**Predicting Forest Cover Types using Machine Learning**

**Approaches**

**ABSTRACT**

In this project, we explore multiple classification approaches using the Forest CoverType dataset from the UCI Machine Learning Repository [1]. Our objective is to evaluate and compare multiple machine learning models on the Covertype dataset with over half a million instances and a mix of numerical and categorical attributes. We employ Logistic Regression, K-Nearest Neighbors, Decision Tree, and Multilayer Perceptron Neural Networks. For each approach, we try to find the best hyper-parameters. We provide a detailed explanation of our methodology, analysis of results, and the future direction for improvement and deployment.

**KEYWORDS**

Machine Learning, Classification, Logistic Regression, Decision tree, Nearest Neighbors, Neural Networks

**1 INTRODUCTION**

Forest classification plays a vital role in ecological monitoring, forest resource management, biodiversity assessment, and environmental planning. Accurate classification of forest cover types enables forest authorities and environmental agencies to make informed decisions regarding conservation strategies, wildfire prevention, reforestation planning, and habitat preservation. With the growing availability of environmental and topographic data, machine learning has emerged as a powerful tool to automate the classification of complex data.

The Forest CoverType dataset provided by the UCI Machine Learning Repository. It contains a total of 581012 instances with 54 attributes describing various environmental variables [1]. These include numerical features such as elevation, slope, aspect, and hillshade alongside binary indicators for wilderness area and soil type. The classification task involves predicting one of seven forest cover types based solely on these physical and geological characteristics. This makes the problem both challenging and highly applicable to real-world forestry analytics.

Our goal in this project is to evaluate the performance of various supervised machine learning approaches those covered in our Machine Learning and Pattern Recognition coursework. These approaches include Logistic Regression, K-Nearest Neighbors (KNN), Decision Trees, and Multilayer Perceptron Neural Networks. Beyond evaluating accuracy, we aim to find the best hyper-parameter for each approach.

The Forest CoverType dataset comprises a total of 54 features, of which 10 are continuous numerical variables and 44 are binary indicators representing categorical information. One key advantage of using this dataset is its cleanliness and lack of missing values, which minimizes the need for extensive preprocessing and allows for the direct application of machine learning algorithms. The continuous features include important topographical metrics such as Elevation, Aspect, Slope, Horizontal and Vertical Distance to Hydrology, and distances to roadways and fire points. These variables provide essential spatial and environmental context. The binary features encode the presence or absence of specific soil types and wilderness areas.

In this study, we aim to investigate the performance of different models in predicting all classes using the Covertype dataset. Specifically, we address the following research questions: How accurately can machine learning models predict the different classes? What hyperparameter settings optimize the generalization performance for each model? Additionally, how does class imbalance influence the per-class prediction performance across different modeling approaches?

**2 PRIOR WORK**

Many researchers have studied how to predict the type of forest cover using different machine learning methods. One important study was done by Blackard and Dean [2], where they compared the performance of an artificial neural network with a linear discriminant classifier. Their results showed that the neural network, which is a non-linear model, was more accurate for predicting forest cover types. This highlighted that non-linear models can better handle complex classification problems than simple linear ones.

Another related study focused on using neural networks to map ecological land systems based on forest cover data [3]. In this research, they used neural network outputs to classify land types and then compared the results to existing ecological maps. The highest accuracy they achieved was 52.0%. Although this number was not very high, the researchers also developed a useful method to estimate how certain the neural network was about each prediction, which can help improve mapping reliability.

In another study, Pratibha and her colleagues focused on the K-Nearest Neighbor (KNN) algorithm to classify forest cover types [4]. They tested its performance on the Covertype dataset and found that it could effectively classify forest cover types when properly tuned. Their research showed that even simple algorithms like KNN can be powerful if used in the right way.

Arvind and his team explored more advanced machine learning techniques [5]. They used an ensemble approach, where they combined three different algorithms such as Decision Trees, Random Forests, and K-Nearest Neighbors. They generated a final prediction from these models. Their results showed that combining models in this way improved the accuracy and consistency of forest cover classification. This suggests that using multiple models together can lead to better results than relying on a single model.

