Remote Sensing for Forest Recovery: Technical Report

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

2025-06-09

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 Methods

We transform raw afforestation data into training-ready formats and apply two model categories.

3.1 Data Preprocessing

- **Static pipeline:** Drops identifier/date columns, scales 11 spectral indices and density, and one-hot encodes species type into a fixed-length feature vector for each site.
- **Sequence pipeline:** Loads per-site time-series Parquet files, pads sequences to uniform length, applies masking, and adds engineered time features for temporal models.

3.2 Traditional Models

- **Logistic Regression:** Fits a weighted linear model on static features to separate low- vs. high-survival sites.
  - **Hyperparameters:** regularization strength (`C`)
- **Random Forest:** Builds an ensemble of decision trees on static features, aggregating votes to improve robustness against noise.
  - **Hyperparameters:** number of trees (`n_estimators`), maximum tree depth (`max_depth`)
- **XGBoost:** Sequentially trains gradient-boosted trees with monotonicity constraints (on Age) to enhance predictive accuracy on survival.
  - **Hyperparameters:** tree depth (`max_depth`), learning rate (`learning_rate`), number of trees (`n_estimators`), regularization (`reg_alpha`, `reg_lambda`)

3.3 Time-Series Models

- **GRU:** Uses gated recurrent units to capture temporal dependencies in annual spectral sequences before combining with static features for classification.
  - **Hyperparameters:** hidden size (`hidden_size`), dropout rate (`dropout`), learning rate (`lr`)
- **LSTM:** Employs long short-term memory cells with input, forget, and output gates to retain information across multiple years of spectral data.
  - **Hyperparameters:** hidden size (`hidden_size`), dropout rate (`dropout`), learning rate (`lr`)

3.4 Evaluation Metrics

- **Precision & Recall:** Measure false positive vs. false negative trade-offs on held-out test data.
- **F1 Score:** Harmonizes precision and recall, used for cross-validation model selection.
- **AUC:** Evaluates overall ranking performance across threshold choices.

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**
   - **Deploy XGBoost** for immediate risk scoring without specialized hardware.
   - **Incorporate GRU** models where GPUs are available to capture early temporal signals.
   - **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
```