Forest Cover Type Prediction: A Comparison of kNN and Classification Tree Approaches

DT Nicolay 26296918
Computer Science Division
Stellenbosch University
Stellenbosch, South Africa
26296918@sun.ac.za

***Abstract*—The k-nearest neighbours (kNN) and classification trees algorithms are both useful classifiers which often demonstrate exceptional performance whilst maintaining interpretability. In this report, we apply both algorithms to predict the forest cover type using a cartographic variable dataset. The dataset presents several data quality challenges including missing values, feature correlation, class imbalance, mixed data types, and outliers, requiring algorithm-specific preprocessing approaches. For kNN, we implemented pre-processing techniques such as imputation, correlation removal, and SMOTE-based class balancing to address distance-based learning requirements. Classification trees on the other hand required fewer pre-processing steps due to certain characteristics providing robustness to data quality issues. An advanced cross-validation approach was implemented for both kNN and classification trees to ensure techniques such as scaling and SMOTE were only applied to training folds. This prevented data leakage and provided unbiased performance estimates. Control parameter tuning evaluated 36 parameter combinations for kNN and 384 combinations for classification trees. Impressive results were recorded using 10-fold cross-validation, with kNN achieving an accuracy of 91.84% and classification trees achieving an accuracy of 92.76% on average across the folds.**

***Index Terms*—k-nearest neighbours, classification trees, forest cover classification, class imbalance, data preprocessing, SMOTE, cross-validation, feature scaling**

I. Introduction

Since the nearest neighbour and classification tree algorithms rely on different theory and assumptions, we begin by explaining the theory behind each model. The forest cover dataset presents nine data quality issues that all need to be addressed by appropriate handling techniques. Due to the difference in the algorithms, it is important to base the handling techniques on the theory and nature of each model. After applying the appropriate data pre-processing for each algorithm, justifying each action carefully, we analyse the various control parameters involved in the algorithms. A custom grid search cross-validation technique is employed to ensure that the best-performing parameter combinations are obtained. This grid search technique ensures no parameter combination goes unmeasured. The effect of certain parameters is also analysed carefully and compared between algorithms to better understand the models.

After the best parameters for each model are obtained, we confirm their performance using 10-fold cross-validation over the entire dataset recording numerous key metrics. The learning curves of the models are considered as an increasingly larger subsets of the data are taken to fit each model and the results plotted. We analyse the vast differences in computational and algorithmic complexity between the two algorithms, especially the difference in the number of parameter combinations to check. The Mann-Whitney U test [1] is employed to perform a pairwise comparison between the models. Finally, the feature importance of the classification trees algorithm is reported on to interpret which features the model found to have discriminatory power.

Our goals are to develop the classification models for the provided dataset, to identify data quality issues that need to be addressed, to identify and implement the necessary data transformations for the classification models, and to compare the performance of the implemented models. We achieve these goals by starting with a thorough theoretical understanding of the models, thereafter making informed decisions on how to go about pre-processing with the appropriate theoretical context. Extensive effort is taken to ensure no data leakage and robust generalisation results during both the parameter tuning and final evaluations.

The code for this assignment is available on GitHub [2].

II. Background

The goal of this section is to explain the background of the k-nearest neighbours and classification trees algorithms. Thereafter, we discuss the expectations with respect to the data quality issues.

A. k-nearest Neighbours (kNN)

The nearest neighbour algorithm is an example of instance-based learning where the prediction for an instance is made by comparing the instance to similar training instances. This is a lazy learning approach since we delay processing training data until prediction is needed [3]. Since kNN is also similarity-based, it is non-parametric, which means that it does not learn a mapping from the input space to the output space.

The k-nearest neighbour algorithm is presented in Algorithm 1 which explains that we calculate the distance from each data point in our training dataset to the distance. From there, we choose the closest k observations and perform inference. In a classification context majority voting may be

used to determine the class, in a regression context the mean or median of the *k* nearest observations is used.

In order to use the algorithm we need to define a value of *k* which decides the number of nearest neighbour instances to consider when predicting. Choosing a low value for *k* results in a high variance, but low bias leading to an unstable model which tends to overfit. Choosing a large value for *k* results in a smaller variance, but high bias, leading to a more stable model which tends to underfit. Measures such as k-fold cross-validation assist in the selection of a value of *k*.

Algorithm 1 k-Nearest Neighbors Algorithm

function KNN(D, x, k)
  for all $x' \in D$ **do**
    $d =$ distance(x, x')
  end for
  sort(d)
  $S =$ set of k patterns in D closest to x
  return class as majority class in S
end function

Notation: D denotes the training dataset, x is the query instance, k is the number of nearest neighbours, and distance$(\cdot)$ is a distance metric (e.g., Euclidean distance).