**3 METHODOLOGY**

In this study, we conducted a detailed comparison of four machine learning models which are Logistic Regression, K-Nearest Neighbors, Decision Tree, and Multilayer Perceptron. Our approach involved optimizing hyperparameters, validating the models, and using consistent evaluation metrics such as accuracy, F1-score, and precision to ensure a fair comparison.

Each model was built using the Scikit-learn framework and trained on the Forest CoverType dataset [6]. To find the best hyperparameters for each model, we used GridSearchCV, which performs a grid search along with k-fold cross-validation. Specifically, we used 5-fold stratified cross-validation, ensuring that the distribution of target classes is maintained in each fold. This approach is especially important due to the class imbalance present in the dataset.

The grid search process was exhaustive within the predefined hyperparameter ranges. For each combination of parameters, the training data was partitioned into five folds. This process was repeated five times with a different validation fold each time, and the results were averaged to estimate the generalization performance. The configuration yielding the best mean cross-validation score was selected for final model evaluation on the held-out test set.

We defined hyperparameters for each model and experimented with different values for each one. By testing these various combinations, we were able to identify the optimal hyperparameter values that gave the best results on the test data. For Logistic Regression, the key hyperparameter adjusted was the regularization parameter, which controls the strength of regularization. A lower value of C implies stronger regularization, helping to prevent the model from fitting noise in the data. By experimenting with different values of C, we were able to find the optimal balance between bias and variance for our dataset.

For K-Nearest Neighbors (KNN), we experimented with different values of k, the number of neighbors, using a range of odd integers. We also tested both uniform and distance-weighted voting schemes to determine the best configuration. Due to the model's high computational cost on large datasets, we downsampled the training data and reserved the full dataset for final evaluation.

For the Decision Tree, we aimed to control the tree's complexity to prevent overfitting. We tuned the maximum depth of the tree, limiting it to prevent it from growing too deep and fitting noise. Additionally, we adjusted the minimum samples per split, which determines how many samples are required to create a split at each node. This helped to control the model's ability to create specific rules and further reduced the risk of overfitting.

Lastly, for the Multi-Layer Perceptron, we fine-tuned several architectural and optimization parameters. These included the number of hidden layers, which impact the model's capacity to learn complex patterns. We also adjusted the alpha parameter for regularization and the maximum number of iterations for training. To enhance model performance, we employed the ReLU activation function, which is commonly used in deep learning models for its ability to speed up training and prevent the vanishing gradient problem [7].

For each algorithm, we established a well-defined search space. To optimize the hyperparameters, we employed either grid search or random search methods in conjunction with 5-fold stratified cross-validation. This approach allowed us to thoroughly explore the parameter ranges outlined below, ensuring a comprehensive evaluation of the model's performance across a variety of settings.

* Logistic Regression: solver ∈ {lbfgs, liblinear}, C ∈ {0.01, 0.1, 1, 10, 100}
* KNN: k ∈ {3, 5, 7, 9}, weighting ∈ {uniform, distance}, metric ∈ {Euclidean, Manhattan}
* Decision Tree: criterion ∈ {gini, entropy}, max\_depth ∈ {None, 10, 20, 30}, min sample split ∈ {2, 5, 10}, min sample leaf ∈ {1, 2, 4}
* MLP: hidden\_layer\_sizes ∈ {(100), (100,50), (150,100,50), (200), activation ∈ {ReLU, tanh}, α ∈ {1e-4, 1e-3, 1e-2}, learning rate init ∈ {1e-3, 1e-2}, batch\_size ∈ {64, 128}

**4 EXPERIMENTAL RESULTS**

We conducted all experiments on the Google Colab environment. The full Covertype dataset comprises 581012 samples across seven cover types, with classes 1 and 2 accounting for 495141 instances, which would overwhelm the available CPU and memory during exhaustive 5-fold GridSearchCV. To create a more manageable and balanced training set, we downsampled the entire dataset to 10% of its original size, maintaining all classes but reducing the overall sample count to 58101. We evaluated all approaches using this downsampled dataset with all seven classes represented. We divided the dataset into two subsets, 80% of the data was allocated for training the model, while the remaining 20% was reserved for testing and evaluating its performance.

