Remote Sensing for Forest Recovery: Technical Report

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

2025-06-16

Table of contents

1 Introduction

Monitoring the success of large-scale afforestation initiatives is a critical yet complex challenge especially for early growth stages. Young trees often produce weak spectral signals due to sparse canopies, making them difficult to detect using traditional remote sensing methods. As part of Canada's 2 Billion Trees program—which aims to plant two billion trees across Canadian provinces by 2031—Natural Resources Canada must track survival rates across hundreds of ecologically diverse and often remote planting sites.

This problem is crucial because the ecological and climate benefits of afforestation—such as carbon reduction, and biodiversity restoration can only be realized if newly planted trees survive and grow.

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 qestions, 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 architectures—we aim to determine which techniques provide the most accurate predictions of tree survival rate to support the mission of supporting sustainable forest management and addressing climate change.

2 Data & Pre-processing

We structure our workflow into two phases—static feature modelling and sequence modelling—each with its own preprocessing steps, followed by evaluation using grouped cross-validation.

2.1 Phase 1: Static Preprocessing & Modelling

Preprocessing Steps - load_data.py: Import RDS (~630 MB) and export Parquet (`afforestation_rasters.parquet`) plus CSVs (`planting_records.csv`, `survival_survey.csv`). - **preprocess_features.py:** Drop invalid sites, one-hot encode `Type`, scale `Density` & `Age`; output `clean_feats_data.parquet`. - **data_split.py:** Binarize Year-7 retention (70 %), stratify by `Type`, and split into `train_data70.0.parquet` and `test_data70.0.parquet`.

Static Models

Model	Implementation	Tuned Hyperparameters
Logistic Regression	`LogisticRegression(class_weight="`	`C` (inverse regularization strength)
Random Forest	`RandomForestClassifier(class_weig`	`n_estimators`, `max_depth`
XGBoost	`XGBClassifier(monotone_constraint`	`max_depth`, `learning_rate`, `n_estimators`, `reg_alpha`, `reg_lambda`

2.2 Phase 2: Sequence Preprocessing & Modelling

Preprocessing Steps - load_data.py: Same raw ingest as Phase 1. - **pivot_data.py:** Mask annual rasters by polygon, compute mean of 12 indices per year, save ~15 000 `site_<ID>.parquet` sequence files. - **data_split.py:** Build lookup table (static features + filename + target) and split identically to Phase 1.

Sequence Models

Model	Implementation	Tuned Hyperparameters
GRU	1-layer GRU (hidden_size=32, dropout=0.2) + static features → dense → logit	`hidden_size`, `dropout`, `learning_rate`
LSTM	1-layer LSTM (hidden_size=32, dropout=0.2) + static features → dense → logit	`hidden_size`, `dropout`, `learning_rate`

2.3 Evaluation Metrics

We use grouped 5-fold CV by site ID (random_state=591) on `train_data70.0.parquet`, optimizing **F1 Score**. Final evaluation on `test_data70.0.parquet` reports:

- **Precision & Recall:** trade-off between false positives and negatives.
- **F1 Score:** harmonic mean of Precision and Recall.
- **AUC:** area under the ROC curve for ranking quality.

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 $\beta = [\beta_1, \beta_2, \ldots, \beta_D]$ 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 **?@sec-metrics**.

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 1.

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 3.2.2 for guidance on selecting an appropriate maximum

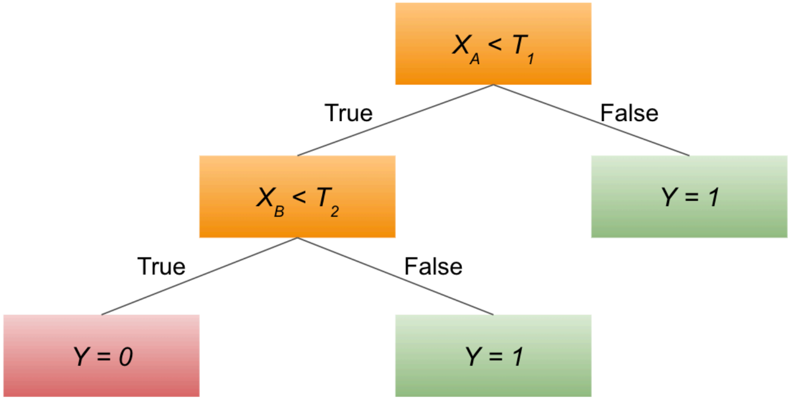

Figure 1: A simple example of a Decision Tree with a depth of 2. The predictions Y are made by comparing selected features X_A and X_B via comparison with threshold values T_1 and T_2.

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. 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:

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

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 3.2.2 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 Gradient Boosting Algorithm.