Another important part in the kNN algorithm is the distance metric. This determines how similarity between observations is measured. Euclidean distance is the most common metric. It calculates the line distance between two points in space, this is useful for continuous numerical features. Manhattan distance computes the sum of absolute differences across each dimension. It is often used when features are not strongly correlated. A generalisation of these two is the Minkowski distance, which introduces a parameter p that allows interpolation between different norms: when $p = 1$, it reduces to Manhattan distance, and when $p = 2$, it becomes Euclidean distance. The choice of metric can significantly impact the performance of kNN, especially in high-dimensional spaces or when feature scales differ.

B. Classification Trees

Classification trees represent a fundamental approach in supervised machine learning for predictive modelling, where the learned model forms a hierarchical tree structure with non-terminal nodes representing decisions on descriptive features and terminal leaf nodes representing target feature predictions [4]. A classification tree is a type of decision tree where leaf nodes represent different discrete classes.

Classification trees employ recursive partitioning where a training dataset is systematically divided into increasingly homogeneous subsets based on feature values. It uses a greedy strategy to select the feature to partition upon at each split that maximises information gain [3]. The information gain criterion, derived from Shannon's entropy measure, quantifies the reduction in uncertainty achieved by partitioning the data according to a specific feature test. Formally, given a dataset D with M different classes, the entropy is calculated as:

$$H(D) = -\sum_{m=1}^{M} p(y_m) \log_M p(y_m)$$

where $p(y_m)$ represents the probability of class y_m occurring in D. When the dataset is partitioned on feature α into O outcomes, the information gain is computed as:

$$\text{gain}(\alpha) = H(D) - H_\alpha(D)$$

where

$$H_\alpha(D) = \sum_{o=1}^{O} p_o \, H(D_o)$$

represents the weighted entropy after the split.

Since we recursively partition until either a stopping condition is satisfied or all subsets are homogeneous, classification trees inherently lead to overfitting, as the algorithm attempts to perfectly classify all training instances. This often creates overly complex trees that capture noise rather than underlying patterns [4]. To remedy this, post-pruning techniques are used to remove branches that do not improve generalisation performance.

Classification trees can naturally handle both categorical and numerical features. Classification tree induction algorithms can cope with missing values. Unfortunately, classification trees exhibit an axis-aligned bias which restricts decision boundaries to be parallel to feature axes.

C. Expectations With Respect to Data Quality Issues

Since kNN relies heavily on distance metrics, it will be severely impacted by missing values since it cannot compute meaningful distances between points. This can however be handled by ignoring the missing values in the distance calculation and scaling up the weight of the non-missing descriptive features. The kNN algorithm can actually be used as an imputer itself as well. If nothing is done the algorithm will either fail or give biased distances. Classification trees on the other hand handle missing values more gracefully. Classification trees can adjust the gain ratio calculation

$$\text{gainRatio}(x) = (1 - F) \times \big(H(D) - H_x(D)\big)$$

where F is the fraction of missing patterns.

With regards to the `Facet` feature being correlated with the Aspect feature, kNN will be moderately impacted since the distance calculation used will overweight certain areas of similarity. There is no built-in mechanism to handle feature correlation as a data issue. Classification trees will be minimally impacted since trees naturally handle correlated features through the feature selection process involving the gain ratio/information gain. This means that the classification tree induction will likely select one feature and ignore the correlated redundant one.

Concerning the Inclination feature only containing noisy values, the impact on the kNN algorithm will depend on the value of k since higher values are more sensitive to noise

and lower values are more robust to noise. However, since there are many predictive features the value won't dominate the distance calculation. Classification trees are robust to noise and will handle it through pruning. The noisy patterns will end up in small leaf nodes as the model overfits and post-pruning removes these leaves. The noise becomes a minority and does not affect majority class prediction.

Since kNN employs distance measures which are often sensitive to outliers since the distance can be seriously skewed even if only one feature is outlying. For small values of k outliers are unlikely to be nearest neighbours and will therefore have limited influence. For large values of k the outliers will also have limited influence due to majority voting. Cumulatively, outliers will have a moderate impact on the kNN algorithm. Classification trees are robust to outliers since they handle them in a similar way to noise. Outliers are first isolated in small leaf nodes during induction. Pruning removes leaves which contain outliers which results in outliers becoming a minority in combined larger subsets. This makes decision trees robust to outliers.

The kNN algorithm will be severely impacted by features with numeric ranges that differ significantly from one another since features with large ranges dominate distance calculations. For example, if one feature takes on the value of 0 or 1 and another takes on the value of 0 or 100 000, the second will overwhelm the distance calculation. Without normalisation, the kNN algorithm will be unreliable. Classification trees have a natural robustness to different feature scales and therefore will be minimally impacted. This stems from trees using individual feature values for splitting thresholds.

