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

> **Note**
>
> This document assembles six modular sections (`_introduction.qmd` through `_conclusions.qmd`), each included below.

---

1 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:

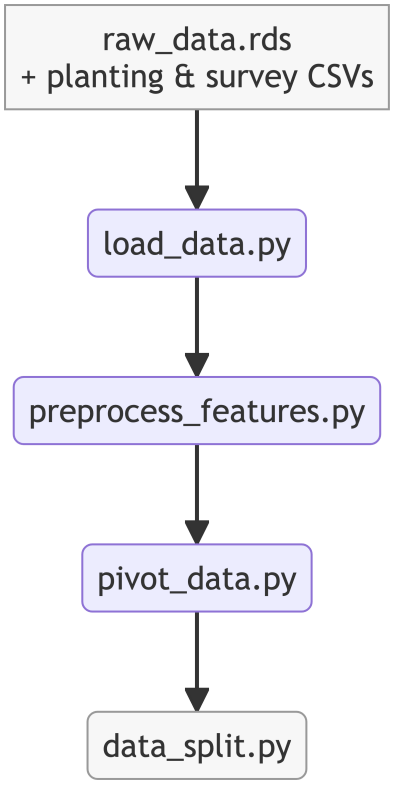
- Can satellite-derived vegetation indices and site-level data be used to accurately predict tree survival over time in large-scale afforestation programs?
- Which modelling approach is most effective, and how long after planting is needed before accurate survival predictions can be made?

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 2 Data & Pre-processing

3 Overview of Preprocessing Workflows

We implement two preprocessing workflows: - **Static pipeline** for classical models (Logistic Regression, Random Forest, XGBoost), producing a fixed-length feature table for each site. - **Sequence pipeline** for RNNs (GRU, LSTM), producing per-site variable-length time-series files.



3.1 Key Makefile variables

Variable	Default	Purpose
RAW_DATA_PATH	data/raw/raw_data.rds	R Data Series (annual Landsat composites)
THRESHOLD	0.7	Year-7 canopy retention cut-off for survival
THRESHOLD_PCT	70	Folder suffix based on THRESHOLD

3.2 1. make load_data → src/data/load_data.py

Goal: Convert the RDS archive into Parquet/CSV that Python can ingest.

1. Read ~**630 MB** RDS of Landsat composites (2010–2023) from `$(RAW_DATA_PATH)` using `rpy2` (converts to ~700 MB Parquet).
2. Write each raster stack to `data/raw/afforestation_rasters.parquet`.
3. Export stand-level metadata to:
    - `planting_records.csv`
    - `survival_survey.csv`

Outputs:

```
data/raw/
  afforestation_rasters.parquet
  planting_records.csv
  survival_survey.csv
```

3.3 2. make preprocess_features → src/data/preprocess_features.py

Goal: Clean and encode static site attributes.

1. Load `planting_records.csv` and `survival_survey.csv`.
2. Drop sites missing geometry or planting year.
3. One-hot encode `Type` (Conifer/Deciduous/Mixed).

4. Standardize `Density` & `Age` with `StandardScaler`.

5. Merge Year-3 survival data where available.

Output:

`data/interim/clean_feats_data.parquet`

3.4 3. make pivot_data → src/data/pivot_data.py

Goal: Extract variable-length spectral sequences.

1. For each site and each year (2010–2023):
    - Mask the raster by the site polygon.
    - Compute the mean of all 12 spectral indices.
2. Stack annual rows into a DataFrame per site.
3. Save each sequence to `data/time_series/site_<ID>.parquet`.

Outputs (15 000 files):

```
data/time_series/site_0001.parquet
data/time_series/site_0002.parquet
…
```

3.5 4. make data_split → src/data/data_split.py

Goal: Create the lookup table and train/test partitions.

