this.

3.4.2 Long-Term Short Term Memory (LSTM)

To address the long-term dependency issue regarding RNNs, several models of similar, but more complex architecture have been proposed. One such model is the LSTM, which includes additional weights in the form of **input**, **output**, **cell**, and **forget** gates (i_t, o_t, g_t, f_t) respectively. These gates determine which aspects of the prior hidden state and current input are 'important' for prediction. These gates are used to update the cell state c_t, which is then used to update the current hidden state according to the equation:

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{12}$$

Where $\odot$ represents the Hadamard product (elementwise multiplication of vector entries) and σ represents the sigmoid function introduced in Equation 3. The cyclical behaviour of the hidden state update helps to control for problematic gradients during training, making the LSTM suitable for many long-term sequential modelling and prediction tasks (Sak, Senior, and Beaufays 2014). However, the introducion of several new weight matrices and bias vectors can lead to excessively complex models that take extensive time to train.

3.4.3 Gated Recurrent Unit (GRU)

Like LSTMs, GRUs were developed to handle the vanishing gradient problem of RNNs. However, GRUs only utilize a **reset**, **update**, and **candidate** hidden state gate: ($r_t, z_t, \hat{h}_t$). This allows for a lighter model than the LSTM, often with comparable performance on certain tasks (Ravanelli et al. 2018). Hidden states are updated according to the equation:

$$\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\
\hat{h}_t &= \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \\
h_t &= (1 - z_t) \odot \hat{h}_t + z_t \odot h_{t-1}
\end{aligned} \tag{13}$$

3.4.4 Bidirectional RNNs

In addition to the usage of GRUs and LSTMs, additional hidden layers that update in reverse sequential direction may also be used to capture more complex, past and future dependencies. The hidden states can then be concatenated and used for prediction, as shown in Figure 5.

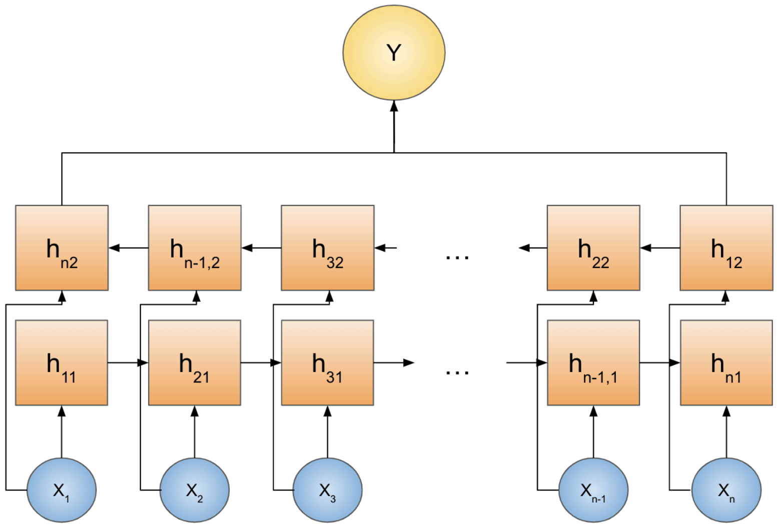

Figure 5: A schematic of a bidirectional RNN, where inputs are passed to hidden states in the forward and reverse temporal direction. The produced hidden states can be combined via concatenation, addition, or averaging to produce a final vector to be used for predicion.

In this analysis, including bidirectional layers appeared to slightly improve model performance. See Section 4 for further details.

3.4.5 Fully Connected Neural Network (FCNN)

The RNN model produces a hidden state which acts as a vectorization the processed vegetation index sequence. To convert this to a single scalar prediction, a **Fully Connected Neural Network (FCNN)** layer can be used. The architecture of this model is shown in Figure 6.

Inputs are passed between layers through a linear transformation using weight matrices and bias vectors, wrapped by an activation function to scale outputs and introduce nonlinearity into the system. For example, the first hidden layer $h^{(1)}$ in Figure 6 is produced by the transformation:

$$h^{(1)} = \text{ReLU}\left(W^{(1)}x + b^{(1)}\right)$$

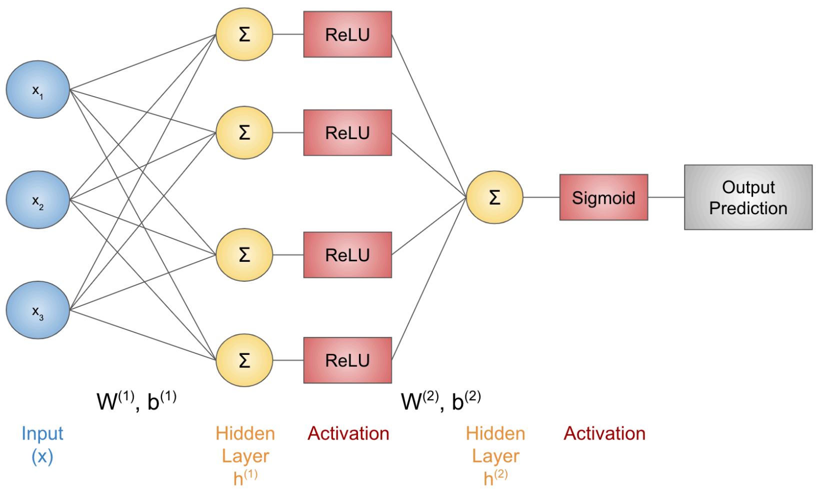

Figure 6: An example of an FCNN of two layers, with a three-dimensional input and one-dimensional output.

Where $W^{(1)}$ is a 4x3 matrix and $b^{(1)}$ is a vector of length 4, both of which have values that are learned during training. A common activation function is the **Rectifier Linear Unit (ReLU)** which simply has the form:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{14}$$

Although Equation 3 is also a common activation function due to its controlled range of (0,1). Generally, more hidden layers and higher dimensional hidden layers allow for higher performing models, at the cost of model complexity and training time.

3.4.6 Proposed Model Architecture

Following preprocessing (Section 2), the deep learning pipeline —comprising components from Section 3.4.2 through Section 3.4.5— proceeds as follows:

1. The sequence of vegetation indices and engineered features is passed through a bidirectional GRU or LSTM, producing a hidden state.
2. The static site features are concatenated onto the hidden state vector and passed to a multilayer FCNN.
3. The final layer of the FCNN output is a scalar, which is passed through a sigmoid activation and multiplied by 100 to produce an estimate of the survival rate of the site pixel.

In addition to these steps, **layer normalization** -which normalizes hidden layer output as a method of stabilizing predictions- was experimented with, although no improvement to predictions was observed. **Dropout** -which randomly sets some parameter values to zero as a method of regularization- was also added within the FCNN layers. Modelling was attemped with and without site features to assess their usage in predicting survival rate. Hidden state and hidden layer sizes, and the number of hidden layers are all variable hyperparameters than may effect model performance (Greff et al. 2017).

3.5 Training Deep Learning Models

Training relied on mini-batch stochastic gradient descent (SGD)—specifically the Adam optimizer, which combines momentum and adaptive learning rates. Gradients of model parameters are computed via backpropagation (chain rule) and updated in the direction opposite to the gradient, scaled by the learning rate. Mini-batch SGD, using shuffled subsets of training data, was employed to improve convergence speed and generalization.

Deep or recurrent architectures (e.g., long sequences or many layers) can suffer **vanishing gradients**, where gradient magnitudes shrink exponentially, or **exploding gradients**, where they grow uncontrollably—both impeding effective training. To mitigate these, we applied regularization techniques from Section 3.4.6 (e.g., dropout, layer normalization) and utilized GRU/LSTM architectures designed to ease gradient propagation.