The mixed data types will affect the kNN algorithm since it requires special handling for mixed types such as using the Gower's similarity with different similarity measures for different types. However, in this dataset since there are two categorical variables with the first encoded over four columns and the second over 40, and 13 numeric features there can be a distance bias toward the numeric features. Euclidean distance assumes features are continuous, but the categorical features are encoded as zero or one. Classification trees naturally handle mixed data types since they directly compare categorical features against each other for splitting, and the splitting is simply decided off information gain.

The impact of the `Water_Level` feature having a cardinality of one will be none at all for kNN. It will only contribute the the distance calculation unnecessarily and be a computational waste. Since all observations have the same value, all the distance measures will be equally skewed, resulting in no impact at all. The classification tree will never select this feature during tree induction since there is zero information gain because all values are identical. Therefore, classification trees naturally handle this issue and it will have no impact.

The `Observation_ID` unique feature will have a sever impact on kNN since each instance appears maximally different from all others on this feature. The problem is that this dominates distance calculations making all instances seem dissimilar. It breaks the fundamental assumption of similarity-based learning. Classification trees will overfit severely since each unique value would create a separate branch. The information gain maximisation favours features with many outcomes so the result will be an extremely busy tree with poor generalisation.

The skewed class distribution impact on kNN depends on the value of k, since a smaller k is more robust to class imbalance whilst a large k results in the majority class dominating the predictions. Methods such as the Synthetic Minority Over-sampling Technique (SMOTE) must be considered. The impact of classification trees is moderate since they are sensitive to skewed distributions. The minority class instances result in small leaves which are likely to be pruned, then the combined instances will be classified as a majority class. This results in poor minority class recognition.

This section discussed the theoretical foundations of the two core algorithms. It also described the expected effect of the data quality issues on the algorithms.

III. Methodology

Tools, libraries and various implementation approaches are discussed in this section.

A. Tools and Libraries

This implementation was created using Python. NumPy [5] was used for numerical computations. Pandas [6] was used for data manipulation and analysis. Matplotlib [7] was used for data visualisation and plotting. Scikit-learn [8] was used extensively for pre-processing, model selection, validation, both machine learning algorithms, and the evaluation metrics. The imbalanced-learn library [9] was used for the SMOTE and Tomek link undersampling techniques. For statistical testing, SciPy [10] was used. All the random seeds were set to 42 when using these libraries for reproducibility.

B. Implementation

A core implementation challenge involved developing custom cross-validation implementation to ensure proper resampling and that the SMOTE and Tomek techniques were only applied within each *training* fold. This ensures the validation was unbiased since it was performed on the original unbalanced data and also prevents data leakage. The standard scikit-learn cross-validation does not accommodate for the requirement. These SMOTE and Tomek techniques were implemented using the imbalanced-learn library.

Algorithm-specific preprocessing pipelines were developed to address the distinct theoretical requirements and assumptions of each algorithm. For kNN, a comprehensive preprocessing workflow included scalar normalisation to ensure all numerical features were scaled between 0 and 1, preventing features with larger ranges from dominating distance calculations. kNN imputation with $k = 5$ was implemented to handle missing values in the `Slope` feature, while correlated features like `Facet` were removed to prevent double-weighting in

distance computations. The noisy `Inclination` feature was eliminated, and categorical variables were converted to binary encoding to maintain compatibility with distance-based calculations. In contrast, the classification tree pipeline leveraged the inherent robustness by maintaining missing values, mixed data types, and correlated features in their original form, allowing the tree induction algorithm to handle these challenges naturally through its splitting criteria and pruning mechanisms.

The scikit-learn library's implementation was the core of both algorithms, with `KNeighborsClassifier` configured using the optimal parameters determined through grid search, while `DecisionTreeClassifier` was implemented with the best-performing control parameters. The grid search implementations utilised `StratifiedKFold` for cross-validation to maintain class distribution proportions across folds. Performance evaluation was conducted using scikit-learn's comprehensive metrics suite including `accuracy_score`, `precision_recall_fscore_support`, `cohen_kappa_score`, and `matthews_corrcoef`, with `roc_auc_score` implemented using the one-vs-rest approach for multiclass classification. Learning curves were generated using the `learning_curve` functions to analyse model behaviour across different parameter values and training set sizes.

This section explained the libraries used and how they were made useful in the various parts of the models.

IV. Empirical Procedure

The data-preprocessing techniques used to tackle the data quality issues for each algorithm are discussed in this section. Thereafter, we explain the rigorous control parameter tuning process. Finally, we cover how the performance metrics were collected and the analysis procedure.

A. Data Pre-processing

In this section we discuss the data pre-processing steps applied to the dataset prior to implementing the k-nearest neighbour and classification tree algorithms.

1) k-Nearest Neighbour Data Pre-processing: First, the unique identifier feature `Observation_ID` was removed because unique values make all instances appear maximally different. This breaks the similarity-based learning assumptions and leads to the feature dominating distance calculations. The feature has no signal that can provide the model with any benefit.

