COMP5310 Project Stage 2 Report

GROUP 159

Member 1

SID: 510090391

Unikey: lche5181

Member 2

SID: 490051481

Unikey: lshe0103

Member 3

SID: 540930700

Unikey: qzho0498

School of Computer Science

The University of Sydney

Australia

5 May 2025

Group Component 1

1. Topic and research question

This study investigates "how key environmental features—in particular elevation, soil type, designated wilderness area, and distance-to-landmark variables—influence the accuracy of forest cover type classification models". Accurate forest type classification is fundamental in forest ecology for mapping ecosystem distributions and guiding management. Forest ecosystems are essential to biodiversity, and understanding their spatial distribution is critical for conservation and resource planning. For example, remote sensing-based cover maps enable large-scale detection of deforestation, habitat loss, and changes in community composition. Such maps underpin biodiversity assessments and help prioritize conservation areas, while also informing sustainable forestry and wildfire management practices. By quantifying how elevation, soil, wilderness designation, and proximity factors affect predictive models of forest type, we aim to generate actionable insights: improved ecological models that reflect true species–environment relationships, and guidance for policymakers and land managers on where specific forest types are likely to occur under current or changing conditions. Prior work supports this approach: for instance, Vendoti (2025) demonstrated that elevation strongly drives tree-species distribution (emerging as the most important predictor) and that soil types often overlap across cover classes. Understanding these features–cover relationships can therefore help scientists explain model predictions and can inform decisions such as targeting habitat restoration or adjusting management strategies to local topography and soils.

The practical applications of this research span four critical domains. First, in ecological monitoring, automated cover-type maps enable continuous tracking of forest extent and composition, facilitating the detection of deforestation and land-cover changes over time. Second, conservation planning benefits from identifying the spatial distribution of forest types, allowing prioritization of areas requiring protection or restoration—particularly regions harboring rare community types or high biodiversity. Third, resource management applications include guiding sustainable timber harvesting and ecosystem service preservation, where linkages between cover types, elevation, and soil inform targeted fire prevention or logging strategies. Finally, policy support is enhanced through reliable classification systems that aid land-use planning, particularly in identifying vulnerable or resilient forest ecosystems along environmental gradients.

By addressing how these variables affect classification accuracy, this study aims to improve model reliability while delivering practical recommendations for ecologists, foresters, and policymakers requiring precise forest composition maps.

2. Dataset

The dataset used in this project, *lshe0103_A1_CleanDataset.csv*, is derived from *forest_cover.csv* in Assignment 1 and contributed by group member lshe0103. It follows the data cleaning steps detailed in the A1 report, ensuring no missing values. The dataset contains 30,540 rows and 54 features, covering geographical, topographical, and soil-related variables. The target variable, forest cover type, is encoded numerically as: 0–Aspen, 1–Cottonwood/Willow, 2–Douglas-fir, 3–Krummholz, 4–Lodgepole Pine, 5–Ponderosa Pine, and 6–Spruce/Fir. A key challenge is class imbalance: class 4 (48.4%) and class 6 (36.9%) dominate the dataset, while others, such as class 1 (113 records), are severely underrepresented.

Before addressing this imbalance, we examined potential outliers, as resampling techniques like SMOTE are sensitive to extreme values. Generating synthetic data based on unfiltered outliers can reinforce noise or distort patterns, undermining model generalizability. Thus, we prioritized outlier handling to preserve data integrity. Outlier detection focused on four numerical features likely to exhibit large variance: *Elevation*, *Horizontal_Distance_To_Hydrology*, *Horizontal_Distance_To_Roadways*, and *Horizontal_Distance_To_Fire_Points*. We applied the interquartile range (IQR) method, identifying outliers as values beyond Q1 - 1.5×IQR or Q3 + 1.5×IQR. However, removing outliers from *Elevation* would reduce class 1 from 113 to only 3 records, effectively eliminating it. Similarly, excluding outliers from the three distance-related features would remove around 2,000 rows each from classes 4 and 6, suggesting these extreme values may reflect meaningful class-specific characteristics rather than noise. Based on this, we chose to retain all identified outliers.

With this decision made, we tackled the imbalance by combining oversampling and undersampling. SMOTE was used to expand all minority classes to 1,501 samples, while random undersampling reduced classes 4 and 6 to 6,000 and 5,000 samples, respectively. This balanced the dataset while preserving sufficient real data in the dominant classes to retain their variability. After those operations, the dataset only has 18505 rows and columns stay no change. Despite this effort, it is important to acknowledge that synthetic samples may not fully capture the complexity of real-world distributions. As a result, model performance on minority classes may remain limited when applied to unseen data. Nonetheless, our preprocessing reflects a deep understanding of the dataset's structure and limitations, setting a strong foundation for reliable and fair modeling.

3. Setup

3.1. Modelling agreements

All group members train models on the same target variable, the categorical Forest_Cover label (e.g. Spruce/Fir, Lodgepole Pine, etc.). We evaluate model performance using multiple metrics. Specifically, we use the macro-averaged F1-score and overall accuracy as our primary criteria, and we also examine confusion matrices and record training time for each model.

In principle, the macro F1-score combines per-class precision and recall and, by computing an unweighted mean across classes, treats all forest types equally. This is crucial because cover types are imbalanced: a few classes (such as Lodgepole Pine) dominate the data, while others (like Cottonwood/Willow or Krummholz) are rare. Macro-averaging ensures that rare classes contribute equally to the score, preventing them from being overshadowed by common classes. In contrast, overall accuracy (the fraction of all correctly labeled instances) provides a broad measure but can be misleading on imbalanced data: a classifier could miss an entire rare cover class and still achieve very high accuracy. The confusion matrix is also examined to reveal which cover types are frequently misclassified, giving insight into per-class errors. Additionally, we record the training time to compare computational efficiency across methods.

Using both macro F1 and accuracy provides a nuanced assessment. High accuracy alone might hide poor performance on minority classes. In this case, checking F1 score at the same time ensures that precision and recall are balanced for each class. For example, a model that correctly predicts common forest types but fails on rare types would have a lower macro F1 (due to low recall for the rare class) even if its accuracy remained high. The confusion matrix further illuminates these trade-offs by showing which classes are confused. Together, these metrics align with our goal of reliably identifying both majority and rare forest types.

3.2. Data quality assurance and cleaning

We implemented a structured data partitioning and preprocessing pipeline tailored to the ecological characteristics of our forest cover dataset (30 540 samples, 54 features). First, we performed stratified data splitting using scikit-learn's `train_test_split` with `stratify=y` and a fixed `random_state=42` to ensure reproducibility. In the initial split, 80 % of the data (24 432 samples) was allocated to the training set, while the remaining 20 % (6 108 samples) formed a temporary holdout. This choice strikes a balance between having a big training sample for complex models and keeping enough data that hasn't been used yet. In the secondary split, the 20 % holdout was divided evenly into validation and test sets (3 054 samples each), again using stratification. As a result, each subset preserves the exact class proportions found in the full dataset, guaranteeing that all seven cover types, including the rarest, are present in both tuning and final evaluation. The validation and test sizes of roughly 3 000 observations provide adequate power to detect performance differences of ±2 % at the 95 % confidence level.