Training used the **Mean Squared Error (MSE)** loss:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i and $\hat{y}_i$ are true and predicted survival rates. MSE is appropriate for continuous targets -especially when additional penalization of drastic errors is reqiured- however this introduced a methodological discrepancy between this phase of deep modeling and the classification-oriented baselines, which is addressed in Section 3.6. Further discussion on transforming targets into binary form appears in Section 2.

3.6 Error Metrics

Model performance was primarily evaluated using F_1 score, **precision**, and **recall**, with classification accuracy treated as a secondary metric due to class imbalance favoring high-survival sites. To enable comparison between classical classification models and deep learning regression models, continuous survival rate predictions were thresholded to produce binary labels. In the context of these metrics, **low-survival sites are treated as the positive class**, reflecting the goal of identifying potentially failing afforestation sites for targeted intervention.

3.6.1 Precision, Recall, F_β Score

The **precision** of a classifier measures the proportion of true positive (TP) predictions among all instances predicted as positive, including both true positives and false positives (FP):

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

In contrast, **recall** (also known as sensitivity or true positive rate) measures the proportion of true positive predictions among all actual positive instances, including false negatives (FN):

$$\text{Recall} = \frac{\sum \text{TP}}{\sum(\text{TP} + \text{FN})}$$

Recall is particularly important in this imbalanced classification task, as it reflects the model's ability to correctly identify **low-survival (high-risk)** afforestation sites—those most in need of intervention.

To balance both precision and recall in a single metric, the F_β **score** is used, defined as the weighted harmonic mean:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}$$

Larger values of β emphasize recall more heavily. This analysis primarily used the F_1 **score** ($\beta = 1$), which equally weights precision and recall, and also reported the F_2 **score** as a secondary metric to emphasize recall in support of the primary goal of detecting unhealthy sites.

3.6.2 ROC and PR Curves

To evaluate classifier performance across a range of decision thresholds, two widely used diagnostic tools are the **Receiver Operating Characteristic (ROC) curve** and the **Precision–Recall (PR) curve**.

The **ROC curve** plots the **true positive rate** (recall) against the **false positive rate** (FPR) at various classification thresholds:

$$\text{TPR} = \frac{TP}{TP + FN}, \qquad \text{FPR} = \frac{FP}{FP + TN}$$

It summarizes a model's ability to distinguish between the positive and negative classes, regardless of their prevalence. **Note:** this treshold is intrinstic to the probabilistic predictions of the model, and is **not** the same threshold used in the preprocessing target conversion.

The **area under the ROC curve (AUC-ROC)** provides a scalar summary of this performance; a value of 1 indicates perfect separation, while 0.5 indicates no discriminative ability.

ROC curves can be misleading in **imbalanced classification problems**, where the majority class dominates performance metrics. In this analysis, most afforestation sites exhibit **high survival**, so the model may appear to perform well overall even if it fails to identify the few low-survival sites of interest. The **Precision–Recall (PR) curve** more directly reflects performance on the **minority (positive) class**, which in this case is defined as **low-survival** sites. This curve focuses on the trade-off between **false positives** and **false negatives** when trying to identify high-risk sites. The **area under the PR curve (AUC-PR)** is more informative than AUC-ROC in this context because it penalizes false positives more explicitly and does not assume a balanced label distribution.

4 Data Product & Results

Our delivered data product is a comprehensive machine learning pipeline, composed of a series of interconnected Python scripts. This pipeline automates the end-to-end process of data preparation, model development, and evaluation. Key functionalities include robust data processing steps such as cleaning, transformation (pivoting), and splitting into training and testing sets. Once the data is prepared, users can leverage dedicated scripts to train various machine learning models, perform hyperparameter tuning for optimization, and rigorously evaluate model performance. The primary objective of this data product is to provide an effective AI-driven solution for identifying and predicting instances of low survival rates.

4.1 Classical Models Evaluation

In the first phase, we trained several classical machine learning models, including logistic regression as a transparent baseline, as well as ensemble methods such as random forest and gradient boosting. These models were selected based on their proven effectiveness in some of similar time series research. Logistic regression provides interpretability and serves as a benchmark, while the ensemble models are capable of capturing more complex, non-linear relationships in the data. This approach allows us to compare model performance and select the most suitable method for predicting low survival rates.

4.1.1 Permutation Feature Importance

The permutation feature importance plot as show in Figure 7 compares how different features influence model performance for Logistic Regression, Random Forest, and Gradient Boosting across four thresholds (50%, 60%, 70%, and 80%) used to define low versus high survival rates. At the lower thresholds (50% and 60%), Logistic Regression (blue) places greater importance on remote sensing vegetation indices, especially NDVI, EVI1, EVI2, and NDWI, indicating that these spectral features are valuable in distinguishing low from high survival when the cutoff is lower. Random Forest (orange) and Gradient Boosting (red) show lower overall importance at these thresholds but begin to emphasize features like Density, Age, and some spectral indices.