Second, we removed the constant feature `Water_Level` since it provides zero discriminative information but adds computational overhead to all the distance calculations. We can be certain that the model performance will not change based on the fact that the feature has a cardinality of one.

Third, we normalised the numeric features by applying linear scaling to ensure the numeric features are in the same range being between zero and one. Features with large ranges will simply dominate distance calculations over smaller range features. This would make the model unreliable and heavily weight features with higher values. Specifically, `Elevation`, `Aspect`, `Slope`, `Horizontal_Distance_To_Hydrology`, `Vertical_Distance_To_Hydrology`, `Horizontal_Distance_To_Roadways`, `Hillshade_9am`, `Hillshade_Noon`, `Hillshade_3pm` are all normalised. The scaler was fit to the training set only in all situations (including the cross fold validation) and applied to the training and test set to ensure no data leakage.

Fourth, we handled the missing values by using kNN imputation with a k of five. There are only 298 observations that are missing the `Slope` feature. For each of these we computed the Euclidean distance between the observation and all others based on available features, then identified the k closest rows (neighbours) that have the missing feature value. The mean of these values were calculated and imputed to the missing observation.

Fifth, we addressed the feature correlation issue by removing the `Facet` feature since it is just providing redundant information. Correlated features can overweight certain aspects in distance calculations resulting in worse model performance. Essentially, the `Aspect` feature would have double the influence on the model as the other features if `Facet` was not removed.

Sixth, the mixed data types issue was handled by converting the one categorical variable `Soil_Type1` to numeric. This was achieved by simply replacing all the instances with a *positive* value with the number one, and the *negative* instances with zero. This approach was chosen since it aligned the feature with the other soil type features which all take on the values of either one or zero.

Seventh, the features containing outliers were left untouched. The kNN algorithm has moderate sensitivity to feature-based outliers, but a small k value provides natural protection.

Eighth, SMOTE was used to improve the class imbalance. Since the five minority classes had significantly fewer observations than the majority two classes, we employed SMOTE to increase the number of minority class observations to 20% of the majority class. This partial balancing approach was used since the dataset only increased in size by 34%, as opposed to the dataset exploding in size if they were all were made equal. For example, if we over-sample class 4 to 283,301 we'd create 280 000 synthetic samples for one class. This is massive and risky for overfitting. On top of this Tomek links for majority class under-sampling was also used to reduce the number of observations in the majority classes. Using this under-sampling technique alone would not be sufficient since only a small fraction of the majority samples are removed. The Tomek link technique resulted in the majority classes reducing by roughly 60 000 observations in each majority class. That is, Class 1 now has 158,427 samples, and Class 2 now has 225,918 samples after reduction. It is important to note that SMOTE and Tomek undersampling were only applied to the *training*

datasets and never to the *test* datasets.

Ninth, the noisy feature `Inclination` was removed since it was confirmed to contain only noise. Noisy features affect distance calculations, although larger k values provide some robustness it is safer to remove it since we know it contains no signal.

2) Classification Trees Data Pre-processing: First, the unique identifier feature `Observation_ID` was removed because it creates severe overfitting. Classification tree induction will result in one branch per unique value in this feature, this triggers the many-values bias in information gain maximisation. Again, since the feature provides no predictive power we can safely remove it without losing anything important.

Second, the constant `Water_Level` feature was removed since it will always result in zero information gain if the classification tree were to split on it. The algorithm will naturally ignore it, but it does waste computational resources.

Third, the target class imbalance was addressed using SMOTE to over-sample the minority classes. Again, the minority classes were oversampled to 20% of the size of the largest majority class. This is done because trees are sensitive to skewed distributions since minority classes get pruned away leading to poor recognition. To use SMOTE, the missing values needed to be temporarily imputed using kNN imputation and the categorical variables encoded. This was performed to enable SMOTE to work as intended. The classification tree was still trained on the dataset containing missing values and categorical features. It is important to note that SMOTE and Tomek undersampling were only applied to the *training* datasets and never to the *test* datasets.

Fourth, the missing values were left as is since the classification trees have excellent built-in missing value handling. Pre-processing is therefore unnecessary in this case. The tree induction algorithm distributes the instance fractionally across all branches where a feature is missing, this is done proportional to the observed frequencies of those branches among the non-missing data.

Fifth, the correlated `Facet` and `Aspect` features are left as is. Trees naturally handle correlation through information gain and redundant features are automatically ignored. The algorithm will select the most informative feature.

Sixth, the mixed data types are left as is. Classification trees natively handle both categorical and numeric features without the need for conversion.

Seventh, the numerical feature scale differences need not be attended to. Trees are scale-invariant since they use a threshold-based split that works regardless of the feature ranges. The features are only compared on an information gain basis and not by their values. This is unlike the kNN algorithm which compares the values of the features during the distance measurements.