3.2 Training Classical Models

3.2.1 L1, L2, ElasticNet Regularization

- penalize loss by parameter magnitude for logistic regression
- Logistic Regression: parameters are β
- Gradient Boosting: weights calculated at each leaf in each treeusing higher order derivatives of loss, see paper
- Larger parameter value = smaller weights, less overfitting

3.2.2 Tree Hyperparameters

- Control number of estimators and tree depth for model complexity
- Random Forest: number of features per tree (usually square root of total)
- Gradient Boosting: Learning rate, smaller = less contribution per tree and less overfitting

3.2.3 Cross-Validation

- Grid Search vs Random Search over hyperparameters
- Split data into folds, use one as simulated test fold
- Split by site to avoid leakage between folds
- Choose model with best predefined score, F1 in our case (see metrics section)
- CV feasible for NN training, use holdout validation set instead

3.3 Sequential Deep Learning Models

- Classical models do not naturally handle sequential data
- Literature indicates changes in VIs across time are vital to predicting survival rates
- Problem: require large and diverse datasets.

3.3.1 Recursive Neural Network (RNN)

- Inputs passed sequentially to model
- Each input updates a hidden state via a linear combination: h_t = (W_{xh} * X_t + W_{hh} * h_{t-1})
- Hidden state represents memory data across time
- Vanilla RNNs suffer from vanishing/exploding gradients (see section **?@sec-backpropagation**)

3.3.2 Long-term Short Term Memory (LSTM)

- Uses keep and forget gates to capture long term memory more effectively
- Show equation for updating hidden state with image of cell

3.3.3 Gated Recurrent Unit (GRU)

- Simplified form of LSTM, one combined gate for forget and keep
- Show equation for updating hidden state with image of cell

3.3.4 Bidirectional RNNs

- Add concatenated backward layer for encoding time in both directions

3.3.5 Fully Connected Neural Network (FCNN)

- Linear transformation of input
- Activation functions: control output scales and add nonlinearity (show ReLU function and justify use)

3.3.6 Final Model architecture

- time series into GRU or LSTM
- concatenate with site features and pass to FCNN
- FCNN output goes to sigmoid function * 100 to predict survival rate
- Dropout Rate before prediction, show paper for evidence of improvement
- hidden layers size, size of hidden state, all hyperparameters

3.4 Training Deep Learning Models

- Loss Function: MSE
- Compute gradient of loss WRT to each parameter in model
- Show equation for FCCN params
- show time dependence for sequential parts
- Show gradient descent equation for updating parameters
- Mini-batch: Compute gradients on subsets of data per update step to reduce computational load
- Repeat over succcessive epochs (epoch = 1 run through of data)

3.5 Error Metrics

3.5.1 Precision, Recall, F_β Score

- definition of each
- justification of F1 over other Fbeta and direct accuracy

3.5.2 ROC and PR Curves

- define TPR, FPR, and curves
- AUROC, AUC to measure model performance
- reference curves in results

4 Data Product & Results

Our primary data product is a self-contained, reproducible Python/Quarto repository that provides partner analysts with a turnkey mechanism to (1) preprocess new satellite and silviculture data, (2) train or update models, and (3) evaluate survival-risk predictions for newly planted forest sites. The repository includes:

1. **Python Package (`src/`)**
   - Implements all data-preprocessing steps (`load_data.py`, `preprocess_features.py`, `pivot_data.py`, `data_split.py`), modelling pipelines (`gradient_boosting.py`, `logistic_regression.py`, `rnn.py`), and evaluation routines (`error_metrics.py`).
   - Exposes high-level Make targets so that running `make time_series_train_data` or `make train_models` executes the entire workflow end-to-end.
   - Facilitates future extension: partners can add new features, incorporate additional spectral indices, or replace models without rewriting core scripts.
2. **Versioned Model Artifacts (`models/`)**
   - For the 70% survival threshold, we provide five trained model files:
     - `logreg_model_70.pickle` (Logistic Regression)
     - `rf_model_70.pickle` (Random Forest)
     - `gbm_model_70.pickle` (XGBoost)
     - `gru_model_70.pth` (GRU network)
     - `lstm_model_70.pth` (LSTM network)
   - Each artifact is accompanied by its hyperparameter configuration and training logs, enabling reproducibility and auditability.
   - Partners can load any artifact in a separate environment for inference or further fine-tuning.

4.1 Intended Usage

Partner analysts should leverage this product to:

- **Rapidly assess survival risk** for new planting cohorts as soon as the latest Landsat composites and silviculture records are available.

- **Prioritize field surveys** by focusing on sites flagged as "high-risk" (predicted low survival) to allocate limited labour and remediation resources.
- **Iteratively retrain** models when new Year-7 survey data arrive, ensuring that the classifier adapts to evolving geographic or stand conditions.

4.2 Pros & Cons of the Current Interface

Pros
- **Simplicity**: A single `Makefile` orchestrates the entire pipeline, minimizing command-line complexity.
- **Modularity**: Individual scripts (`src/data/*.py`, `src/models/*.py`, `src/training/*.py`) can be modified or replaced without breaking the workflow.
- **Reproducibility**: Version control of code, hyperparameters, and environment specifications (via `environment.yml`) ensures partners can recreate any result on a new machine.
- **Transparency**: All intermediate Parquet files and logs are stored, making it easy to trace back from final predictions to raw inputs.