The machine learning models used in this study were tuned by testing a wide range of key hyperparameters, as summarized in Table 1. We carefully selected and explored various combinations of these values to ensure that the models were configured for optimal performance under the given conditions, ultimately identifying the best set of hyperparameters for each model.

| **Model** | **Hyperparameters** |
| --- | --- |
| Logistic Regression | C: 100, solver: lbfgs |
| K-Nearest Neighbors | k: 3, weights: distance, metric: Manhattan |
| Decision Tree | criterion: gini, max\_depth, min\_samples\_leaf: 1,  min\_samples\_split: 2, |
| MLP Neural Network | learning\_rate\_init: 0.001, learning\_rate: constant, hidden\_layer\_sizes: {100, 50}, alpha: 0.01, activation: ReLU, |

Table 1: Models and Hyperparameters

To evaluate the effectiveness of the models, we used accuracy, precision, recall, and F1-score performance metrics. Accuracy offers a general sense of correctness, while F1-score better reflects how the model performs across all classes, especially in imbalanced scenarios. Precision indicates how many of the positive predictions made by the model are actually correct, focusing on the quality of positive predictions. Recall measures how many actual positive instances were correctly identified by the model, emphasizing the model’s ability to capture all relevant positives. Together, precision and recall help balance the trade-off between false positives and false negatives.

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| --- | --- | --- | --- | --- |
| Logistic Regression | 0.7268 | 0.7131 | 0.7268 | 0.7168 |
| K-Nearest Neighbors | 0.8972 | 0.8969 | 0.8972 | 0.8970 |
| Decision Tree | 0.8188 | 0.8193 | 0.8164 | 0.8187 |
| MLP Neural Networks | 0.7278 | 0.7373 | 0.7278 | 0.7129 |

Table 2: Model performance metrics after 5-fold CV hyper-parameter tuning

Table 2 presents the summarized outcomes of the final evaluations for each model, with hyperparameters tuned as described earlier. The test set used was untouched during training and crossvalidation, ensuring unbiased estimates of performance.

**4.1 Logistic Regression**

The logistic regression model, optimized via exhaustive GridSearchCV with 5‐fold stratified cross‐validation, achieved its best performance using a regularization strength of C = 100 and the lbfgs solver. On the held-out test set, the model attained an overall accuracy of 72.68%, with weighted‐averaged precision, recall, and F₁‐score of 71.31%, 72.68%, and 71.68%. Logistic Regression served as our baseline. As expected, this linear model struggled to capture the complex non-linear relationships present in the data. Although efficient in terms of training time and resource usage, it consistently underperformed, particularly on minority classes where the decision boundaries were not easily separable. Its F1-score was relatively low, indicating that it had difficulty balancing precision and recall across all classes.