Eighth, the outliers are left in the dataset. Classification trees are naturally robust to outliers through the pruning process. If a branch only exists because of a few outliers, that branch has few training examples. Therefore, its estimated error rate is high.

Ninth, the noisy feature `Inclination` is kept since the pruning process handles noise. Trees handle noise well by isolating noisy patterns in small leaves that get pruned.

B. Control Parameter Tuning

The custom cross-validation approach used to tune the parameters for each of the algorithms is discussed in this section. The key principle is that the SMOTE/Tomek techniques are only applied to training folds, never to validation folds, ensuring realistic performance estimates.

1) k-Nearest Neighbour Control Parameter Tuning: The three primary parameters were optimised over during a 5-fold stratified cross-validation on a 20% subset for computational efficiency. First, is the value of k which controls the number of nearest neighbours to consider. Values of [3, 5, 7, 9, 11, 15] were considered. Second, both the options for the weights parameter were considered. Specifically, uniform where all neighbours are weighted equally and distance weighting where closer neighbours have more influence. Third, three distance metrics were considered: Euclidean distance, Manhattan distance, and Minkowski distance. Since there are three parameters with six, two, and three options respectively, there are $6 \times 2 \times 3 = 36$ parameter combinations to test in the cross-validation.

2) Classification Trees Control Parameter Tuning: The five primary parameters were optimised over during a three-fold stratified cross-validation on a 10% subset for computational efficiency. First, four maximum depth parameter values were considered: [10, 15, 20, 25]. This parameter controls tree depth to balance complexity against generalisation. Second, the minimum number of samples required to split an internal node. The values of [2, 5, 10, 20] were considered. Higher values prevent overfitting by requiring more evidence for splits. Third, the minimum number of samples required at leaf nodes was tuned, with values of [1, 2, 5, 10]. This prevents the formation of overly specific rules that do not generalise well. Fourth, the splitting criterion was evaluated using both *gini* (Gini impurity) and *entropy* (information gain). Gini is computationally faster, while entropy provides a more theoretically principled approach. Finally, the *max_features* parameter was tuned over ['sqrt', 'log2', None], which controls the number of features considered for the best split. Using *sqrt* or *log2* introduces randomness to improve generalisation, while None considers all features. Since there are five parameters with four, four, four, two, and three options respectively, there are $4 \times 4 \times 4 \times 2 \times 3 = 384$ parameter combinations tested.

C. Performance Metrics

Four primary metrics were considered to evaluate the performance of the models. Accuracy was used to evaluate the overall correctness of predictions across all classes. Precision was used to evaluate the ability to avoid false positives. Recall was used to test the ability to find all positive cases. F1-Score was also employed as a measure of the overall performance of the algorithms. Cohen's Kappa which is a statistic that measures the level of agreement between two classifiers on

categorical data and adjusts for the agreement could happen by chance. This metric was employed to compare the predictions of the classifier to the true values of the test set. Similarly, the Matthews Correlation Coefficient (MCC) was employed as a comprehensive performance metric that extends beyond binary classification to evaluate the quality of multiclass predictions, providing a balanced assessment that accounts for the correlation between predicted and actual classifications across all classes while being less sensitive to class imbalance than traditional accuracy measures.

10-fold cross-validation mean accuracy was employed to provide robust performance estimation. The standard deviation of the cross-validation score was used to measure the algorithm stability across the various folds. The range of the cross-validation scores was also evaluated to show the best and worst fold performance.

Computational metrics such as training time, prediction time, and model complexity (e.g., parameter count, tree depth, memory usage) were also recorded. Weighted averages were applied to handle class imbalance, and multiple metrics were used to avoid over-reliance on a single measure.

D. Analysis Process

The learning curves of both the classification trees and kNN algorithms were compared by sampling increasingly larger training subsets of the data. These training subsets are then used to train the models, obviously in the case of kNN the training data is the data it is *fit* to (since kNN does not train). The accuracy performance was then plotted for both the training and test sets. Each iteration was performed with three-fold cross-validation.

While all the tests were run, the number of iterations and running time was recorded. This assisted in the analysis of the computational complexity of the algorithms.

The Mann-Whitney U test [1] was used to perform a pairwise comparison (1×1) between the two models' cross-validation results. This is a non-parametric test and assumes the data (accuracies in this case) is non-parametric, and that the runs of the models are independent. The test can be performed on small samples (in this case 10 per group) making it useful despite having lower power.

For the classification trees, the library used has a built-in feature importance attribute which measures the contribution of each feature to the reduction of node impurity. This was a valuable tool for understanding which features had the greatest discriminative power in the classification task and provided insights into the underlying data structure.

All the important metrics and model comparison was performed using multiple runs using cross-validation. This results in the metrics being more robust and a better indication of the performance on true unseen data.

V. Research Results

The parameter tuning and overall research results are all discussed in this section. We also investigate the effects of various components through experimentation.