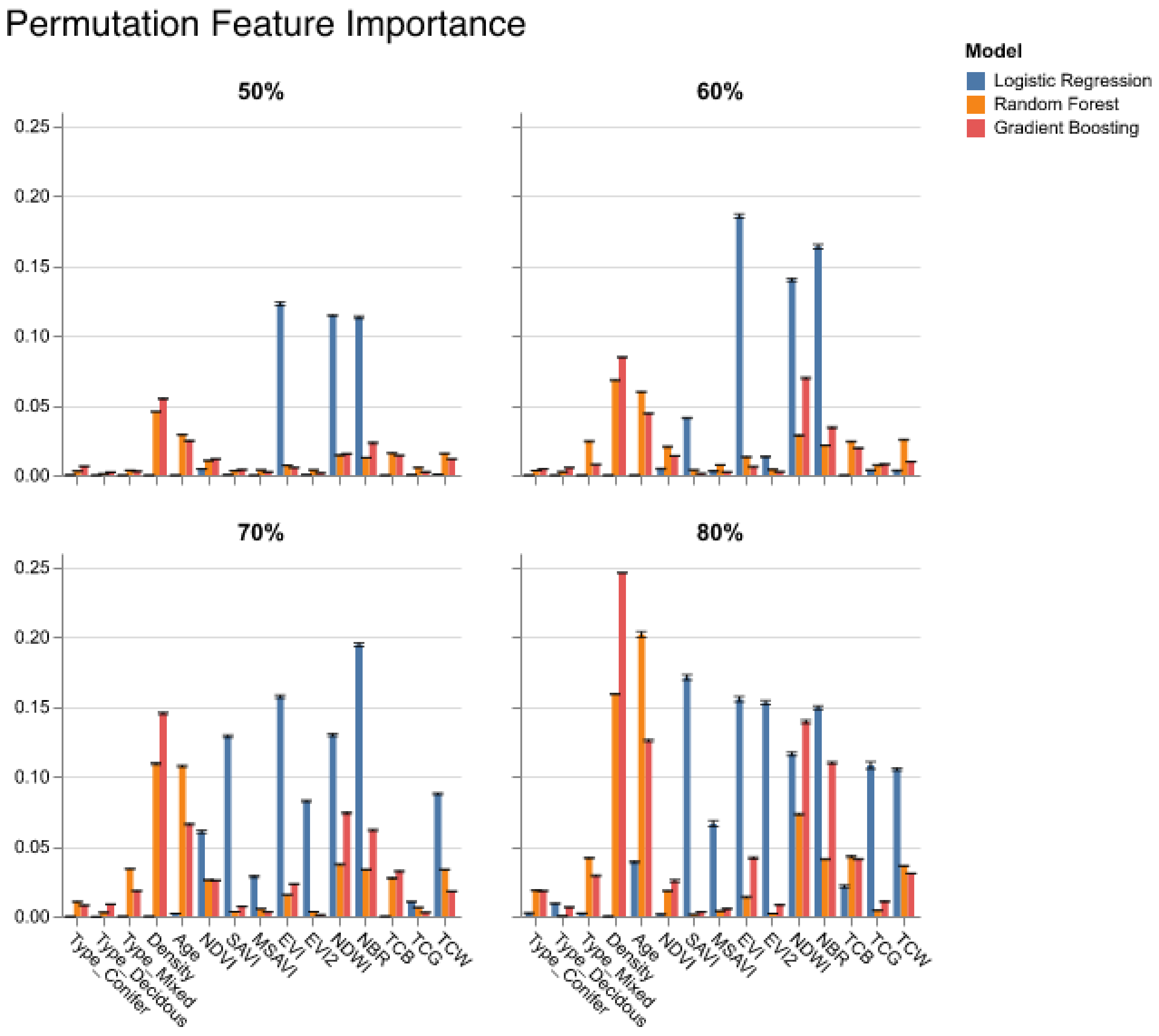

Figure 7: Permutation importance bar plot

As the threshold increases to 70% and 80%, the pattern shifts. Tree-based models, particularly Gradient Boosting, assign higher importance to structural features like Density and Age, with Density becoming the most important feature for Gradient Boosting at the 80% threshold. Meanwhile, Logistic Regression continues to rely heavily on spectral indices, but their relative importance becomes slightly more balanced with structural variables like NBR and NDWI. Random Forest maintains moderate importance across both types of features but shows no dominant patterns.

Overall, Logistic Regression favors spectral vegetation indices, especially when the threshold for low survival is set lower. In contrast, Gradient Boosting and, to a lesser extent, Random Forest prioritize structural stand features like Density and Age more strongly as the survival threshold increases. This suggests that tree-based models may better capture non-linear relationships between structural features and survival, particularly when distinguishing very low from very high survival outcomes.

4.1.2 SHAP Feature Importance

The SHAP feature importance plot as shown in Figure 8 illustrates how different features contribute to predictions. Across all thresholds, Logistic Regression (blue) consistently assigns high SHAP values to vegetation indices such as NDVI, MSAVI, EVI1, EVI2, NDWI, and NBR, indicating that it heavily relies on spectral information from remote sensing data to make predictions. The influence of these indices becomes more pronounced as the survival rate threshold increases, with NDWI and MSAVI emerging as particularly dominant at the 70% and 80% thresholds.

In contrast, Random Forest (orange) contributes minimal feature importance across all thresholds, as indicated by its near-zero SHAP values, suggesting either weak feature attribution under SHAP for this model or that Random Forest is relying more on complex interactions that are not easily captured by additive SHAP values. Gradient Boosting (red), on the other hand, demonstrates moderate and evolving feature importance. At the 50% threshold, it shows high importance for Density and some attention to NDVI. As the threshold increases, Age becomes increasingly important for Gradient Boosting, especially at 70% and 80%, though the magnitude of SHAP values remains lower than those seen in Logistic Regression.

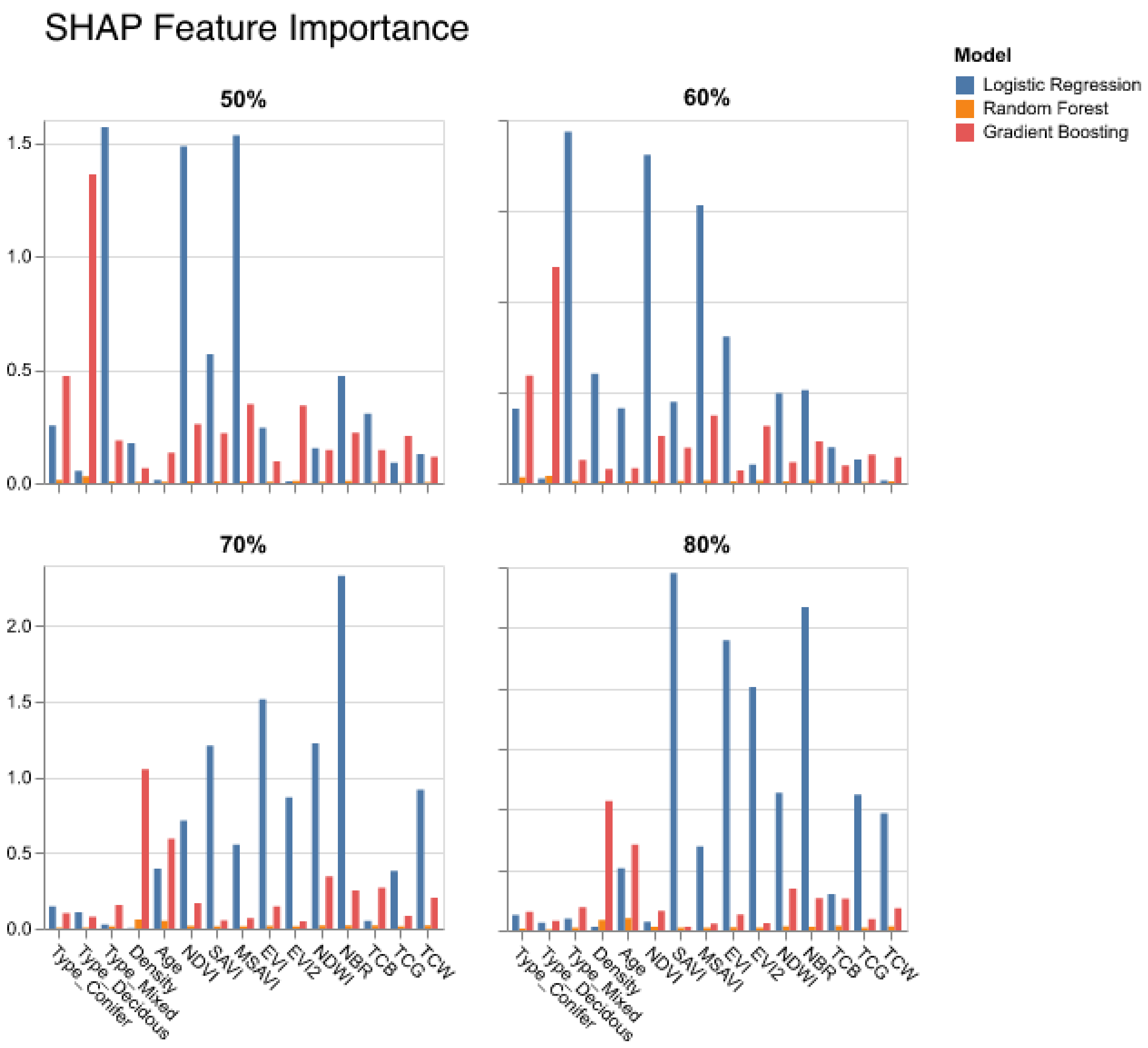

Figure 8: Shap feature importance bar plot

4.1.3 Recursive Feature Elimination Importance

The provided plot as illustrated in Figure 9 illustrates the Recursive Feature Elimination (RFE) importance rankings for various machine learning models, including Logistic Regression, Random Forest, and Gradient Boosting, across different feature selection thresholds (50%, 60%, 70%, and 80%). Each heatmap represents the importance ranking of features, with darker green shades indicating higher importance (rank 1) and lighter shades indicating lower importance (rank 15). At the 50% threshold, Logistic Regression, Random Forest, and Gradient Boosting show a varied distribution of feature importance, with some features like Age and Density appearing more significant across models. As the threshold increases to 60% and 70%, the concentration of important features becomes more pronounced, with Random Forest and Gradient Boosting consistently highlighting certain features, such as EVI and NDVI, as highly relevant.