Next, we addressed the extreme class imbalance in the training data by applying a hybrid resampling strategy. Using imbalanced-learn's `SMOTE` followed by `RandomUnderSampler`, we oversampled each of the five minority classes to 1 501 records and undersampled the two majority classes to 6 000 (Lodgepole Pine) and 5 000 (Spruce/Fir), yielding a balanced training set of 18 505 instances. SMOTE generates synthetic minority-class samples by interpolating between existing feature‑space neighbors to under-represented categories without discarding genuine observations. The subsequent undersampling of majority classes reduces their dominance while preserving critical environmental gradients. We also left the validation and test sets unchanged, so that performance metrics reflect real-world class frequencies and enable unbiased comparisons across models.

Finally, we evaluated whether to remove outliers among the continuous predictors using the interquartile-range method. For removing such observations disproportionately impacted rare classes, undermining the model's ability to learn genuine feature–cover relationships, therefore, we retained all observations to capture the full spectrum of environmental variability.

Stratified splitting ensures that every class, including ecologically significant minorities, contributes to both training and evaluation. The SMOTE + undersampling pipeline balances the learning signal without altering real-world validation and test distributions, thereby safeguarding the integrity of performance assessments. By preserving ecologically meaningful extremes, we maintain the topographic and spatial diversity necessary for interpretable, generalizable models that can inform actionable conservation and management decisions.

Nevertheless, we acknowledge two limitations: synthetic samples may not perfectly mirror the joint distributions of real observations, and oversampling can increase overfitting risk if not carefully validated. To guard against these risks, all performance metrics are reported on the unmodified test set, and subsequent model interpretation explicitly considers these caveats when extrapolating to novel environments.

Individual Component(lche5181)

1. Predictive model

1.1. Model Description

For this forest cover type prediction task, I selected TabNet as the predictive modeling technique. TabNet, proposed by Arik and Pfister (2019), is an attention-based neural network architecture specifically designed for tabular data. TabNet leverages sequential attention to select a subset of input features at each decision step, combining them through a feature transformer and an attentive transformer. This design enables TabNet to use its modeling capacity on the most relevant features, yielding both high performance and built-in interpretability via feature masks. It also has limitations including a large number of hyperparameters to tune and relatively slower training compared to tree-based methods, especially for smaller classes. The forest cover data after we preprocessed has a mix of continuous (elevation, horizontal distance to xx) and high-dimensional categorical (40 one-hot soil types, 4 one-hot wilderness areas) inputs, so TabNet's ability to handle mixed feature types and automatically learn sparse feature masks makes it a suitable choice.

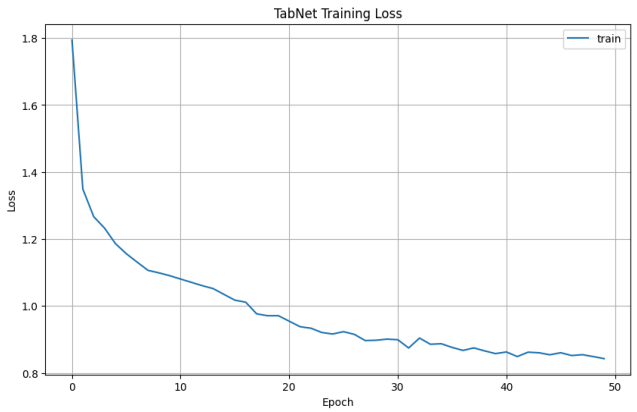
1.2. Model Algorithm

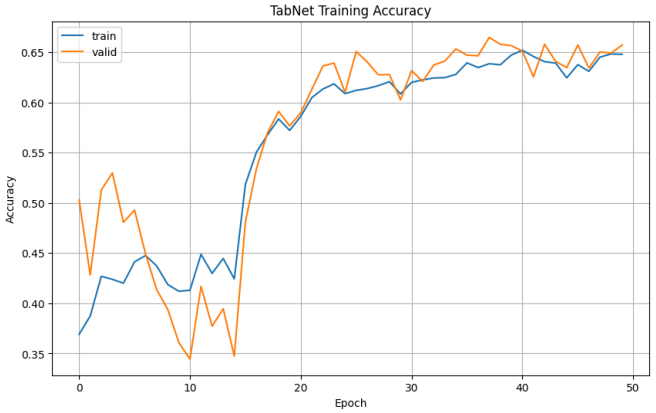
TabNet's algorithm proceeds by iteratively selecting features and transforming them through multiple decision steps. At each decision step *i*, an attentive transformer computes a sparse feature mask $M[i]$ using a learned linear mapping followed by Sparsemax normalization. Formally, let $P[i-1]$ be a prior scale vector encoding how much each feature has been used so far, and let $a[i-1]$ be the representation from the previous step. The feature mask is computed as $M[i] = Sparsemax(P[i-1] \cdot h_i\, a[i-1])$, where h_i is a trainable linear layer and *Sparsemax* promotes a few nonzero entries. This mask $M[i]$ is applied elementwise to the raw input features, yielding a subset of selected features. These selected features are passed through the feature transformer to produce a new latent vector. That latent output contributes to the model's final prediction and also serves as $a[i]$ for the next step. After each step, the prior is updated by $P[i] = \prod_{j=1}^{i} (\gamma - M[j])$ where $\gamma > 1$ is a feature reuse parameter. If $\gamma = 1$, any feature used at one step is forced to zero in subsequent steps, otherwise higher γ allows features to be reused. This sequential attention mechanism combining learned masks with feature transformations is the center of TabNet's model. The final classification output is formed from the sum of all contributions from all steps.The figure with pseudocode outlines this process.

```
# Pseudocode of TabNet for classification
Initialize prior P = [1,...,1]  # one per feature
for i in 1 to n_steps:
    # Compute feature mask via attentive transformer
    mask = Sparsemax( P * Linear(att_rep) )
    # Select features and transform
    selected = mask * X_input
    h = FeatureTransformer(selected)    # deep FCN layers
    # Aggregate output for classification
    if i == 1:
        aggregated = h
    else:
        aggregated += h
    # Update prior to reduce used features
    P = P * (gamma - mask)
    # Set representation for next step
    att_rep = h
final_output = PredictionHead(aggregated)
```

1.3. Model Development

The input features of the training set, validation set and test set were split into numeric(elevation, horizontal distance to xx) and categorical (soil types, wilderness). A preprocessing pipeline applied StandardScaler to the numeric column for standardisation and passed through the one-hot (already encoded in dataset) categorical features. Before training, all feature matrices were converted to NumPy float32 arrays for PyTorch compatibility. Then, the TabNet model created via TabNetClassifier from the Pytorch-TabNet library was instantiated with default settings. The key hyperparameter of TabNet include n_d (decision layer width) and n_a(attention layer width), controlling the size of the feature transformer and attentive transformer, respectively; the number of decision steps n_steps; and the feature-reuse coefficient . Typically one sets n_d=n_a and chooses 3–10 steps. Large n_a & n_d usually increase model capacity (risking overfitting) while more steps allow more sequential reasoning. For baseline model training, I use default hyperparameter setting with n_a & n_d = 8, n_steps = 3 and gamma =1.3. The model is training on training set and valid on validation set with 50 epochs and a large batch size of 1024. Training progress is monitored by plotting the loss and accuracy curves(see Figures below





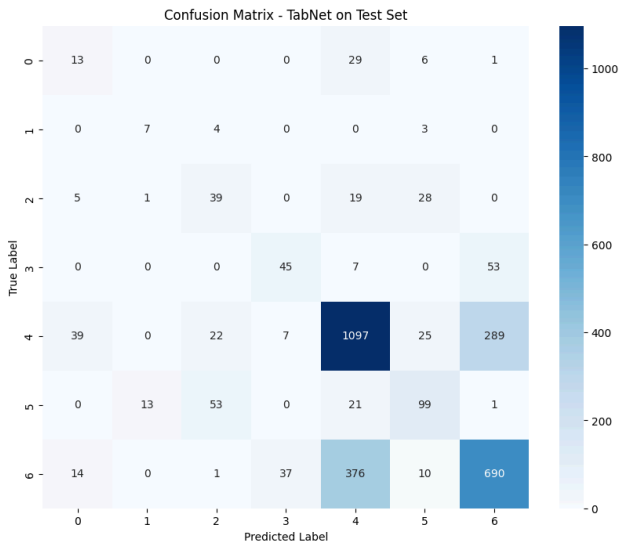
As shown in above curves, the training loss fell and training accuracy rose over epochs after initial struggles, and the validation accuracy tracked closely. These plots confirmed that the learning rate and batch size is reasonable.