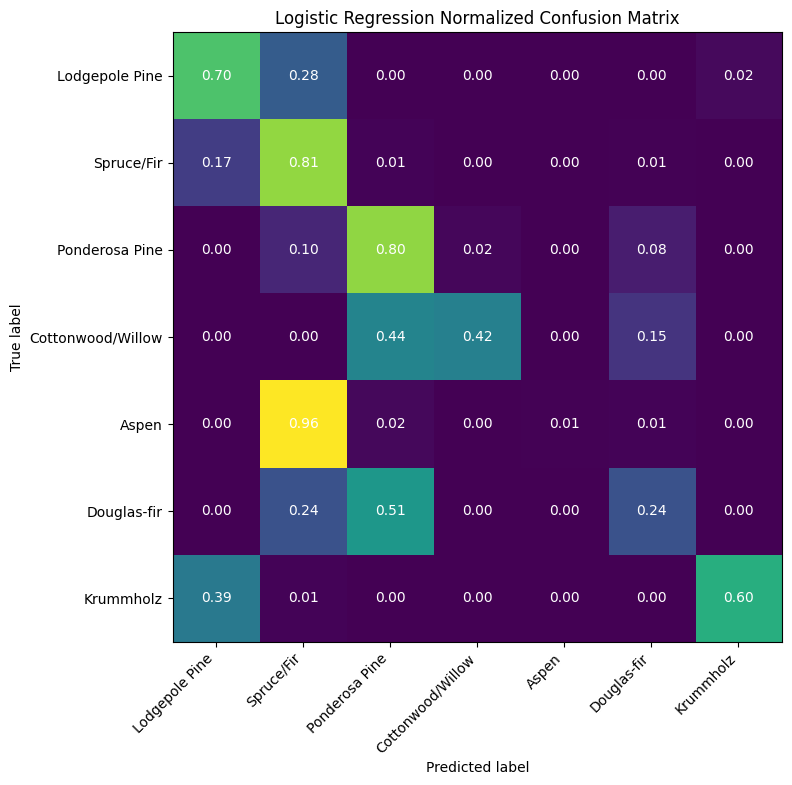


Figure 1: Normalized Confusion Matrix for Logistic Regression

**4.2 K-Nearest Neighbors**

To identify the optimal K-NN configuration, we performed exhaustive hyperparameter tuning via scikit-learn’s GridSearchCV. We defined a search grid over the number of neighbors k={3, 5, 7, 9}, voting scheme weights={uniform and distance}. We evaluated every combination using stratified 5-fold cross-validation on the training set. K-Nearest Neighbors showed a modest improvement in accuracy and F1-score. GridSearchCV aggregated each combination’s cross-validated F1, automatically refitted the model on the full training data with the best setting, and revealed a 3-NN classifier. The K-Nearest Neighbors classifier achieved its good performance on the Covertype test set with a single neighbor (n = 3) and distance voting weights. Under this configuration, overall accuracy reached 89.72%, while weighted‐averaged precision, recall, and F1‐score were 89.69%, 89.72%, and 89.70%. Class‐specific The macro‐average F1‐score of 84.30% underscores the classifier’s strong generalization across all cover types, demonstrating that even a simple 3-NN model can yield high‐fidelity predictions on this high-dimensional, multi‐class problem.

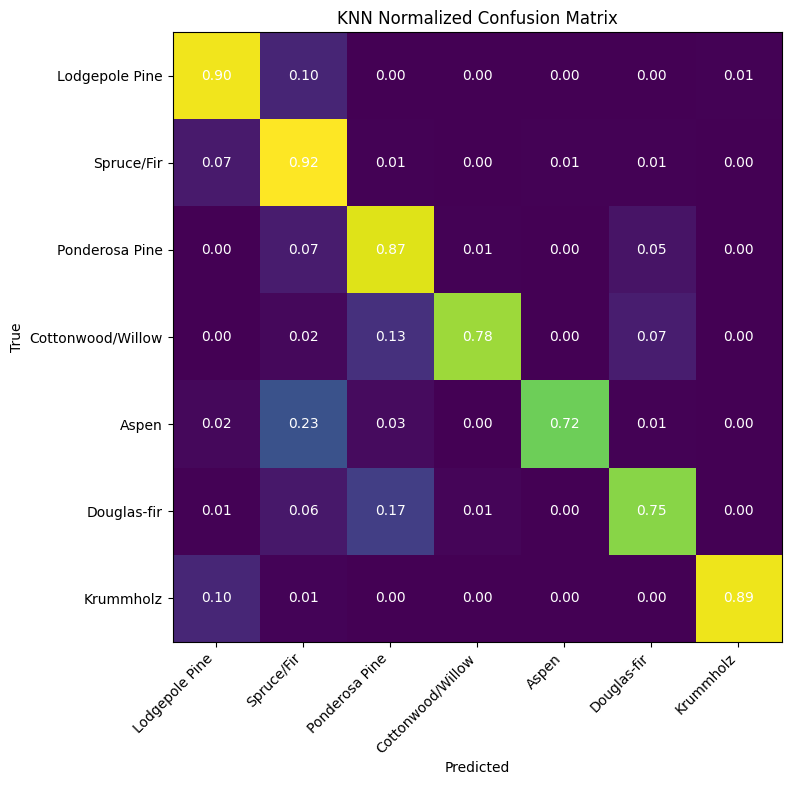


Figure 2: Normalized Confusion Matrix for K-Nearest Neighbors

**4.3 Decision Tree**

Decision Tree models performed significantly better than Logistic Regression. With tuned depth and split parameters, we were able to control overfitting and generalize well. The model was efficient, and training time remained reasonable. The Decision Tree classifier, tuned via 5-fold grid search, achieved its best performance using an information‐gain entropy split criterion, max\_depth=None, a minimum of one sample per leaf and a minimum split size of two samples. The accuracy, precision, and recall were 81.88%, 81.93%, 81.64%. The macro‐averaged F1‐score of 81.87% further confirms that the model performs robustly, demonstrating both its discriminative power and stability despite class‐imbalance. However, despite relatively strong performance, the Decision Tree tended to favor majority classes, a limitation we observed in its confusion matrix.