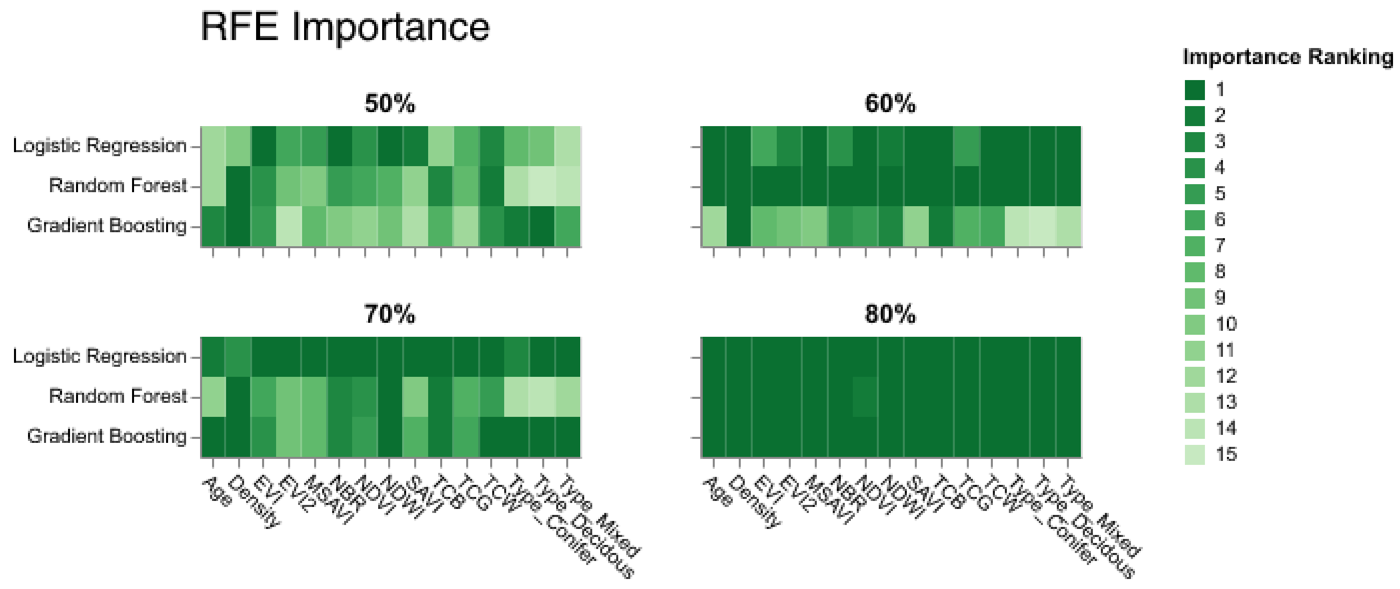

Figure 9: RFE heatmap

At the 80% threshold, the feature importance becomes more focused, with a significant portion of the heatmap dominated by darker green shades, indicating that only a few features retain high importance rankings. This suggests that as more features are eliminated, the models prioritize a core set of variables, such as Type_Mixed and Coniferous, which stand out across all three models. The progression from 50% to 80% demonstrates how RFE refines the feature set, potentially improving model performance by focusing on the most impactful predictors while discarding less informative ones. This analysis highlights the robustness of Random Forest and Gradient Boosting in identifying key features compared to Logistic Regression, which shows a more dispersed importance ranking throughout the thresholds.

In summary, Logistic Regression strongly depends on remote sensing indices across all thresholds, with increasing emphasis as the survival rate cutoff rises. Gradient Boosting balances between structural features like Age and Density, while Random Forest appears to contribute less interpretable feature importance under SHAP, possibly due to model complexity or SHAP limitations with tree ensembles.

4.1.4 Precision-recall Curves

The precision-recall curves as showin in Figure 10 for Gradient Boosting, the curve starts high at low recall values but declines steadily, indicating a strong initial precision that decreases as recall increases. Logistic Regression shows a similar trend, with a sharp drop in precision after a moderate recall level, suggesting it maintains decent performance only at lower recall thresholds. In contrast, Random Forest exhibits a more stable curve, particularly at the 80% threshold, where it sustains higher precision across a broader recall range, reflecting better balance and robustness in classification performance. Overall, Random Forest appears to outperform the other models at higher thresholds, while Gradient Boosting and Logistic Regression show limitations as recall increases.

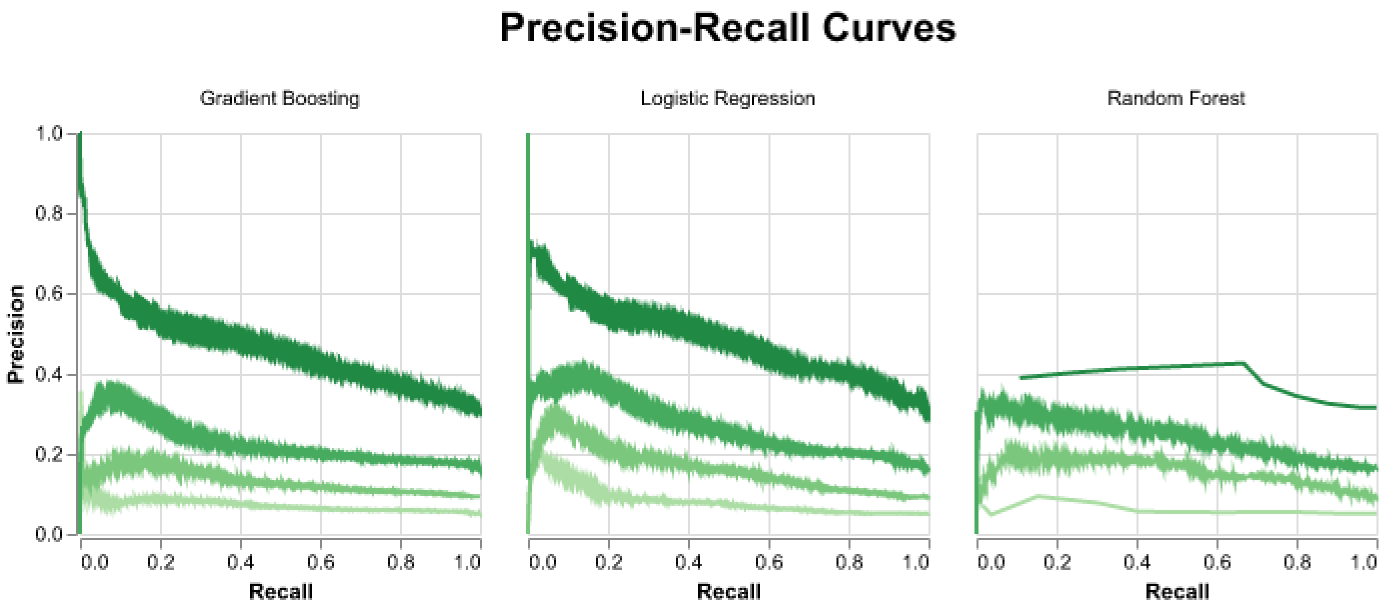

Figure 10: Precision Recall curves for each model across different thresholds

4.1.5 ROC Curves

Our ROC curves as shown in Figure 11 appear more linear rather than hugging the top-left border, suggesting that the models are not performing very well. The linear shape indicates that as we increase the number of true positives, the number of false positives also increases. In a good model, we would expect to find a point with a high true positive rate and a low false positive rate.