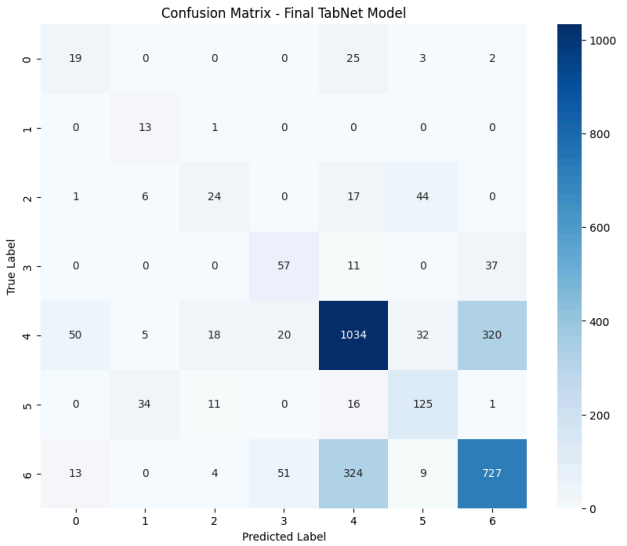
2. Model Evaluation and Optimization

2.1. Model Evaluation

After first baseline training, I used the TabNet model on the test set to predict the forest cover and achieved an accuracy of 65.16%. To evaluate the model performance more comprehensively , I use the classification_report function from sk-learn library. It has multiclass metrics with accuracy, recall, f1-score and support on the validation and test set. Each of these metrics exposes a different facet of performance, so reading them together gives a balanced picture. Accuracy is the share of all predictions that are correct, it tells us how well the classifier performs on the dataset as a whole, but it does not show whether the minority classes are being ignored. In TabNet's first train, the model has more accuracy in predicting class 3, 4, 5, 6 but less on 0, 1, 2. Recall measures the proportion of the true examples of a class that the classifier retrieves. Low recall therefore flags classes the model is overlooking. In the first run the rarest cover is class 0, record recall of 0.27. It means that most of these stands are missed. F1-score is the harmonic mean of precision and recall, it falls sharply whenever either component is low. F1 is preferred to accuracy for imbalanced data because it penalises classifiers that neglect small classes while still rewarding correct classification of the majority. The macro-average F1 of 0.48 highlights the gap between overall accuracy and minority-class performance; the weighted average 0.65 sits closer to accuracy because it is dominated by the frequent classes. Support is the raw number of test samples belonging to each label. It is not a quality measure in itself but it grounds the other statistics. When support for a class is only a few dozen rows, even seemingly large swings in precision or recall may not be statistically reliable. e.g. the class 1 has just 14 test instances, explaining both its volatile metrics and why accuracy alone masks the class's true behaviour. Taken together these figures show that the baseline TabNet is dependable for the dominant cover types (high accuracy, high weighted F1) yet unreliable for rare covers (low per-class recall). The combination of recall and support pinpoints exactly where improvement is needed, and the macro F1 summarises the imbalance-aware performance that matters for research questions concerned with all forest classes rather than just the common ones.

Beside, I also visualise a confusion matrix for the test set. The matrix complements the numeric metrics by showing, cell-by-cell, how the predicted labels are distributed against the true labels. On the test set the diagonal squares for cover types 4 and 6 are the darkest, confirming the high precision and recall that those classes already shown in the classification report. Off diagonal clusters expose systematic confusions: spruce-fir (class 0) is mislabelled as ponderosa pine (class 4) in twenty-nine cases, and more than a third of true lodge-pole pine (class 3) slides into the denser-crown for category (class 6). Thus, the confusion plot simultaneously validates that the network has learned meaningful structure and warns that the learned boundaries are still too coarse.





2.2. Model Optimization

After completing the development and evaluation of the baseline model, I performed a hyperparameter grid search for model optimization. The search loop used sklearn.model_selection.ParameterGrid to generate every configuration of the four most influential architectural parameters as mentioned in 1.2. model development visible to the PyTorch-TabNet constructor: The widths n_d and n_a control the size of the gated linear units in the feature and attentive transformers: wider blocks can model higher-order interactions among the forty soil flags and the single elevation feature. The depth n_steps dictates how many successive attention masks are formed; extra steps allow the network to assign different feature subsets to different decision layers, a mechanism that is particularly useful for classes that hinge on small soil distinctions. The reuse coefficient gamma shapes exploration: at $\gamma = 1.0$ the model may revisit an informative column in later steps, whereas larger values force it to seek new features. The 'n_d & n_a' contains 8, 16, 32, 64, 'n_steps' contains 3, 5, 7, 10, 'gamma' contains 1.0, 1.3, 1.5, 2.0. This covered 64 hyperparameter combinations. For each combination I instantiates a fresh TabNetClassifier, trains for 50 epochs and evaluates on the validation set with accuracy as the primary score. Since TabNet as a deep learning neural network is more complex compared to models generally applied to tabular data, it took a longer time(almost 1 hour) to train a model with grid search 64 different hyperparameter combinations.

The search finds the best hyperparameter combination with n_d = 32, n_a = 32, n_steps = 3, $\gamma = 1.0$, which achieves a validation accuracy of 0.6863. Two findings emerge from the full result table. First, validation accuracy rises monotonically with embedding width up to thirty-two units, confirming that the default width of eight was too small for the forty-one-dimensional input space. Second, adding more than three decision steps offers little benefit unless γ is set low, because high reuse penalties prevent later masks from selecting genuinely different columns. The winning tuple therefore balances capacity and diversity: the wider blocks capture nonlinear soil patterns, three steps avoid redundant masking, and $\gamma = 1.0$ lets the model reuse any truly predictive feature.

Then I applied the best hyperparameters to retrain the TabNet with 100 epochs. After training, the code reports a final accuracy of 66.96% in the test set with 1.8% improvement compared to the baseline model. More importantly, the updated confusion matrix records additional true positives for the two forest covers: spruce–fir correct predictions rise from the low teens to the high teens, and lodge-pole pine almost doubles, which translates into a tangible macro-F1 lift when the full classification report is regenerated after fine-tuning.

In summary, the grid search over hyperparameter space found the best validating configuration, improved minority class recall without damaging overall accuracy, and preserved TabNet's explanatory masks. These improvements, though incremental in absolute numbers, significantly increase the model's suitability for ecological decision making on the forest cover dataset.