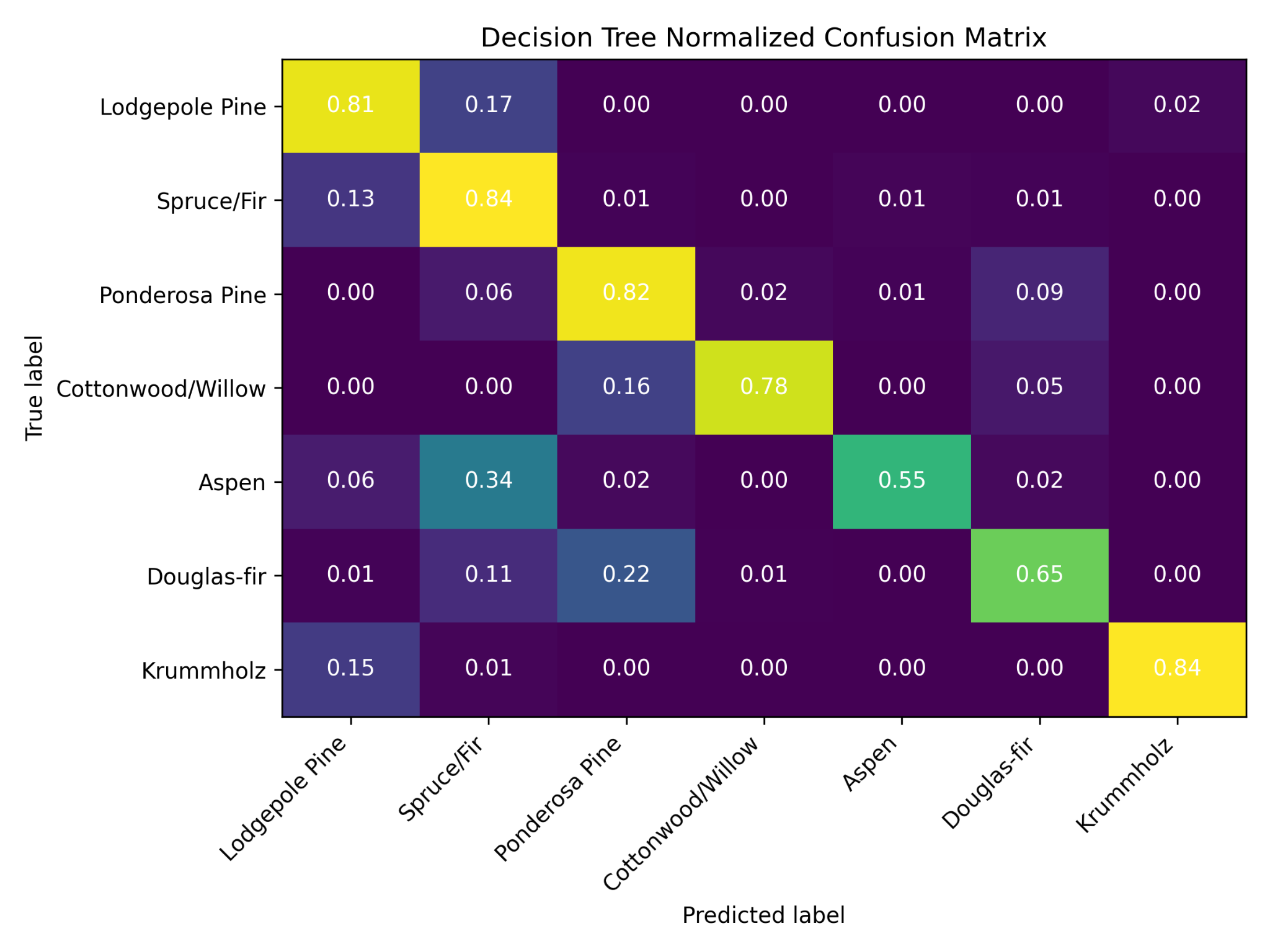


Figure 3: Normalized Confusion Matrix for Decision Tree

**4.4 Multilayer Perceptron Neural Networks**

The deep learning approach enabled the model to capture intricate non-linear patterns that simpler models missed. After trying different hyper parameter values, the final MLP model used two hidden layers 100 and 50 neurons with ReLU activation. The model trained by the Adam optimizer learning rate = 0.001, a weight decay (alpha) of 0.01, and batch size of 64. The model reached 72.78% accuracy, 73.73% precision, 72.78% recall, and a 71.29% F1-score. MLP required significantly longer training and was highly sensitive to hyperparameter tuning. Without careful regularization and early stopping, overfitting became a serious issue. Nevertheless, once tuned, the model achieved high macro-F1 scores, showing improved balance across all classes.

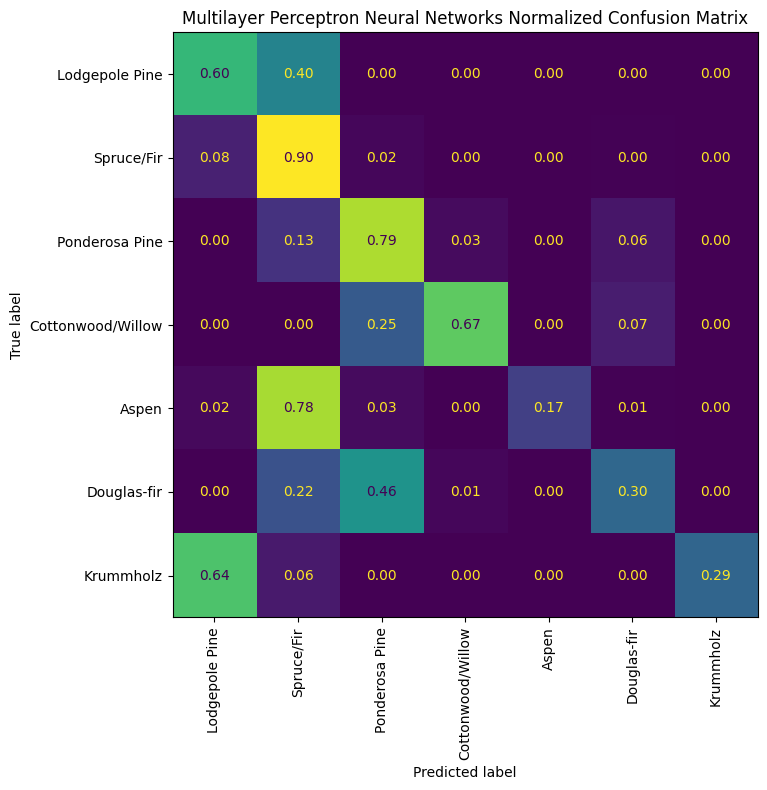