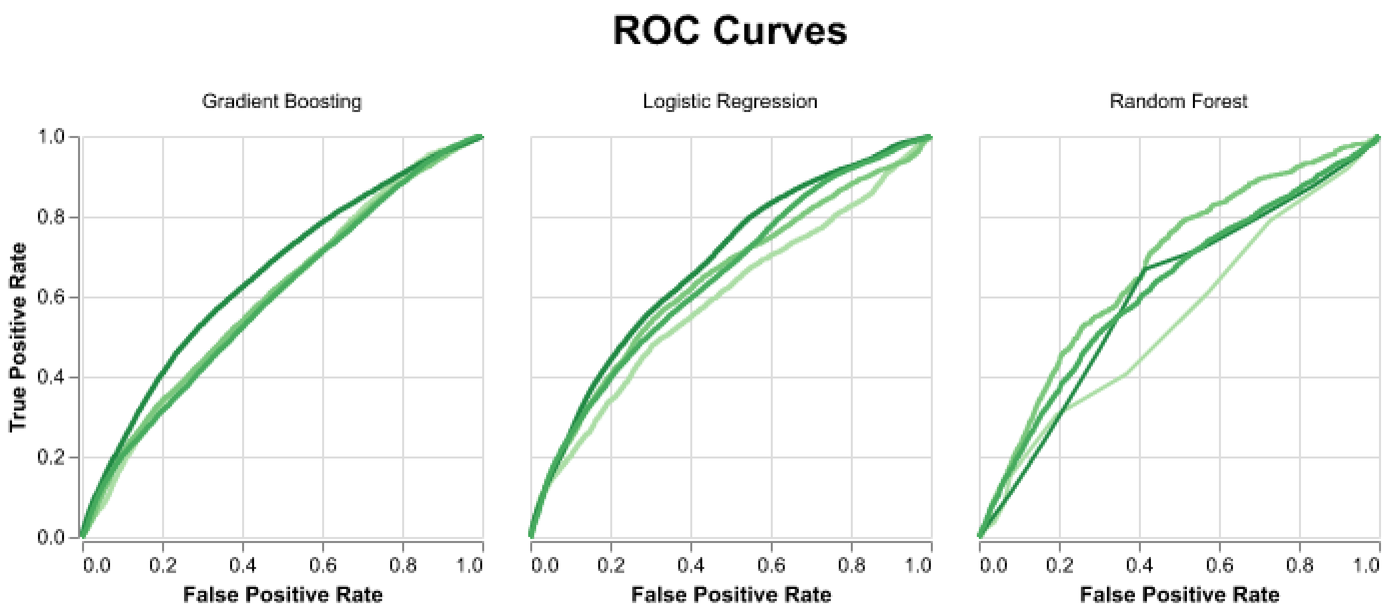

Figure 11: ROC curves for each of model across different thresholds

4.1.6 Confusion Matrices

Confusion matrices as illustrated in Figure 12 reaveals how our classical machine learning models is predicting our true positives which low survival rate. Similar to what ROC curves suggest, we observe more true postives and a relatively fewer false positives at higher thresholds, but false negatives begin to increase rapidly which suggests that the issue is beyond class imbalance.

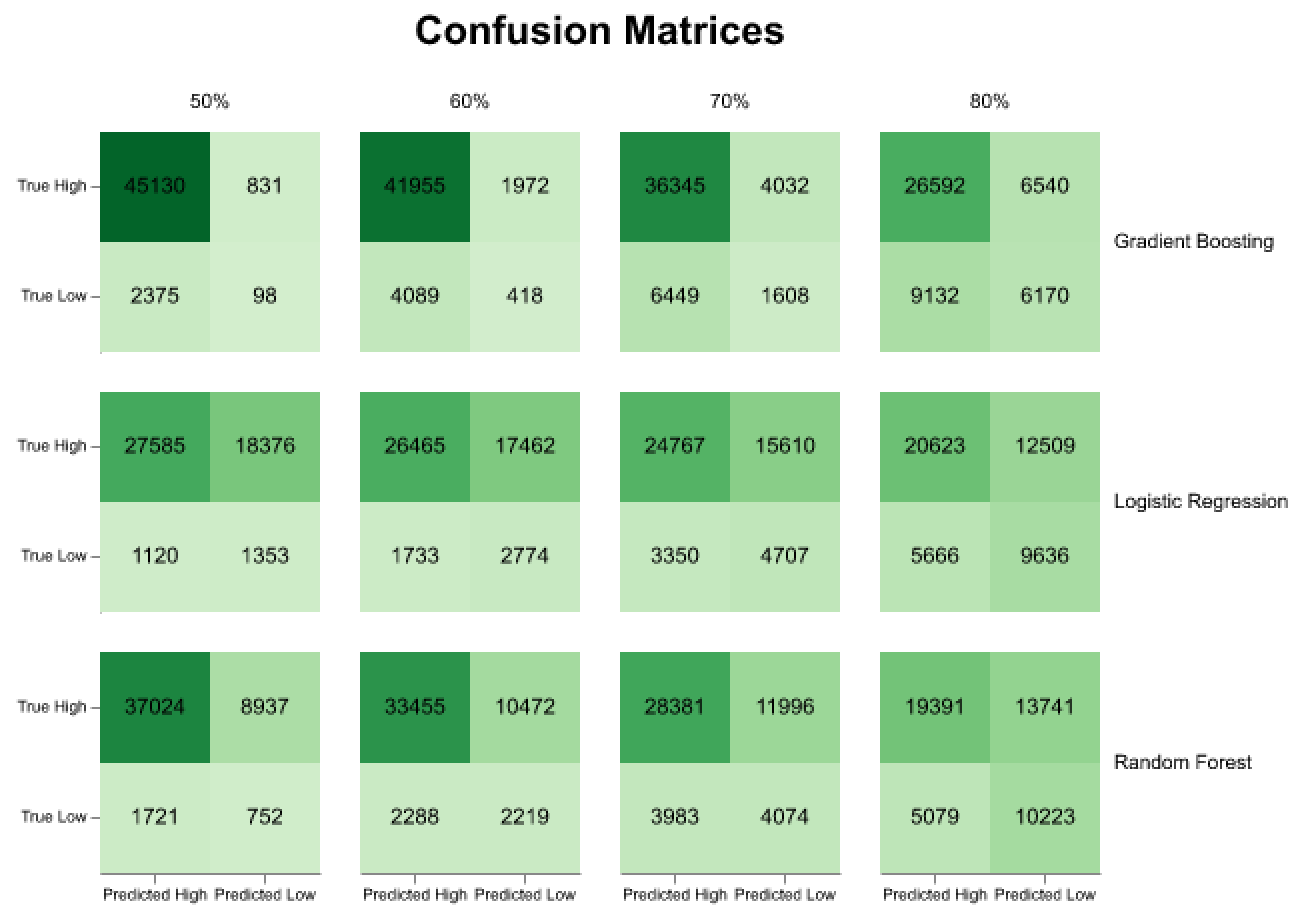

Figure 12: Confusion matrices for each model across different thresholds

4.1.7 Evaluation Metrics

Given the pronounced class imbalance in our dataset, we prioritized the F1 score as our main evaluation metric, as it balances both precision and recall. The F1 scores as shown in Table 4 across different models and thresholds reveal important trends. At the 50% threshold, all models perform poorly: Gradient Boosting achieves an F1 score of just 0.058, Logistic Regression reaches 0.122, and Random Forest attains 0.124. As the threshold increases to 80%, performance improves markedly—Gradient Boosting rises to 0.441, Logistic Regression to 0.515, and Random Forest achieves the highest F1 score at 0.521. These results suggest that both Random Forest and Logistic Regression benefit from higher thresholds, with Random Forest consistently outperforming the others at the upper end. Gradient Boosting also improves but remains slightly behind.