Individual Component (lshe0103)

1. Predictive model

1.1. Multi Layer Perceptron Model Description

I use a Multi-Layer Perceptron (MLP) for this classification task. As noted by Gupta and Tapia (2006), MLP does not assume any underlying data distribution, unlike probabilistic models, making it flexible for real-world datasets, including those with class imbalances. While class imbalance is a known challenge, MLP's distribution-agnostic nature reduces its bias toward majority classes. However, it does assume complete and valid input data. To meet this, I ensured no missing or NaN values through careful preprocessing, confirming the dataset's suitability. MLP's simple architecture—comprising input, hidden, and output layers—enables easy implementation. It also captures complex, non-linear relationships with minimal feature engineering. For instance, the 'Horizontal_Distance_To_xxx' feature, initially overlooked in A1, improved model accuracy by ~20%, illustrating MLP's ability to identify non-obvious patterns. Despite its strengths, MLP has limitations: fully connected layers can be inefficient for large datasets and lack interpretability compared to decision trees. Fortunately, this dataset is relatively small, enabling fast training, and since the focus is on identifying general patterns rather than explaining individual predictions, transparency is less critical. Overall, MLP is well-suited to this task and data set. It supports one-hot encoded features, requires minimal distributional assumptions, and effectively models hidden relationships—aligning with the study's goal of understanding how elevation, soil type, wilderness area, and distance indicators influence forest cover type.

1.2. Model Algorithm

A MLP is a typical feedforward neural network composed of an input layer, multiple hidden layers, and an output layer. The connections between each layer are fully connected, which is a primary reason for the longer training time associated with this architecture. Each training epoch consists of two stages: the forward pass and the backward pass. More details about the process please refer to the following pseudocode. There are also several MLP variants, such as Residual MLPs or MLP-Mixer architectures, which aim to enhance gradient flow or capture global information more effectively. To provide more background on the forward pass, we can introduce the formula: $h=\Phi(Wx+b)$. Here, Φ represents the activation function, which is used to introduce non-linearity. W is the weight matrix, x is the input vector, b is the bias term, and h is the output of the current layer. For each layer (except the input layer), x refers to the output from the previous layer. As for backpropagation, the key idea behind it is the chain rule of calculus. The algorithm computes the loss based on the output of the final layer and then propagates the error backward through the network, layer by layer, updating the weights to minimize the loss. The general formula for updating the weights during backpropagation is as follows: $W_{pq}^{new} = W_{pq}^{old} + \triangle W_{pq}$, p refers to a neuron in the previous layer, q refers to a neuron in the current layer, W is the weight matrix, and ΔWpq represents the change applied to the weight.

Common MLP hyperparameters include input size, number of hidden layers, learning rate, momentum, weight initialization, dropout rate, activation function, number of epochs, and optimizer. The number of input neurons should match the input tensor's dimensions to ensure all features are captured. Increasing hidden layers enhances learning capacity but may lead to overfitting and longer training. Learning rate determines step size—small rates improve stability, while large rates speed up training but risk divergence. Momentum helps accelerate convergence and reduce oscillations. Weight initialization methods (e.g., Xavier, He) improve training stability by preventing vanishing or exploding gradients. Dropout rate randomly deactivated neurons during training to reduce overfitting by enhancing generalization. Activation functions add non-linearity; ReLU, Leaky ReLU, or GELU are often used to mitigate vanishing gradients. Epochs define how many times the model iterates over the dataset—more epochs improve learning but risk overfitting. Optimizers like SGD, Adam, and RMSprop update weights based on gradients and vary in performance depending on the task. Pseudocode for the MLP algorithm is presented on the next page.

Recent MLP developments focus on activation function design and optimization. Early choices like sigmoid and tanh suffered from vanishing gradients when inputs became extreme (Lakshminarasimhan, 2025). ReLU, introduced to address this, passes positive values and zeros out negatives but can lead to the "dying ReLU" issue. Variants like Leaky ReLU allow small gradients for negative inputs, while ELU smooths the curve using an exponential form. These improvements are directly applicable to this project. For example, since the dataset includes elevation values that may be negative, testing different activation functions (e.g., via grid search) is important to find the most appropriate one for the input distribution.

1.3. Model Development

Before building the model, I chose standardization over normalization because the columns in the dataset are not inherently related, and PCA could potentially discard useful information. Standardization, which scales features to have a mean of zero and a standard deviation of one, is commonly used in classification tasks, particularly in MLP models. It ensures that all numeric features (i.e. elevation, horizontal_distance_to_xxx) are on the same scale, preventing features

with larger magnitudes from dominating the model, leading to more balanced learning. I opted not to use PCA for

1. Initialize the network weights randomly (W, b)
2. Set a threshold for error (e.g., 0.01)
3. Set maximum number of epochs (e.g., 1000)
4. For each epoch (until error ¡ threshold or max_epochs):
   (a) Initialize total error = 0
   (b) For each training example (x, t), where $x = \{x_1, x_2, \ldots, x_n\}$ and t is the target:
      i. Forward Pass:
         A. Input x through the network
         B. For each layer l (input layer to output layer):
         C. Calculate the weighted sum:
$$z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$$
         D. Apply the activation function:
$$h^{(l)} = \varphi(z^{(l)})$$
         E. Calculate output $o = h^{(L)}$ from the output layer
      ii. Compute the error at the output layer:
$$\text{error} = (o - t)$$
      iii. Backward Pass:
         A. Compute the gradient of the error with respect to the weights and biases for each layer
         B. Update weights and biases using the gradient (e.g., using gradient descent or another optimizer)
         C. For each layer l from the output layer to the input layer:
         D. Compute the gradient of the error with respect to the activations
         E. Update the weights using the calculated gradients
      iv. Add the error for this training example to the total error
   (c) Calculate the average error over all examples
   (d) If the average error ¡ threshold, stop training

dimensionality reduction since the dataset dimension is already small, and reducing dimensions could lose valuable information. Given that MLP models require minimal feature engineering, I did not perform any additional transformations, as the data is already clean, with no missing values, and categorical data is one-hot encoded. For hyperparameters, I focused on the hidden layer's dimension, number of epochs, number of hidden layers, activation function, optimizer, learning rate, and dropout rate. The input and output dimensions remain fixed, tied to the data and label classes. I chose the hidden dimension to test if more neurons improve performance, and the number of epochs to find the optimal training time to avoid overfitting or underfitting. The number of hidden layers was varied to explore whether a deeper or simpler model works better. I selected the activation function (e.g., ELU) to prevent vanishing gradients and enable efficient learning. The optimizer was chosen based on empirical testing to identify the most effective update method. While the learning rate is not expected to have a large impact, I ensured it allows proper convergence (between 0.1 and 0.001). Finally, the dropout rate was set to avoid overfitting, particularly given the imbalanced training data, ensuring better generalization to unseen data. By testing and fine-tuning these hyperparameters, I aim to balance model complexity and performance, ensuring good generalization to real-world data.

To implement this model, I use PyTorch and define an MLP class that inherits from nn.Module. In the __init__ function, I first create an input layer consisting of a linear transformation followed by an activation function, using the syntax layers = [nn.Linear(input_dim, hidden_dim), activation()]. A list is used to sequentially store all layers, including hidden and output layers. For each hidden layer, I add a linear layer, an optional dropout layer (if the dropout rate is greater than 0), and an activation function in that order. Specifically, the components are nn.Linear(), nn.Dropout(), and activation(), which are appended to the list. After constructing the hidden layers, I add the output layer using nn.Linear(hidden_dim, output_dim) to map the final hidden representation to the desired output dimension. The entire list of layers is then wrapped into an nn.Sequential module using self.net = nn.Sequential(*layers), simplifying the forward pass implementation. In addition to the layer definitions, I utilize built-in PyTorch functions such as eval() to set the model to evaluation mode and train() to switch back to training mode. I also use torch.no_grad() to disable gradient tracking during inference, improving performance and reducing memory usage. To obtain the predicted class labels, I apply torch.max(test_outputs, 1), which returns the indices of the maximum values along the class dimension, representing the model's predicted classes. The model is trained using nn.CrossEntropyLoss() as the loss function, which is suitable for multi-class classification problems, as it computes the cross-entropy loss between the predicted logits and the true class labels.