A. Parameter Tuning Results

1) K-nearest Neighbours: The custom five-fold cross-validation grid search returned the best kNN parameters with $k = 3$, the Manhattan distance metric and the distance weight function used in prediction. Table I gives the top five models ordered by mean cross-validation score they achieving during the grid search. The values of k are all low in the table indicating that higher values of k result in the model underfitting and not performing as well on the validation sets. This table also suggests that the choice of distance metric has a smaller effect on performance compared to the number of neighbours. Additionally, using distance-based weighting appears consistently beneficial, as it prioritises closer neighbours and improves prediction accuracy.

TABLE I
Top Performing KNN Models with Different Hyperparameters

Model	Mean Score	Std CV Score	Metric	k	Weights
1	0.8312	0.0019	manhattan	3	distance
2	0.8310	0.0030	manhattan	5	distance
3	0.8295	0.0015	euclidean	3	distance
4	0.8295	0.0015	minkowski	3	distance
5	0.8281	0.0028	manhattan	7	distance

Experimentation exhibited in Figure 1 shows how as the value of k increases, the training and test error both decrease and never trend upward again. We confirm that the parameters [3, 5, 7, 9, 11, 15] over which we performed the grid search are reasonable since they should all result in their own balance between underfitting and overfitting. The k value of one was excluded from this search due to the glaring difference which will always occur between the test and the train accuracies. Adding the value would also increase the computational complexity unnecessarily.

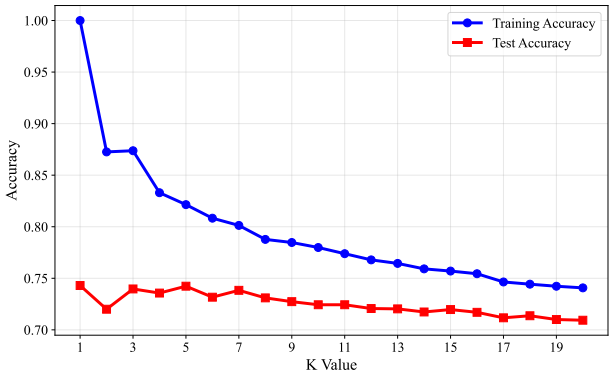

Fig. 1. Training vs. test accuracy of kNN algorithm as k increases.

2) Classification Trees: The classification trees parameter tuning resulting in the optimal model with the gini criterion being used, a maximum depth of 25, no maximum number of features, the minimum number of samples in a leaf to be one and the minimum number of samples required to split on being two. Table II presents the top five models from the grid search ranked by mean cross-validation score. The results

indicate that deeper trees tend to perform better, as shown by the highest-scoring model having a depth of 25. Reducing the maximum depth or increasing the minimum number of samples required to split generally leads to a slight decrease in performance, suggesting that the dataset benefits from the ability to model complex interactions.

The small differences in mean cross-validation scores among the top models indicate that the performance is relatively stable across minor variations in parameters. Additionally, the low standard deviations show that the model's performance is consistent across the folds of cross-validation. Overall, these results suggest that while the decision tree can capture non-linear relationships effectively, careful tuning of depth and split parameters is necessary to balance model complexity and generalisation.

TABLE II
COMPARISON OF DECISION TREE MODELS WITH DIFFERENT HYPERPARAMETERS

Model	Mean CV Score	Std CV Score	Parameters
1	0.7965	0.0018	gini, 25, None, 1, 2
2	0.7957	0.0037	gini, 20, None, 1, 5
3	0.7955	0.0021	gini, 25, None, 1, 5
4	0.7949	0.0039	gini, 25, None, 2, 2
5	0.7947	0.0026	gini, 20, None, 1, 10

[a]Parameter order: criterion, max_depth, max_features, min_samples_leaf, min_samples_split.

The effect of the depth parameter is shown by Figure 2. We observe that as the maximum depth of the classification tree increase, the training error decreases as expected. The test error decreases until a point at which it begins to increase again due to overfitting. This is since the model becomes too complex and begins modelling noise as opposed to signal.

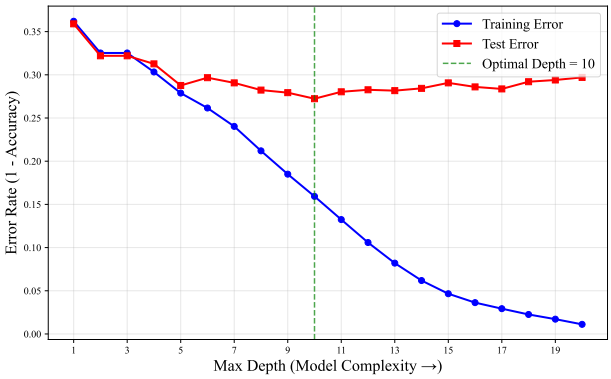

Fig. 2. Effect of maximum depth parameter on classification trees.