Table 4: Scores at 50% threshold

	Model	Accuracy	F1 Score	F2 Score	AUC	AP
0	Gradient Boosting	0.934	0.058	0.045	0.599	0.072
8	Logistic Regression	0.597	0.122	0.228	0.590	0.083
4	Random Forest	0.780	0.124	0.192	0.545	0.061

Table 5: Scores at 80% threshold

	Model	Accuracy	F1 Score	F2 Score	AUC	AP
2	Gradient Boosting	0.676	0.441	0.417	0.653	0.465
10	Logistic Regression	0.625	0.515	0.578	0.680	0.483
6	Random Forest	0.611	0.521	0.600	0.606	0.386

4.1.8 Conclusion

Despite these improvements, the highest F1 score observed (0.521 with Random Forest) remains modest, underscoring the challenges posed by class imbalance and the inherent complexity of the prediction task. These findings indicate that classical machine learning models may be limited in their ability to capture temporal dependencies within the data. To address this limitation and further enhance predictive performance, more advanced modeling techniques—such as sequence models—are warranted.

4.2 Sequence Model Evaluation

In the second phase, we advanced to deep learning approaches to address the temporal dependencies inherent in our dataset. Specifically, we implemented Recurrent Neural Network (RNN) architectures, including both Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models. These architectures are well-suited for sequential data, as they can capture patterns and dependencies across time steps that classical machine learning models may overlook. By leveraging LSTM and GRU networks, our goal was to improve predictive performance—particularly the F1 score—by enabling the model to learn from the temporal structure present in the survival rate data.

4.2.1 Residual Plots

The residual plots, as shown in Figure 13, for the Satellite and Site-Satellite datasets, featuring the GRU and LSTM models, reveal a distinctive pattern resembling a convex function, with the residuals predominantly centered around the 80% mark on the true value axis. This clustering suggests that both models tend to predict values close to 80% with high frequency across the range of true values, from approximately 30 to 100. Such a concentration indicates a potential bias in the models, where they consistently favor this particular value regardless of the actual data distribution. This behavior is highly undesirable, as it implies the models lack the flexibility to accurately capture the full spectrum of true values, rendering their predictions less useful for practical applications. The tight grouping of residuals around 80% for both GRU and LSTM, with some spread at the extremes, further highlights a limitation in their ability to adapt to diverse data points, undermining their overall predictive reliability.

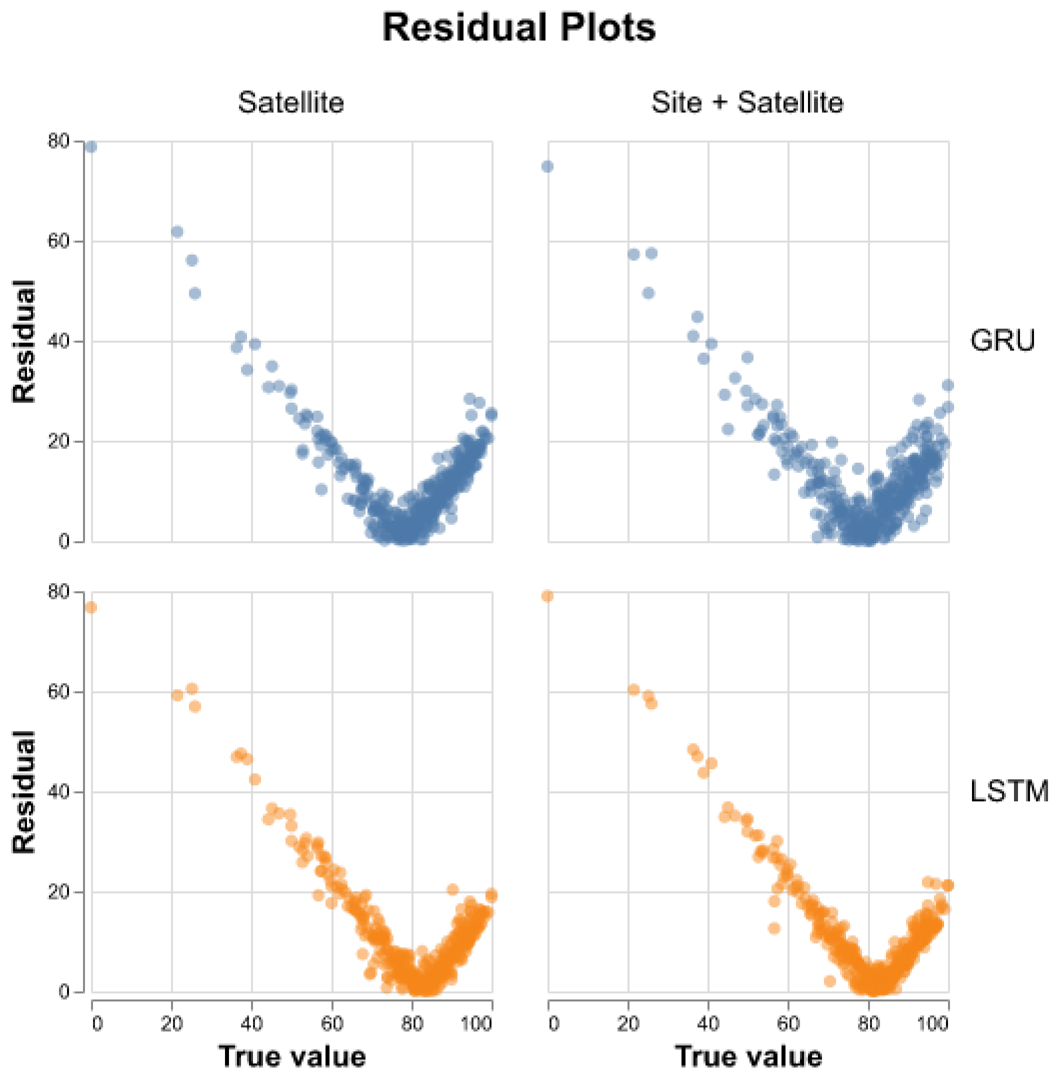

Figure 13: Residual plots

4.2.2 Confusion Matrices

The confusion matrices, as shown in Figure 14, at lower thresholds reveal an intriguing outcome: the model fails to make any correct predictions for lower survival rates, which is unexpected and suggests a significant limitation in its ability to identify these cases. This is likely due to the data being heavily skewed toward higher survival rates, with most data points concentrated around 100%, allowing the model to correctly predict only the high survival rates. At the higher threshold, as shown in Figure 15, of 80%, the model begins to show improvement by making some correct predictions, indicating that it is starting to learn the underlying patterns in the data. However, the number of true positives remains lower compared to classical models, suggesting that while the model is adapting, it has not yet achieved the same level of accuracy or robustness in identifying positive cases across the full range of survival rates.