Cons
- **Compute Requirements**: Full data-preprocessing and model training (especially the GRU) require significant CPU/GPU resources. Partners without a compatible HPC or GPU-equipped workstation may experience long runtimes.
- **Command-Line Interface**: Although `Makefile` targets simplify execution, partners unfamiliar with command-line tools may face a learning curve.
- **Monolithic Outputs**: The current product writes large Parquet files (~15k per-site series) which can strain disk space.
- **Minimal Web Interface**: There is no interactive dashboard; partners must inspect outputs in Python or Quarto. Developing a web-based front end (e.g., Streamlit or Dash) could improve user experience.

4.3 Justification & Comparison

Compared to alternative approaches—such as distributing only a standalone Jupyter notebook or providing a cloud-hosted API—our package-based product:

- **Reduces dependency on continuous internet access**: All code and data live locally once cloned, so partners in remote offices can run the pipeline offline.
- **Enables customization**: Internal data scientists can incorporate new sensors (e.g., Sentinel-2 or LiDAR) by extending `get_time_series.py` rather than rewriting a monolithic notebook.

- **Avoids vendor lock-in**: No reliance on commercial platforms or paid APIs; the entire stack uses open-source Python libraries.

However, a cloud-hosted API might be preferable if partners require real-time web access or integration with other information systems. Our current approach trades off ease of immediate integration for maximal transparency and reproducibility.

4.4 Results Overview

On held-out (20%) test data for the 70% canopy-retention threshold:

Phase 1: Classical Models

- **Logistic Regression**: performance metrics (Precision, Recall, F1, AUC) will be inserted here.
- **Random Forest**: performance metrics (Precision, Recall, F1, AUC) will be inserted here.
- **XGBoost (GBM)**: performance metrics (Precision, Recall, F1, AUC) will be inserted here.

4.5 Phase 2: Sequence Model Evaluation

- **GRU (spectral+static):** performance metrics (Precision, Recall, F1, AUC) will be inserted here.
- **LSTM (spectral+static):** performance metrics (Precision, Recall, F1, AUC) will be inserted here.

  Insight: Placeholder comment on sequence model performance.

- The **XGBoost model** strikes the best balance (highest F1) for identifying low- survival sites, making it the recommended default for partner deployment.

- The **GRU** achieves comparable AUC and better captures temporal signals, suggesting a potential future improvement once partners can provision GPU resources.

- **Logistic regression** serves as a transparent baseline; its lower AUC and F1 indicate that non-linear interactions among spectral indices and stand attributes are important.

The complete confusion matrix for XGBoost (70% threshold) is:

- **LSTM (spectral+static):** performance metrics (Precision, Recall, F1) will be inserted here.

4.6 Potential Enhancements

- **CNN-LSTM Hybrid:** Add convolutional layers that learn spatial correlations among spectral indices (e.g., local band interactions) before passing the extracted feature maps into an LSTM layer for temporal modeling. This approach can improve predictive accuracy by jointly capturing spatial and temporal patterns in the data.

By focusing on a reproducible, modular product now, we enable partners to adopt and extend the workflow flexibly, while future iterations can add web interfaces and advanced features as computational resources permit.

5 Conclusions & Recommendations

We developed a fully reproducible pipeline covering:

1. **Data Ingestion & Preparation**
   - Loaded raw RDS rasters and planting/survey tables via `load_data.py`.
   - Cleaned static features (`preprocess_features.py`) and extracted per-site time series (`pivot_data.py`).
   - Created consistent train/test datasets with `data_split.py`.
2. **Model Development & Benchmarking**
   - Phase 1: Trained and compared classical classifiers (Logistic Regression, Random Forest, XGBoost) using fixed-length features.
   - Phase 2: Trained GRU and LSTM networks on temporal sequences, demonstrating the benefit of sequence data.
3. **Evaluation & Results Interpretation**
   - Employed grouped 5-fold CV to avoid spatial leakage, optimized F1 for model selection, and reported Precision, Recall, F1, and AUC on held-out data.
   - XGBoost emerged as the top static model; GRU showed promise for temporal modeling.
4. **Operational Recommendations**
   - **Use XGBoost models as a reference** for identifying high-risk sites, rather than for direct operational deployment.

- **Explore GRU models** as proof-of-concept for temporal analysis, acknowledging their current performance limitations.
- **Treat all model outputs as advisory** insights; validate predictions with local surveys before taking field action.
- **Enhance data** through expanded Year-7 surveys and new covariates (soil, LiDAR).
- **Automate retraining** via Makefile and CI to adapt to new data.

By integrating data engineering, machine learning, and reproducible DevOps practices, this workflow provides a scalable tool for early afforestation monitoring and resource prioritization.

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.

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.

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.