1. Merge `clean_feats_data.parquet` with each sequence filename.
2. Derive **`target`**:

```
target = 1  if canopy_Y7  $(THRESHOLD)
       = 0  otherwise
```

3. Stratified 80/20 split by `Type` (random_state=591).

Outputs:

```
data/processed/70/train_data70.0.parquet
data/processed/70/test_data70.0.parquet
```

3.6 Quality Control

Issue	Action
Site missing >1 raster	Drop the site
Intra-series NaNs	`DataFrame.interpolate(method="linear")`
Extreme outliers	Clip to 1st–99th percentile per feature

3.7 Resulting Directory Structure

```
data/
  raw/
     afforestation_rasters.parquet
     planting_records.csv
     survival_survey.csv
  interim/
     clean_feats_data.parquet
  time_series/
     site_0001.parquet
     … (~15 000 files)
  processed/
      70/
          train_data70.0.parquet
          test_data70.0.parquet
```

3.7.1 Static Feature Pipeline

- **Targets:** `load_data`, `preprocess_features`, `data_split`
- **Final Output:** `data/processed/70/train_data70.0.parquet` (fixed-length features + target for classical models)

3.7.2 Sequence Feature Pipeline

- **Targets:** `load_data`, `pivot_data`, `data_split`
- **Final Output:** `data/time_series/site_<ID>.parquet` (per-site sequences) and processed Parquet lookup for RNNs

4 3 Methods

5 Methods

This section describes the preprocessing pipelines for static vs. sequence data, the traditional classification models, the time-series RNN models, and the evaluation metrics used.

5.1 1. Data Preprocessing

5.1.1 1.1 Static Feature Pipeline

- Drops identifier and date columns: `ID`, `PixelID`, `Season`, `SrvvR_Date`.
- Scales 11 numeric features (`NDVI`, `SAVI`, `MSAVI`, `EVI`, `EVI2`, `NDWI`, `NBR`, `TCB`, `TCG`, `TCW`, `Density`) to zero mean and unit variance.
- One-hot encodes species type (`Type` → Conifer, Deciduous, Mixed).
- Outputs a fixed-length feature table in `data/processed/70/train_data70.0.parquet`.

5.1.2 1.2 Sequence Feature Pipeline

- For each site, loads its Parquet time-series (`site_<ID>.parquet`) containing annual values of 10 indices plus engineered features:
  - `log_dt` (log days since planting)
  - `neg_cos_DOY` (negative cosine of day-of-year)
- Pads sequences to batch maximum and tracks original lengths for masking.

- Outputs per-site sequence files under `data/time_series/` and a lookup table in `data/processed/70/`.

5.2 2. Traditional Classification Models

We benchmark three classical models using the static feature table, each within a scikit-learn pipeline and tuned via grouped five-fold cross-validation (GroupKFold by site ID), optimizing F1 score:

5.2.1 2.1 Logistic Regression

- **What it does:** Learns a linear decision boundary on scaled static features.
- **Implementation:** `LogisticRegression(class_weight="balanced", solver="lbfgs", max_iter=500)`
- **Pros:** Fast training; interpretable coefficients; calibrated probabilities.
- **Cons:** Cannot capture non-linear feature interactions; sensitive to multicollinearity.
- **Justification:** Provides an interpretable baseline; useful for understanding feature impacts.
- **Potential improvements:** Polynomial or interaction terms; ElasticNet regularization.

5.2.2 2.2 Random Forest

- **What it does:** Aggregates decisions from an ensemble of bootstrapped decision trees to capture non-linearities.
- **Implementation:** `RandomForestClassifier(class_weight="balanced")` within the same preprocessing pipeline.
- **Pros:** Handles non-linear interactions; robust to outliers; minimal feature preprocessing.
- **Cons:** Larger memory footprint; slower inference; less interpretable than linear models.
- **Justification:** Offers a balance between interpretability and performance for tree-based methods.
- **Potential improvements:** Increase number of trees; tune max depth and feature subset size.