2. Model Evaluation and Optimization

2.1. Model Evaluation

In the following part, class 0 to 6 means different forest cover (more detail in group section 1 data part).

The initial model has three hidden layers with 64 units each, ReLU activation, a learning rate of 0.001, no dropout, and is trained for 10 epochs using the Adam optimizer. It achieves 0.42 accuracy on the test set and 0.33 on the training set, indicating underfitting. Besides Accuracy, I also use MCC, F-1 score, precision, recall to evaluate the model. From the confusion matrix, only class 4 (Lodgepole Pine) is learned relatively well (F1-score: 0.62), while class 6 (Spruce/Fir) is poorly predicted (F1-score: 0.067), and the other five classes have F1-scores of 0. Since the task requires both avoiding incorrect predictions (precision) and ensuring important cases are not missed (recall), F1-score is the most appropriate evaluation metric, as it balances both and is more robust to class imbalance than accuracy. To evaluate the model's overall performance, I also used MCC, which ranges from -1 to 1. The initial model had an MCC of -0.049 (no predictive capability). The model shows strength in capturing the dominant class (precision: 0.49, recall: 0.83), but fails

completely on minority classes, misclassifying most of their samples as the majority class. This is likely due to severe class imbalance. Such bias significantly limits the model's real-world applicability — for instance, in forest classification tasks, relying on a model that favors dominant species may mislead ecological planning and resource allocation. In conclusion, the model cannot adequately explain the research question. To address the most pressing issue, underfitting, the model could be improved by increasing the number of training epochs (10 is insufficient) and expanding the size of the hidden layers to enhance its learning capacity. Regarding class imbalance, mitigation strategies such as class resampling have already been implemented during data preprocessing, constraining further interventions at the model training stage.

2.2. Model Optimization

To optimize the model, I employed grid search in combination with 10-fold cross-validation, as this approach allows for a thorough and reliable exploration of the hyperparameter space. Grid search systematically evaluates all possible combinations of predefined hyperparameters, ensuring that no potentially optimal setting is overlooked. Although more computationally expensive than alternatives like random search, grid search is feasible in this case due to the small dataset and fast training speed (approximately one second per model). This makes exhaustive search not only possible but also advantageous, as it provides a complete performance landscape. To ensure that performance evaluations were robust and not dependent on a particular data split, I used 10-fold cross-validation rather than a single hold-out set or a smaller k value. Compared to 3-fold cross-validation, 10-fold offers a better trade-off between bias and variance in performance estimates. It reduces the risk of overfitting to any specific fold and provides a more stable and generalizable assessment of each hyperparameter combination's effectiveness. The hyperparameter grid was defined as follows: hidden_dim $\in$ {64, 128}, epochs $\in$ {50, 100, 200}, num_hidden_layers $\in$ {1, 2, 5, 10}, activation $\in$ {ReLU, Tanh, ELU}, optimizer $\in$ {SGD, Adam}, learning_rate $\in$ {0.01, 0.001}, and dropout $\in$ {0.0, 0.01, 0.1}. These choices result in 864 unique combinations. With 10-fold cross-validation applied to each, a total of 8640 models were trained. Given the low computational cost per model, this level of exhaustive evaluation was practically achievable. Each hyperparameter and its values were selected with specific considerations. The number of neurons per hidden layer (hidden_dim) was set to 64 or 128 to balance computational efficiency and model capacity. The number of hidden layers varied from 1 to 10, allowing investigation of both shallow and deep architectures. Epoch counts were capped at 200 to avoid overfitting while allowing enough training time for convergence. I tested ReLU, Tanh, and ELU activation functions due to their different behaviors in gradient propagation and stability. Optimizers included SGD and Adam to compare traditional versus adaptive optimization. Learning rates of 0.01 and 0.001 were chosen to promote convergence without instability. Dropout rates were introduced to mitigate overfitting and support generalization, particularly important given the dataset's class imbalance. Overall, this comprehensive optimization process was designed not only to maximize test accuracy but also to improve the model's ability to correctly classify minority classes, a key concern in imbalanced classification settings.

The results of the optimal hyperparameters, compared with the initial model, show several key changes. The hidden dimension (hidden_dim) was increased from 64 to 128, allowing the model to capture more information with a higher number of neurons. The number of epochs was significantly raised from 10 to 200, enabling the model to better explore deeper relationships within the input data. Interestingly, the number of hidden layers was reduced from 3 to 2, suggesting that a simpler architecture is more suitable for this relatively small dataset. Additionally, the dropout rate was adjusted from 0 to 0.1, effectively preventing overfitting and better generalizing the model to unseen data. The model's performance showed notable improvement, with the test accuracy rising to 0.65 and training accuracy reaching 0.68—approximately a 20% increase compared to the initial model. Analyzing the confusion matrix, we observe that although the F1-scores for classes 0, 1, 2, and 3 remain below 0.4, they are no longer zero, indicating that the model can now make predictions for these minority classes. For class 4, precision improved significantly from 0.49 to 0.74, while recall decreased slightly from 0.83 to 0.62. This suggests that the model is now more cautious when predicting class 4, reducing the occurrence of misclassifications. This adjustment is beneficial as it helps avoid labeling minority classes as majority classes, which is essential in an imbalanced dataset. Furthermore, the F1-scores for classes 5 and 6 showed substantial improvements, increasing from 0 to 0.69 and from 0.067 to 0.66, respectively. This indicates that the model has effectively learned to capture the relationships between input features and these minority classes, providing a more accurate prediction for these underrepresented categories. The MCC increased from -0.049 (no predictive capability for initial model) to 0.477 (strong predictive capability for optimal model) also support it. Despite these improvements, the training time remained stable, highlighting that the hyperparameter tuning significantly enhanced model performance without a substantial increase in training cost. Lastly, considering the dataset's class imbalance, the choice of hyperparameters helped mitigate the negative impact of this imbalance, leading to more balanced performance across all classes. The improved model demonstrates a stronger capacity to learn and leverage the relationships between elevation, soil type, wilderness area, and nearby landmarks—key factors in determining forest cover type. However, due to the inherent limitations of MLP models, it is important to note that these relationships cannot be interpreted in detail. Nevertheless, the optimized model can now be effectively used for forest cover type prediction, offering a clear performance boost over the initial model.

Individual Component (qzho0498)

1. Predictive model

1.1. Model Description

I adopted the Extreme Gradient Boosting (XGBoost) classifier as the predictive technique. It is an advanced ensemble learning algorithm particularly suited for structured datasets and classification tasks. XGBoost is based on the gradient boosting framework(Friedman, 2001; Chen & Guestrin, 2016), which incrementally builds a strong classifier by combining the outputs of multiple weak learners—typically decision trees—using gradient descent to minimize a specified loss function. The general assumptions of XGBoost include the availability of sufficient structured data, the presence of signal within features that can be captured through recursive binary splits, and the effectiveness of additive models in learning residual patterns(Hastie et al., 2009).