Figure 4: Normalized Confusion Matrix for Decision Tree

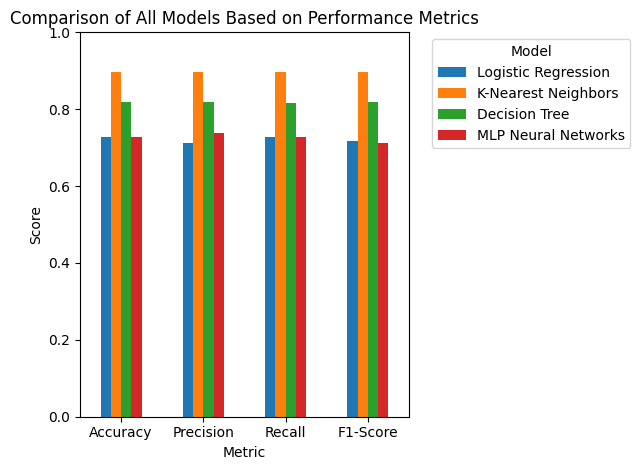


Figure 5: Comparison of All Models Based on Performance Metrics

**5 DISCUSSION**

This section reflects on the outcomes of our experiments and critically evaluates the comparative performance of different supervised learning approaches explored throughout this project. As required by the project guidelines, we selected a dataset that exceeds the minimum threshold of 5,000 instances and 20 attributes. We performed the experiment using all classes of the Forest CoverType dataset. Due to limitations in the training environment, we downsampled the dataset to 10% of its original size. This reduction preserved all classes while lowering the total sample count to 58101, making the training process more manageable. The dataset provided an ideal for benchmarking multiple models under realistic, high-dimensional, and imbalanced conditions.