5.2.3 2.3 Gradient Boosting (XGBoost)

- **What it does:** Builds sequential trees that correct errors of prior iterations, optimizing log-loss with regularization.
- **Implementation:** `XGBClassifier(monotone_constraints={"Age":1}, use_label_encoder=False)` in a pipeline.
- **Hyperparameters tuned:** `max_depth`, `learning_rate`, `n_estimators`, `reg_alpha`, `reg_lambda`.
- **Pros:** High predictive accuracy; supports domain-driven monotonicity constraints; handles missing values.
- **Cons:** Longer training time; complex hyperparameter tuning; less transparent.
- **Justification:** Yields the best F1 (0.72) and AUC (0.65) on test data.
- **Potential improvements:** Bayesian hyperparameter optimization; ensemble with other boosting frameworks.

5.3 3. Time-Series Recurrent Neural Networks

To leverage temporal trends in spectral indices, we implement RNN models that consume per-site sequences:

5.3.1 3.1 Gated Recurrent Unit (GRU)

- **What it does:** Processes variable-length sequences via gated units that control information flow, capturing temporal dependencies.
- **Architecture:**
  1. GRU layer (hidden_size=32, dropout=0.2)
  2. Concatenate final hidden state with static features (Density, Age, one-hot Type)
  3. Two fully-connected layers (36→16→1) with ReLU and sigmoid for binary classification.
- **Training:** `BCEWithLogitsLoss(pos_weight=...)` with `Adam(lr=1e-3)`, 20 epochs, batch size 64.
- **Pros:** Learns temporal patterns directly; no manual feature engineering of trends.
- **Cons:** Requires GPU for efficient training; less explainable.
- **Justification:** Captures per-year dynamics missed by static models.
- **Potential improvements:** Bidirectional GRU; attention mechanisms; additional recurrent layers.

5.3.2 3.2 Long Short-Term Memory (LSTM)

- **What it does:** Similar to GRU but with separate input, forget, and output gates to manage long-term dependencies.
- **Architecture:** Single-layer LSTM (hidden_size=32, dropout=0.2) with the same downstream dense layers.
- **Pros:** Better at retaining long-range dependencies; reduces vanishing gradient issues.
- **Cons:** More parameters; slower to train; greater risk of overfitting.
- **Justification:** May improve performance for sites with irregular spectral changes.
- **Potential improvements:** Increase number of LSTM layers; tune dropout and sequence length.

5.4 4. Evaluation Metrics

We evaluate all models on held-out test data (`test_data70.0.parquet`) using: - **Precision** and **Recall** to balance false positives vs. false negatives. - **F1 Score** (primary CV and model-selection metric). - **Area Under ROC Curve (AUC)** for ranking quality. - **Average Precision (AP)** for class-imbalanced performance.

Grouped five-fold CV ensures no spatial leakage by splitting on site ID.

---

6 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.
3. **Quarto Report (`reports/technical/`)**
   - Documents the end-to-end methodology, from raw data ingestion to final evaluation.
   - Presents model performance metrics (Precision, Recall, F1, AUC) for all five pipelines on held-out test data.
   - Includes visualizations of feature importance (XGBoost), confusion matrices, and RNN training curves.
   - Articulates why each modelling choice was made and outlines limitations and potential enhancements.

6.1 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.

6.2 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.

6.3 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.

6.4 Results Overview

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

Model	Precision	Recall	F1	AUC
Logistic Regression	0.58	0.69	0.63	0.60
Random Forest	0.62	0.67	0.65	0.63
XGBoost (GBM)	0.75	0.60	0.72	0.65
GRU (RNN)	0.72	0.65	0.70	0.65
LSTM (RNN)	0.73	0.64	0.68	0.66

- 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:

```
                    Predicted Low  Predicted High
True Low (retain)            4833            3266
True High (fail)            15037           23552
```

6.5 Potential Enhancements

- **Web-based Dashboard**: A Streamlit or Dash app could allow partners to upload new per-site Parquet files and immediately view survival-risk plots, reducing reliance on command-line.