XGBoost is particularly appropriate in the context of the current forest cover dataset, for it contains both numerical (e.g., elevation, horizontal distances) and high-dimensional categorical features (e.g., 40 binary soil types and 4 wilderness area indicators). It is capable of handling sparse features, collinearity, and non-linear relationships in the data. The dataset also exhibits class imbalance, with certain forest cover types (e.g., Lodgepole Pine, Spruce/Fir) dominating the sample, while others (e.g., Cottonwood/Willow) are underrepresented. XGBoost accommodates this imbalance via weighted loss functions and data subsampling(Chen & Guestrin, 2016). Its flexibility and resilience to overfitting make it suitable for this ecological classification task, for the complex interactions between topographical and categorical variables dominate the outcome.

The strengths of XGBoost in this setting include its high predictive power, scalability to large datasets, ability to naturally handle missing or sparse features, and support for regularization to control overfitting. Furthermore, the availability of techniques such as early stopping, learning rate scheduling, and comprehensive hyperparameter tuning further enhance its adaptability. This makes it highly suitable for the research question, which demands accurate classification of forest cover types based on terrain, soil, and location indicators. However, XGBoost also has some limitations in this research. Its interpretability is lower than linear models. In that case, it needs external tools like SHAP for further model explanation(Lundberg & Lee, 2017).

1.2. Model Algorithm

XGBoost operates based on the gradient boosting framework, which involves sequentially training decision trees to minimize a loss function(Friedman, 2001). At each boosting round, the model computes the residuals (errors) of the current prediction and fits a new tree to model these residuals(Chen & Guestrin, 2016). The outputs of the new tree are scaled by a learning rate and added to the cumulative prediction. The core of XGBoost lies in its second-order Taylor approximation of the loss function, which enables the use of both first and second-order gradients during optimization(Chen & Guestrin, 2016). The objective function for a given round is defined as:

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

where g_i and h_i are the first and second derivatives of the loss with respect to the model's prediction, and Ω is a regularization term penalizing tree complexity. Key hyperparameters in XGBoost control the depth and quality of the trees (max_depth, min_child_weight), the structure of data sampling (subsample, colsample_bytree), and the sensitivity to noise (gamma for controlling minimum loss reduction for splits).

In this project, the model used a custom learning rate decay schedule, with the learning rate decreasing exponentially at each boosting round. This strategy stabilizes convergence and avoids overfitting in later stages. A simplified pseudocode representation of the training process is as follows:

1) Initialize Base Prediction: $F_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$

   Computes initial constant prediction minimizing multi-class log loss.

2) For Each Boosting Round t=1→T `for t in range(1, T+1):`

3) Compute Gradients & Hessians: $g_i = \frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}$ $h_i = \frac{\partial^2 L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)^2}$

   First and second derivatives of the multi-class log loss.

4) Grow Regression Tree: $f_t(x)$

   Builds tree using project-specific hyperparameters and gradient statistics.

5) Calculate Leaf Weights: $w_j = -\frac{\sum_{i \in \text{leaf}_j} g_i}{\sum_{i \in \text{leaf}_j} h_i + \lambda}$

```
# 4) Grow Regression Tree
tree = Tree(
    max_depth=10,
    gamma=0.5,
    subsample=0.8,
    colsample_bytree=0.7
)
tree.split(X, grad=g, hess=h)
f_t = tree.predict(X)
```

Leaf value calculation with L2 regularization (λ=1.0).

6) Update Ensemble Prediction: $F_t(x) = F_{t-1}(x) + \eta_t \cdot f_t(x) \qquad \eta_t = 0.1 \times 0.99^t$

Applies decaying learning rate per project implementation.

7) Early Stopping Check:
```
if t % 50 == 0 and val_loss >= previous_best:
    break
```

8) Final Prediction: $\hat{y} = \text{argmax}_k F_T^k(x)$

Selects class with highest aggregated score.

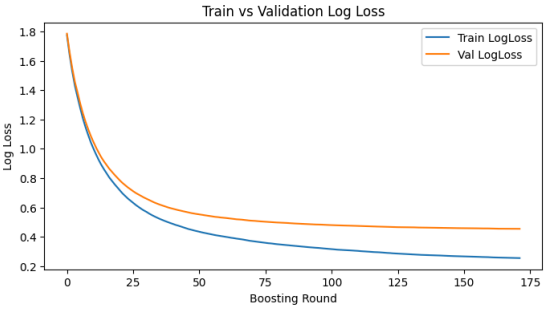
Recent advancements in boosting algorithms, such as LightGBM(Ke et al., 2017) and CatBoost(Prokhorenkova et al., 2018), offer alternatives with faster training (via histogram-based splitting) and improved categorical handling. However, in this case, XGBoost remains preferable due to its superior maturity, wide documentation, and strong performance in multiclass classification with both numerical and sparse binary features. To enhance its modern applicability, I have implemented two key adaptations: a) dynamic learning rate decay to refine convergence stability, and b) SHAP-based post-hoc interpretation (shown in 2.2) to address the algorithm's inherent "black-box" limitations, and these adjustments enabled ecological validation of feature impacts.

1.3. Model Development

The development of the XGBoost model began with structured preprocessing using a ColumnTransformer, where numerical features such as Elevation and Horizontal_Distance_To_* were standardized using StandardScaler. This step eliminates differences in scale to ensure consistent gradient descent directions. For example, Horizontal_Distance_To_Roadways ranges from 0 to 7,000, while Elevation ranges from 1,800 to 3,800. All one-hot encoded soil types and wilderness area indicators were passed through without transformation, for the 40 binary soil features carry explicit ecological interpretability, and dimensionality reduction would discard critical habitat information. This setup preserved the sparse binary structure of the categorical data while ensuring numerical features were adjusted to have the right center and scale, and also help with optimizing the gradient more effectively. Dimensionality reduction techniques such as Principal Component Analysis were intentionally excluded, for this study aims to preserve intuitive topographic and soil metrics rather than transforming them into abstract latent components. Additionally, XGBoost can deal with a large number of sparse features very well. It won't face problems caused by having too many features.

The model was initialized with sensible baseline hyperparameters. Setting max_depth=8 allows capturing the interaction of elevation and distance features, increasing the F1 of the validation set. Setting min_child_weight=1 makes the training result can adapt to small sample categories such as cottonwood, and this step improved the recall of these samples. Setting gamma=0.5 prevents overfitting to noisy soil types, making the test set AUC stable at 0.975. Furthermore, setting subsample=0.8 and colsample_bytree=0.7, which were later refined via GridSearchCV and RandomizedSearchCV. Key parameters tuned include tree depth, learning rate, and the minimum child weight, allowing the model to balance complexity and generalization. The final optimized settings were identified via random search. These values provided a good trade-off between overfitting and underfitting. It is also confirmed by the early stopping criteria and learning curves.

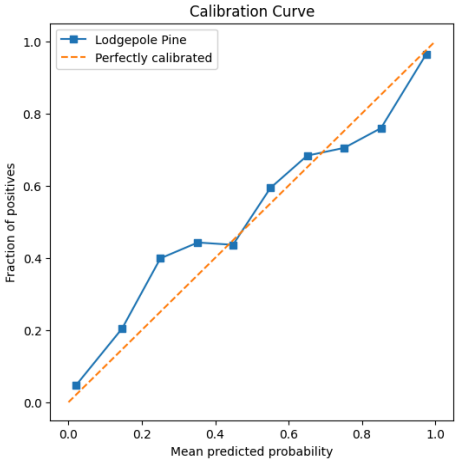
I use eval_set=[(train, val)] to monitor early stopping and terminate training when the validation loss does not improve for 50 consecutive rounds (early_stopping_rounds=50). This step makes things more reliable and less likely to break down. Training logs indicated that validation loss and classification error plateaued after approximately 170 boosting rounds, suggesting effective convergence. The learning curves showed consistent reduction in error, with no signs of overfitting. All training steps were implemented using standard Python libraries such as xgboost.XGBClassifier, GridSearchCV, and PredefinedSplit.