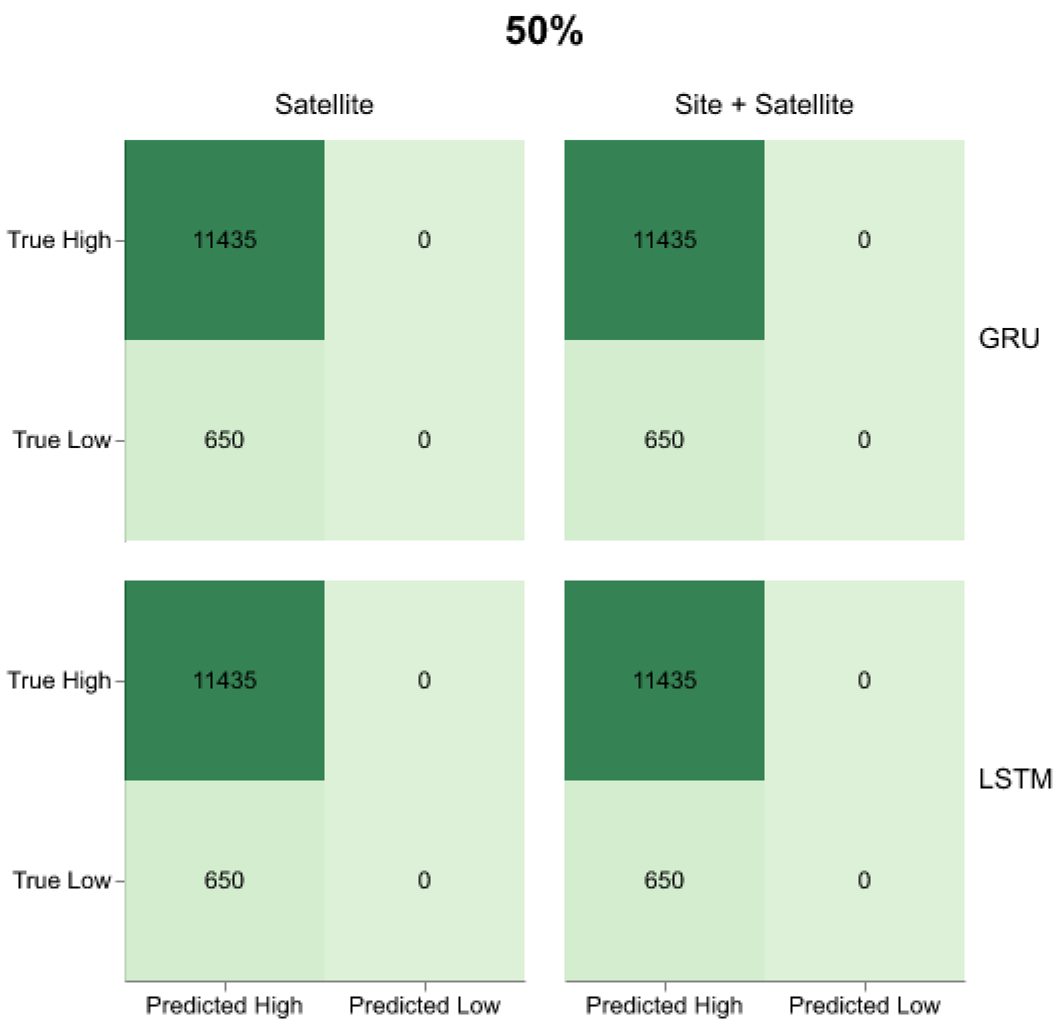

Figure 14: Confusion matrices for 50% threshold

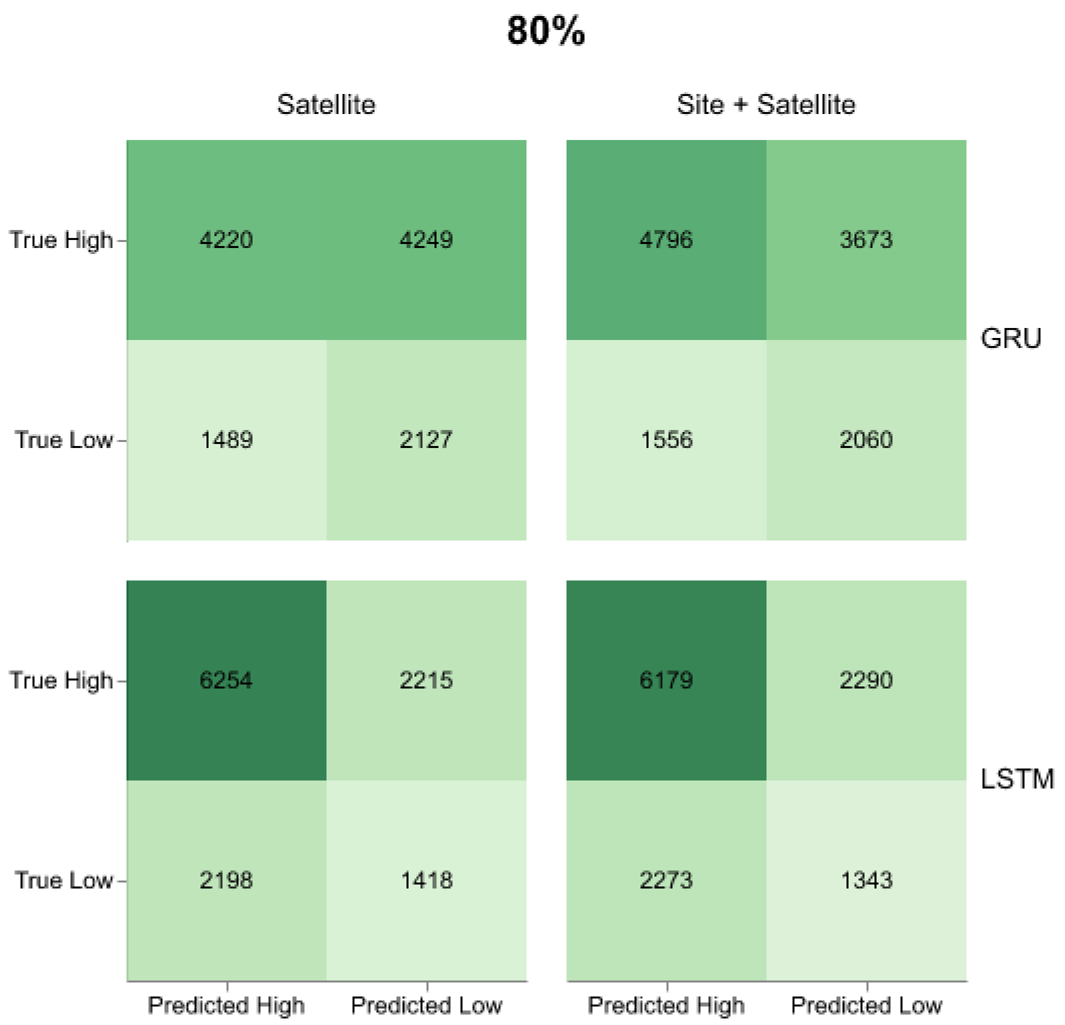

Figure 15: Confusion matrices for 80% threshold

The table highlights the F1 Scores for LSTM and GRU models using Site + Satellite and Satellite features at 50% and 80% thresholds. At the 50% threshold, all models and feature combinations show an F1 Score of 0, indicating no predictive capability. At the 80% threshold, performance improves: LSTM with Site + Satellite features achieves an F1 Score of 0.368, while LSTM with Satellite features reaches 0.393. GRU with Satellite features records a higher F1 Score of 0.434, and GRU with Site + Satellite features attains 0.44, demonstrating the best performance at this threshold. While GRU consistently outperforms LSTM, with notable improvements at the 80% threshold, it fails to outperform our classical machine learning models.