B. Cross-validation Results

Note that scores presented in Table II and Table I represent the accuracy on a small subset (20% for kNN and 10% for the classification trees) of the data. The following results are from the model applied to the *entire* dataset. Ten-fold cross-validation was used since it provides an optimal balance between computational efficiency and reliable performance estimation, as it uses 90% of the data for training while ensuring each observation serves as validation data exactly once. This approach minimizes both bias and variance in performance estimates compared to smaller fold numbers, while avoiding the computational overhead and potential instability associated with leave-one-out cross-validation.

1) K-nearest Neighbours: For each fold in the 10-fold cross-validation, the SMOTE and Tomek techniques were all only applied to training samples. This resulted in a kNN mean accuracy of 0.9184 across the 10 folds. The accuracy scores ranged from 0.9170 to 0.9202 which showed that the algorithm generalises well. The high one-vs-rest ROC-AUC score of 0.9721 demonstrates excellent discriminatory ability across all seven classes, significantly outperforming the classification tree model. The model achieves strong precision (0.9196) and recall (0.9184) balance, indicating effective identification across the multiclass problem. The F1-score of 0.9187 confirms the robust balance between precision and recall. The low standard deviations across all metrics indicate stable and reliable classification performance across different data splits. The kNN algorithm also reported an MCC of 0.8591 and a Cohen's Kappa of 0.8590. These are similar since they are both change-corrected metrics that account for class imbalance. These metrics indicate that we have obtained consistent performance across classes.

TABLE III
PERFORMANCE METRICS OF KNN AND CLASSIFICATION TREE MODELS (MEAN ± STD) OVER 10 FOLDS

Metric	KNN	Classification Tree
Accuracy	0.9184 ± 0.0011	0.9276 ± 0.0023
Precision	0.9196 ± 0.0011	0.9280 ± 0.0024
Recall	0.9184 ± 0.0011	0.9276 ± 0.0023
F1-Score	0.9187 ± 0.0011	0.9278 ± 0.0023
ROC-AUC	0.9721 ± 0.0005	0.9562 ± 0.0015

2) Classification Trees: For the classification trees 10-fold cross-validation was performed with SMOTE only applied to the training folds. The maximum accuracy of all runs was 0.9309 and the minimum was 0.9233. The mean accuracy of 0.9276 and other metrics can be observed in Table III. The very low standard deviation of 0.0023 on the accuracy metric indicates that the model has stable performance. While achieving higher overall accuracy than kNN, the classification tree model shows a lower one-vs-rest ROC-AUC score of 0.9562, suggesting slightly reduced discriminatory power when distinguishing between individual classes and all others. The model demonstrates excellent precision (0.9280), recall (0.9276), and F1-score (0.9278) performance. The superior performance across accuracy, precision, recall, and F1-score metrics suggests better overall classification effectiveness compared to the kNN approach. The consistently low standard deviations across all metrics confirm the model's stability and reliability in multiclass prediction tasks. The classification trees algorithm reported a MCC of 0.8196 and a Cohen's Kappa of 0.8196. These are similar since they are both change-

corrected metrics that account for class imbalance. We observe that these values indicate excellent classification performance, with the model achieving strong agreement beyond what would be expected by random chance and demonstrating robust predictive capability even in the presence of imbalanced class distributions.

C. Learning Curves

Figure 4 provides a visual illustration of how the classification tree learns the signal in the data as the size of the training set increases. This figure was generated using only 10% of the data for computational purposes. We observe the upward trend of the validation score and the slight downward trend of the training score. This is indicative of the model improving on its generalisation performance. Considering these trends, the results reported in Table III make sense. We see how the validation score would eventually increase to approximately 0.9184 as the training set increases in size to the full dataset. The learning curve for the kNN algorithm in Figure 3 is similar, however due to the small sample dataset, the training score is near perfect. This will change as more minority classes are included in the larger dataset.

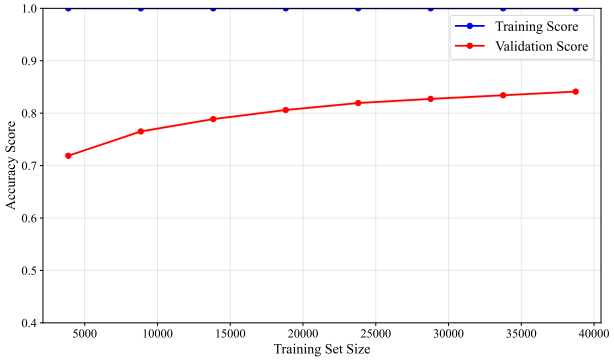

Fig. 3. The learning curve of the kNN algorithm.