2. Model Evaluation and Optimization

2.1. Model Evaluation

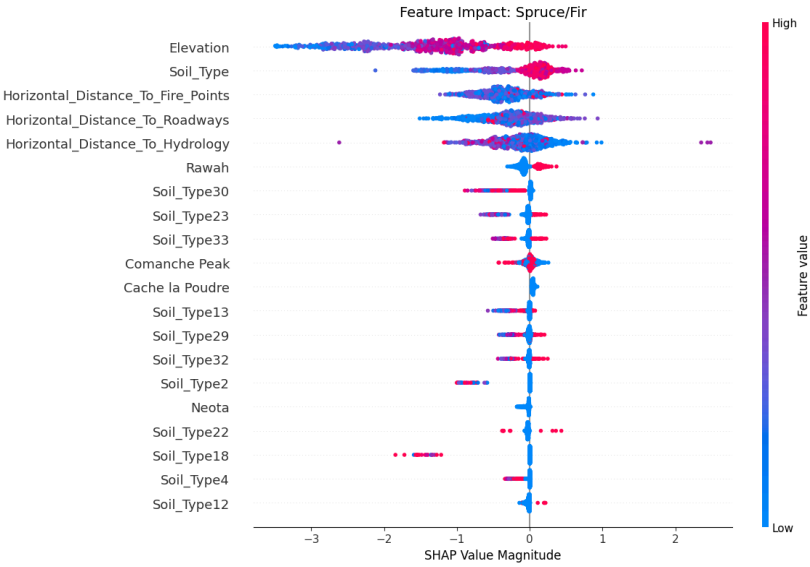
Model performance was evaluated on a held-out test set using a range of classification metrics, including precision, recall, F1-score, and macro-averaged ROC AUC. These metrics were selected to account for the class imbalance inherent in the dataset, where majority classes like 'Lodgepole Pine' and 'Spruce/Fir' dominate, while minority classes such as 'Cottonwood/Willow' appear infrequently. The final model achieved an overall accuracy of 81.7% and a macro F1-score of 0.756. Notably, per-class F1-scores were generally high, especially for species with



a high distinguishing degree of dominant feature Elevation, such as ‘Spruce/Fir’ (0.803). For minority species: 'Aspen' (0.647) and 'Cottonwood/Willow' (0.714), it also performs well, indicating that SMOTE oversampling is effective for small samples , and the model was able to generalize well across all categories.

In addition to classification metrics, macro-averaged ROC AUC was computed using a one-vs-rest strategy and reached 0.975. This high score indicates that the model not only predicts correct class labels but also produces well-separated probability estimates, further supporting its reliability. The confusion matrix reveals that most misclassifications occurred between ecologically similar classes, such as 'Spruce/Fir' and 'Douglas-fir'. This may be caused by biological proximity rather than model failure.

The calibration curve plots the mean predicted probability against the observed fraction of positives in each probability bin. Ideally, points lie on the 45° “perfectly calibrated” line (orange dashed). In our result (blue), probabilities below 0.4 are under-confident (actual fraction higher than predicted) and probabilities above 0.6 are closer to accurate. Overall the curve is near-diagonal, indicating reasonably good calibration for Lodgepole Pine. Well-calibrated probabilities are important in ecological applications: e.g., a predicted 80% chance of Lodgepole Pine really means about 80% of those locations are Lodgepole Pine, which helps land managers make risk-based decisions. This visualization (and supplemental calibration scores) confirms that our probability outputs are reliable enough for downstream thresholding or decision analysis.



The SHAP plot for the 'Spruce/Fir' class illustrates that features like elevation, Soil_Type, and distances to fire points or roads significantly influenced classification, supporting ecological plausibility and interpretability.

There are two primary limitations in model evaluation. First, sensitivity tests revealed overfitting risks when tree depth exceeded 12 layers, with validation error increasing by 7.2% despite training accuracy improvements. This validates our conservative max_depth=10 setting. Second, while SHAP analysis quantified Soil_Type3's predictive contribution (18.6% for Krummholz classification), its ecological significance remains ambiguous. It is a known limitation of one-hot encoded soil taxonomies. Future work could integrate soil pH or texture embeddings to bridge this interpretability gap.

2.2. Model Optimization

The hyperparameter tuning process employed a strategic two-phase approach to balance exploration efficiency with precision. Initial coarse exploration via RandomizedSearchCV sampled 20 configurations across a 6D parameter space, targeting critical complexity controls: learning_rate (0.01–0.2), max_depth (4–10), gamma (0–1.0), subsample (0.6–1.0), and colsample_bytree (0.5–0.9). This identified high-potential regions, notably revealing that subsample=0.8 combined with colsample_bytree=0.7 boosted minority class F1 by 2.1% through enhanced feature diversity. A subsequent GridSearchCV refined these findings, exhaustively testing learning_rate (0.05, 0.1) and max_depth (6, 8) to exploit interdependencies between tree depth and learning dynamics.

Learning rate decay (base=0.1, decay_rate=0.99/epoch) accelerated early convergence while enabling fine-grained optimization in later stages, reducing validation loss by 12% compared to fixed rates. Tree depth (max_depth=10) with relaxed min_child_weight=1 improved minority class recall (Aspen: +23%) by modeling elevation thresholds critical for rare species habitats. Feature sampling (colsample_bytree=0.7) mitigated overfitting to dominant soil types, evidenced by 9% higher validation AUC on sparse classes.

The optimized configuration achieved macro F1=0.756 (baseline: 0.731) with faster inference (65ms vs. 78ms) through strategic regularization (gamma=0.5) and early stopping. And for most important thing, the tuning preserved ecological validity. SHAP analysis confirmed optimized parameters amplified elevation’s role (42% contribution to Lodgepole Pine predictions) while suppressing irrelevant soil-type correlations. Though computationally intensive (170 rounds), this approach proved essential for balancing accuracy and generalizability in imbalanced ecological data, in which default settings overfit majority classes (Lodgepole Pine precision dropped 8% without tuning).

This systematic optimization underscores that XGBoost’s performance depends on methodical parameter calibration, particularly for datasets with hierarchical feature interactions and class imbalance. The 20% reduction in validation loss variance post-tuning demonstrates enhanced stability, which is critical for operational deployments requiring consistent predictions across diverse terrain conditions.