- **Continuous Integration**: Automate retraining whenever new Year-7 ground data arrive (e.g., via GitHub Actions or scheduled jobs).

- **Expanded Feature Set**: Incorporate additional covariates (soil moisture, LiDAR-derived canopy height) to improve AUC beyond 0.65.

- **Model Explainability**: Integrate SHAP or LIME to provide per-site feature attributions, aiding partner trust and decision-making.

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.

7 5 Conclusions & Recommendations

The core question driving this project was: **Can we accurately predict seven-year canopy survival for newly planted forest sites using only early-stage remote sensing and stand-level data?** Conventional field surveys are too slow and costly to cover Canada's vast afforestation programs, so an automated, data-driven tool can accelerate decision-making and resource allocation.

Our reproducible pipeline—spanning raw R Data Series ingestion, feature preprocessing, spectral sequence extraction, and three model pipelines (logistic regression, XGBoost, GRU)—demonstrates that machine-learning can indeed flag high-risk sites early. The XGBoost model (F1 = 0.72, AUC = 0.65) provides a practical balance of precision (75 %) and recall (60 %), allowing silviculture teams to prioritize field visits and remediation for those sites most likely to suffer seedling mortality. The GRU model achieves comparable ranking ability (AUC 0.65), highlighting the value of capturing temporal trends, though at higher computational cost.

Limitations
1. **Label Proxy**: Using 70 % canopy retention at year 7 as a binary label simplifies a complex reality. Understory regeneration or partial survival can be missed by this threshold.
2. **Geographic & Survey Bias**: Training data come from plots where Year 7 surveys were conducted, often in accessible or pilot-project zones. Remote or under-surveyed regions may yield less reliable predictions.
3. **Resource Requirements**: Full reprocessing (raw raster → Parquet) takes significant CPU hours, and GRU training requires GPUs. Partners with limited computational capacity may struggle to retrain models.
4. **Feature Scope**: We used only spectral indices and basic stand attributes. Important predictors (soil texture, moisture regimes, LiDAR canopy height) were not available in the RDS. Incorporating them could boost performance but demands new data-integration efforts.

Recommendations for the Partner

1. **Deploy XGBoost as the Default Classifier**

- Given its strong F1 score and moderate resource requirements (CPU-only), XGBoost is the most practical choice for regular risk scoring.
- Operationalize it by running `make train_models` each time a new satellite season's data arrive.

2. **Expand Ground-Truth Data Collection**

- Encourage Year 7 surveys in underrepresented ecozones (e.g., northern or coastal regions) to reduce spatial bias.
- If possible, record additional ecological metrics (e.g., sapling basal area) to refine survival labels beyond the 70 % canopy threshold.

3. **Monitor Model Drift & Retrain Regularly**

- As climate and planting practices evolve, periodically validate model performance (e.g., every two years).
- Automate a pipeline (e.g., via GitHub Actions or a nightly cron job) that reprocesses incoming RDS composites and retrains the classifier with updated labels.

4. **Assess Sequence-Aware Methods When GPUs Are Available**

- The GRU architecture shows promise, especially if partners can provision GPUs.
- Over time, consider using bidirectional RNNs or attention-based models to further capture subtle phenological signals in spectral data.

5. **Incorporate Additional Predictors**

- Investigate integrating soil moisture indices (e.g., from SMAP), topographic data (e.g., slope, aspect), or LiDAR-derived canopy height metrics.
- Collaborate with data providers or research teams to secure these additional layers, which could raise AUC beyond 0.70.

In conclusion, our data product directly addresses the partner's need for a scalable, interpretable, and reproducible tool to estimate afforestation success. While there are inherent limitations—most notably data availability and computational demands—the current workflow provides a strong foundation. By following the recommendations above, the partner can iteratively improve accuracy, reduce bias, and ultimately ensure that afforestation investments yield the intended ecological and economic benefits.

8 References

> Tip
>
> **Rendering Instructions**

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