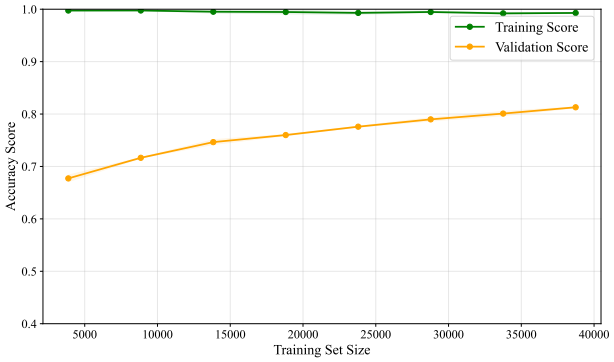

Fig. 4. The learning curve of the classification trees algorithm.

D. Computational Analysis

The majority of the computational complexity stemmed from the grid search approach used in the control parameters tuning. Despite sampling a significantly smaller portion of the data, the number of parameter combinations (36 and 384 respectively) still increased the computation time drastically. Since five-fold cross-validation was used for each combination in kNN, there were $36 \times 5 = 180$ fits to run. Only three folds were used for decision trees due to the significantly larger number of combinations. This resulted in $384 \times 3 = 1152$ fits to run. The addition of the Tomek link undersampling greatly increased the running time of the parameter tuning process. This is due to the SMOTE and Tomek (for kNN) undersampling being applied only to the training data in *each* iteration of the parameter tuning process. One complex optimisation would be to store the calculations required for this technique and re-use them over each iteration, however this would require a substantial amount of memory and care to prevent data leakage. The Tomek link procedure adds considerable computational overhead as it requires identifying and removing borderline instances through nearest neighbour calculations for every data point in each training fold.

To evaluate the performance of the models thoroughly, it was imperative that all the data was used. Thus, 10-fold cross-validation over the entire dataset was employed to ensure robust performance estimates. The computational analysis revealed significant differences in training efficiency, with each fold requiring approximately 12.6 minutes for kNN and 2.1 minutes for Decision Trees, highlighting the substantially higher computational cost associated with kNN's distance-based lazy learning approach compared to the tree-based eager learning methodology.

E. Statistical Significance Testing

The Mann–Whitney U test [1] was performed using the 10 cross-validation scores previously described. This resulted in a p-value of 0.00018 which is very small, therefore indicating there is strong evidence to reject the null hypothesis that the classification trees and the kNN scores come from the same distribution.

F. Feature Importance

The most useful features for reducing the impurity of the nodes in the classification tree algorithm are seen in Table IV. The feature `Elevation` contributes the most to the reducing the Gini impurity. Note that the absolute magnitude of these feature importance values have no intrinsic meaning, since they are normalised to sum to one. Interpreting the relative magnitude of the `Elevation` feature indicates that it contributes roughly three times as much as the second most important feature, `Horizontal_Distance_To_Roadways`. A shortfall of this table is since the many soil type features are encoded with dummy variables, they are not individually as informative on their own, but collectively may contribute an even larger amount to the reduction of impurity.

VI. Conclusion

Our goals were to develop the classification models for the provided dataset, to identify data quality issues that need to be

TABLE IV
TOP 10 FEATURE IMPORTANCES FROM DECISION TREE CLASSIFIER

Feature	Importance
Elevation	0.3463
Horizontal_Distance_To_Roadways	0.1246
Horizontal_Distance_To_Fire_Points	0.1074
Horizontal_Distance_To_Hydrology	0.0607
Vertical_Distance_To_Hydrology	0.0314
Soil_Type3	0.0289
Wilderness_Area3	0.0277
Hillshade_9am	0.0248
Hillshade_Noon	0.0243
Wilderness_Area1	0.0190

addressed, to identify and implement the necessary data transformations for the classification models, and to compare the performance of the implemented models. We achieved these goals by starting with a thorough theoretical understanding of the models, thereafter making informed decisions on how to go about pre-processing with the appropriate theoretical context. The model-specific data pre-processing steps aided in achieving extremely high accuracies of 91.84% for kNN and 92.76% for classification trees which performed better overall. We found through feature importance analysis that the `Elevation` feature had the most power. The kNN algorithm showed better discriminatory ability with a higher ROC-AUC score.

Through rigorous methodology and careful algorithm selection, this comparative study demonstrates that thoughtful pre-processing can unlock the predictive potential hidden within complex datasets.

REFERENCES

[1] H. B. Mann, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[2] David Nicolay, "ml441_assignments: assignment 2," https://github.com/Voltzz9/ml441_assignments/tree/main/assign2, 2025, [Online; accessed 4-Sep-2025].

[3] J. Kelleher, B. Namee, and A. D'Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics, second edition: Algorithms, Worked Examples, and Case Studies*. MIT Press, 2020. [Online]. Available: https://books.google.co.za/books?id=wtGMEAAAQBAJ

[4] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[5] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[6] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[7] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.