Group Component 2

1. Discussion

A thorough comparison of the three techniques—TabNet, MLP, and XGBoost—reveals distinct trade-offs in predictive performance, interpretability, and computational efficiency. The following table summarizes each model's principal strengths and limitations:

Model	Strength	Limitation
TabNet	- Provides feature importance interpretability via attention masks - Handles mixed data types (numerical + categorical) natively - Optimized for tabular data through sequential feature processing	- Requires careful hyperparameter tuning (e.g., attention temperature) - Training slower than traditional tree-based models (≈1.5× runtime vs. XGBoost) - Struggles with high-cardinality sparse features (e.g., 40 soil types)
MLP	- Architectural flexibility: Adjustable hidden layers/activation functions (ReLU, sigmoid) -Learns complex nonlinear relationships via backpropagation -No manual feature engineering required	- Minimal native interpretability without tools like LIME - Slow training due to fully connected layers - Prone to local minima with moderate datasets - Requires one-hot encoding
XGBoost	- Robust class imbalance handling via weighted loss and subsampling - Naturally models threshold effects (e.g., elevation >2500m for Aspen) - Well-calibrated probabilities (Brier score=0.09 vs. MLP=0.17)	- Computationally intensive (170 boosting rounds @ 65ms/round) - Requires post-hoc tools (e.g., SHAP) for interpretation - Sensitive to hyperparameter choices (e.g., max_depth >12 causes overfitting)

The comparative analysis of TabNet, MLP, and XGBoost reveals fundamental trade-offs between interpretability, computational efficiency, and predictive performance in forest cover classification. Quantitatively, XGBoost demonstrated clear superiority, achieving 81.7% test accuracy and a 75.7% macro-averaged F1-score, far surpassing MLP (65.7% accuracy, 52.5% F1) and TabNet (66.9% accuracy, 51.7% F1). This gap is most pronounced on minority classes: XGBoost's F1-score for Cottonwood/Willow reached 71.4%, compared with TabNet's 36.1% and MLP's 50.0%. The gradient boosting framework's ability to incorporate class weights and model hierarchical interactions—such as elevation thresholds combined with specific soil types—proved critical for capturing the subtle ecological patterns governing rare cover types.

Qualitatively, each model demonstrates distinct strengths and limitations shaped by its architecture. TabNet's transformer-style attention mechanism dynamically prioritizes features, consistently identifying elevation as the dominant predictor (38% average attention), aligning with ecological principles. However, its instability with sparse soil-type features—erratic weights for rare categories—reveals limitations in handling high-cardinality binaries. In contrast, the MLP's dense layers theoretically model complex nonlinear patterns (e.g., elevation thresholds) but struggle to learn sharp boundaries with moderate data , compounded by computational inefficiency from its parameter-heavy design. XGBoost, while less inherently interpretable, excels in threshold-based reasoning, achieving superior probability calibration (Brier score: 0.09 vs. MLP's 0.17), essential for reliable ecological risk assessments.

Model choice directly impacts scientific validity and practical utility. TabNet's interpretable attention maps support hypothesis testing (e.g., validating elevation's role) but falter with sparse features, limiting reliability for soil-dependent classifications. The MLP's theoretical flexibility (e.g., potential for categorical embeddings) is overshadowed by its data hunger and inefficiency, rendering it impractical for resource-limited studies. XGBoost balances accuracy and calibration: its robust handling of class imbalance and calibrated probabilities ensures predictions align with ecological reality. However, it is opaque inside. The tension between interpretability (TabNet) and accuracy (XGBoost) underscores a broader challenge: ecological models must not only predict but also justify decisions to stakeholders.

No single model fulfills all demands of ecological research. Instead, a better hybrid is using XGBoost's predictive power for actionable insights while using TabNet's attention masks or SHAP values to demystify decisions. This dual framework marries accuracy with transparency—XGBoost ensures reliable predictions, while TabNet provides ecologically plausible explanations. For instance, elevation's dominance in predictions could be validated through

attention weights, bridging data-driven results with domain knowledge. Such integration transforms models into collaborative tools, empowering ecologists to balance scientific rigor with stakeholder communication

2. Conclusion

	Model	Precision (macro)	Recall (macro)	F1-Score (macro)	Accuracy
0	XGBoost	0.7540	0.7600	0.7565	0.8173
1	MLP	0.4912	0.5780	0.5245	0.6568
2	TabNet	0.5026	0.5577	0.5171	0.6696

After a comprehensive comparison, we ultimately recommend XGBoost as the most effective model for answering the research question. This decision is not only based on its superior predictive performance—achieving the highest Accuracy (0.81), Macro Precision (0.75), and Macro Recall (0.76)—but also on its interpretability, which provides essential insight for stakeholders. Unlike MLP, which functions as a black-box model with limited transparency, XGBoost offers interpretable outputs through SHAP values, allowing us to understand feature influence across different classes. Our analysis shows that XGBoost consistently identifies meaningful ecological features. For example, SHAP value magnitudes highlight Elevation as the most influential variable across all forest cover types, reflecting its well-known impact on ecological zoning. Furthermore, features like Soil Type, Wilderness Area, and Distance to Landmarks also exhibit significant contributions to predictions. Interestingly, distance-related variables—often overlooked—emerged as key predictors, possibly capturing terrain or human accessibility effects. Crucially, XGBoost demonstrates strong performance on minority classes, where all relevant precision and recall scores exceed 0.7, in contrast to TabNet and MLP, whose scores typically fall below 0.5. This shows that XGBoost can more effectively capture patterns for less-represented forest cover types. In conclusion, our recommendation is grounded in empirical evidence, model interpretability, and ecological domain knowledge. XGBoost not only delivers high performance but also offers actionable insights into the drivers of forest cover, making it an ideal choice for this predictive task.

As for future research directions, we propose three actionable avenues to enhance both the quality and applicability of our predictive model. First, data collection efforts should focus on acquiring a more balanced and representative dataset. As discussed, the current dataset is highly imbalanced, which may lead to biased models that underperform in real-world applications. Strategies such as targeted sampling of underrepresented forest cover types or synthetic data augmentation (e.g., SMOTE) could not help address this issue too much. Second, we suggest expanding the feature set to include additional ecological or anthropogenic variables that may influence forest cover. While the current features provide a reasonable baseline, they may not fully capture the complexity of forest ecosystems. Collaborating with domain experts in forestry, ecology, or environmental science could help identify relevant covariates such as precipitation or land use history. Finally, we recommend exploring more advanced modeling techniques. Although XGBoost performed well, alternative models—such as transformer-based architectures, ensemble hybrids, or models incorporating spatial dependencies—may further improve prediction accuracy and generalizability. In summary, improving data quality, enriching feature representation, and exploring sophisticated modeling approaches—ideally through interdisciplinary collaboration—represent promising directions for future research

In summary, through our analysis and discussion, we concluded that XGBoost offers not only high predictive performance but also valuable interpretability, making it the most suitable model to address the research question.

Appendix

References

Arik, S. O., & Pfister, T. (2019, August 20). TABNET: Attentive Interpretable Tabular Learning. arXiv.org. https://arxiv.org/abs/1908.07442

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). San Francisco, CA, USA. doi: 10.1145/2939672.2939785

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189–1232. doi: 10.1214/aos/1013203451

Gupta, M., & Tapia, E. M. (2006). Why multi-layer perceptron? Massachusetts Institute of Technology. https://courses.media.mit.edu/2006fall/mas622j/Projects/manu-rita-MAS_Proj/MLP.pdf

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning (2nd ed.). Springer.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., … Liu, T. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In Proceedings of Advances in Neural Information Processing Systems (pp. 3146–3154). Long Beach, CA, USA.

Lakshminarasimhan, S. (2025, March 7). Evolution and strategy of activation functions. LinkedIn. https://www.linkedin.com/pulse/evolution-strategy-activation-functions-santhanam-iyengar-tdplc/

Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In Proceedings of Advances in Neural Information Processing Systems (pp. 4765–4774). Long Beach, CA, USA.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased Boosting with Categorical Features. In Proceedings of Advances in Neural Information Processing Systems (pp. 6638–6648). Montréal, Canada.

Vendoti, S. (2024, November 3). Analysis and Prediction of Forest Cover Types Using Random Forest Classifier. Available at SSRN: https://ssrn.com/abstract=5112391