Table 6: Scores for LSTM model without site features

	F1 Score	F2 Score	Precision	Recall	Accuracy
50%	0.000	0.000	0.000	0.000	0.946
60%	0.026	0.017	0.176	0.014	0.907
70%	0.126	0.093	0.295	0.080	0.801
80%	0.391	0.392	0.390	0.392	0.635

Table 7: Scores for LSTM model with site features

	F1 Score	F2 Score	Precision	Recall	Accuracy
50%	0.000	0.000	0.000	0.000	0.946
60%	0.019	0.013	0.129	0.010	0.906
70%	0.128	0.095	0.300	0.081	0.801
80%	0.371	0.371	0.370	0.371	0.622

Table 8: Scores for GRU model without site features

	F1 Score	F2 Score	Precision	Recall	Accuracy
50%	0.000	0.000	0.000	0.000	0.946
60%	0.093	0.082	0.121	0.076	0.869
70%	0.213	0.212	0.214	0.212	0.719
80%	0.426	0.510	0.334	0.588	0.525

Table 9: Scores for GRU model with site features

	F1 Score	F2 Score	Precision	Recall	Accuracy
50%	0.000	0.000	0.000	0.000	0.946
60%	0.112	0.112	0.113	0.112	0.843
70%	0.256	0.266	0.240	0.274	0.714
80%	0.441	0.510	0.359	0.570	0.567

4.2.3 Conclusion

To conclude, both modeling techniques explored—LSTM and GRU—have failed to deliver satisfactory results. Their performance, as sequence modeling approaches, falls short compared to classical modeling techniques, highlighting several potential areas for improvement. This suggests the necessity for acquiring additional data to enhance model training, developing more effective methods to address missing values, and exploring innovative ways to organize and engineer features tailored to our dataset. Addressing these aspects could significantly boost the models' predictive capability and overall effectiveness.

5 Conclusions & Recommendations

5.1 Limitations

Despite exploring both classical modeling and RNN modeling approaches, all our models failed to deliver satisfactory results for predicting tree survival rates. Here we outline the key limitations of our modeling approaches:

1. **Data Imbalance**

   The most significant problem lies with the highly imbalanced target distribution. As mentioned in Section 4, our data was heavily skewed towards high survival rates, where the majority of survival rates ranging above 70%. We believe the lack of low survival rate data was the leading cause for the biased predictions across all of our models. Without sufficient low survival rate data, our model tends to overfit to the majority class, unable to generalise to the low survival rate cases.

2. **Loss of Temporal Information in Classical Models**

   While our classical models performs slightly better than the RNN models, they fail to capture complex temporal structures in the satellite data. Since classical models were not designed to handle sequential data, we had to do extensive aggregation on the dataset. By averaging the satellite data over time, we were losing a lot of vital information, including seasonal variations and short-term vegetation responses.

3. **Lack of Spatial Information in RNN Models**

   Although RNN models can handle the temporal dynamics, our current RNN model lacks the ability to model spatial relationships between pixels. Each pixel is processed independently, ignoring spatial context within the same site. Since survival rates are measured at the site level and neighboring pixels often share similar micro-climate and environmental conditions, this approach likely overlooks key spatial dependencies that influence vegetation response.

4. **Misleading Target Labels**

   While our models were predicting at pixel level, survival records were recorded at site level and assigned uniformly to each pixel within a site. This can mislead the model, especially when the spectral responses within the same site vary significantly. As a result, 'healthy' pixels and 'unhealthy' pixels are assigned identical targets labels, potentially confusing the model during training. This spatial mismatch creates ambiguity: each pixel within a site is assigned the same survival rate, regardless of local conditions or spectral responses. This not only introduces label noise but may also mislead the model during training, especially when individual pixels exhibit vegetation signals inconsistent with the site-level outcome.

5.2 Recommendations

1. **Using Higher Resolution Satellite Data**

   The current satellite data has a resolution of 30m x 30m. Using higher-resolution satellite data may help capture finer details and spatial variations between pixels of the a site, potentially improving model performance.

2. **Obtaining Higher Resolution Field Survival Records**

   Our current field-measured survival rates are recorded only at the site level, while satellite data is available at the pixel level. To improve the precision of our training targets, we recommend collecting survival rates at a finer spatial resolution—ideally at the pixel or sub-site level. When combined with high resolution satellite data, this would allow models to learn localized vegetation dynamics more efficiently, improving model accuracy.s

3. **Obtaining Annual Survival Records**

   For current dataset, most sites only have 3 (Year 1, 2, 5) survival rate records. Imputing these records could improve the temporal continuity of the dataset and allow the RNN models to better capture the dynamics between vegetation growth and spectral variations. Ideally, acquiring additional training data with complete annual survival rate records would substantially enhance the dataset's temporal resolution and modeling potential.

4. **Modeling at Site Level**

   Given the mismatch spatial resolution of the survey data and satellite data, we recommend that future models should aggregate satellite information across all pixels within a site and making predictions per site rather than per pixel.

5. **Incorporating Spatial Data**

   Currently, our dataset does not have any spatial information. We suggest incorporating spatial data such as GPS coordinates to the current dataset. This would allow the model to capture spatial correlations across sites and pixels.

6. **CNN-LSTM model**

   Alternatively, we suggest using raw satellite imagery instead of pre-extracted spectral indices. Using satellite image directly would allow us to utilize convolutional architectures to learn spatial patterns, potentially improving model performance and reducing preprocessing bias.

   We propose exploring a CNN-LSTM architecture as a next step. In this hybrid approach, each site will be represented as a pixel grid. The site grid will first passed through a CNN to extract spatial features. The sequence of CNN outputs corresponding to each time steps is then fed into an LSTM or GRU to capture temporal patterns. The final hidden state can be passed through fully connected layers to predict the survival rate for the entire site. This architecture naturally accommodates both spatial and temporal dependencies, addressing key shortcomings of our current models.

Tip

Rendering Instructions

```
cd reports/technical
conda activate mds-afforest-dev
quarto render report.qmd
open report.pdf
```

Bergmüller, Kai O, and Mark C Vanderwel. 2022. "Predicting Tree Mortality Using Spectral Indices Derived from Multispectral UAV Imagery." *Remote Sensing* 14 (9): 2195.

Chen, Tianqi, and Carlos Guestrin. 2016. "XGBoost: A Scalable Tree Boosting System." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–94. KDD '16. ACM. https://doi.org/10.1145/2939672.2939785.

Greff, Klaus, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. "LSTM: A Search Space Odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28 (10): 2222–32. https://doi.org/10.1109/TNNLS.2016.2582924.

Lundberg, Scott M, and Su-In Lee. 2017. "A Unified Approach to Interpreting Model Predictions." In *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 4765–74. Curran Associates, Inc. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Natural Resources Canada. 2021. "2 Billion Trees Program." https://www.canada.ca/en/campaign/2-billion-trees/2-billion-trees-program.html.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. "On the Difficulty of Training Recurrent Neural Networks." https://arxiv.org/abs/1211.5063.

Paszke, Adam, Sam Gross, Francesco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

Ravanelli, Mirco, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. 2018. "Light Gated Recurrent Units for Speech Recognition." *IEEE Transactions on Emerging Topics in Computational Intelligence* 2 (2): 92–102. https://doi.org/10.1109/tetci.2017.2762739.

Sak, Haşim, Andrew Senior, and Françoise Beaufays. 2014. "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition." https://arxiv.org/abs/1402.1128.

University of British Columbia Master of Data Science Program. 2025. "Remote Sensing for Forest Recovery." https://pages.github.ubc.ca/mds-2024-25/DSCI_591_capstone-proj_students/proposals/Remote_Sensing_for_Forest_Recovery.html.

Wikipedia contributors. 2025. "Logistic Regression — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Logistic_regression&oldid=1291686859.

Zhou, Zhi-Hua. 2025. *Ensemble Methods: Foundations and Algorithms*. CRC press.