The key goal was to assess how the machine learning models covered in class which are Logistic Regression, K-Nearest Neighbors, Decision Tree and Multilayer Perceptron perform on this dataset in predicting the correct forest cover types. Each model’s architecture, hyperparameters, and decision logic were carefully examined, implemented, and tuned using Python libraries, primarily Scikit-learn, supported by auxiliary libraries such as NumPy, Pandas, and Matplotlib for preprocessing, evaluation, and visualization.

Logistic Regression, while computationally efficient and easy to interpret, was clearly inadequate for this classification problem. Its linear decision boundaries failed to capture the complex, non-linear relationships inherent in the feature space, particularly between topographic variables and cover type classes. Even with careful regularization through the C parameter, the model struggled to generalize well, particularly on underrepresented classes. Precision and recall for most classes were markedly lower compared to other models.

K-Nearest Neighbors offered modest improvements in accuracy and F1 score, but its practical utility was limited by poor scalability. Since KNN is a non-parametric, memory-based algorithm, the prediction time grows linearly with the number of training instances. Given our dataset’s scale, even a moderate grid search over values of k became computationally expensive. This further validated the importance of considering algorithm complexity, not just predictive power, especially in big data contexts.

Decision Trees proved highly sensitive to their hyperparameters, especially max\_depth and min\_samples\_split. When carefully tuned, they provided interpretable decision structures with relatively good accuracy. However, they also exhibited a strong tendency to overfit, especially when allowed to grow unrestricted. Their performance on minority classes was still skewed due to class imbalance.

Multilayer Perceptron demonstrated competitive accuracy and strong F1 scores but required significantly more effort to tune and longer training times. The model’s sensitivity to hyperparameters like alpha , hidden\_layer\_sizes, and early stopping thresholds made experimentation more resource intensive. However, once properly configured, MLP proved capable of capturing deep non-linear interactions in the data. It showed relatively balanced performance across classes, including improved recall for minority classes compared to other methods.

Across all models, the confusion matrices highlighted a consistent trend of poor performance on minority classes, particularly classes 4 and 5. Class 4 and Class 5 have considerably fewer instances compared to the other classes. This issue is largely attributed to data imbalance, which causes many classification algorithms to favor the majority classes. Among the models, Logistic Regression and Decision Trees were most susceptible to this bias, while the MLP model exhibited relatively better robustness.

One of the project’s key requirements was to identify the best hyperparameters for each model. This was achieved via exhaustive grid search using GridSearchCV with 5-fold stratified cross-validation. This approach ensured that our tuning process was strict and that final performance metrics were generalizable. Each model’s optimal configuration was derived through systematic experimentation over a defined parameter space.

All models analyzed in this study were covered in the coursework of our Machine Learning and Pattern Recognition class. This project gave us a hands-on opportunity to translate theoretical concepts such as bias-variance tradeoff, model complexity, hyperparameter tuning, and evaluation metrics into real-world practice. Each model’s strengths and limitations became clearer through practical application, reinforcing classroom learning.

In summary, the experiments validate the effectiveness of ensemble learning and neural networks for structured, high-dimensional classification tasks. While traditional models like Logistic Regression and KNN have educational value and interpretability, they are less suited to complex datasets like Forest CoverType. Models that incorporate randomness, regularization, and non-linearity are better positioned to achieve strong, balanced performance. No single model is universally optimal, and effective machine learning requires matching model capabilities with data characteristics.

**6 CONCLUSION**

This research project set out to compare several supervised learning models, specifically those discussed in our Machine Learning and Pattern Recognition course for the purpose of classifying forest cover types using the UCI Forest CoverType dataset. The dataset not only satisfied the requirement of having more than 5,000 instances and 20 attributes, but also posed real-world complexities.

Through a structured pipeline of data preprocessing, model implementation, hyperparameter tuning, and evaluation, we explored the behavior and capabilities of four different classifiers: Logistic Regression, K-Nearest Neighbors, Decision Tree and Multilayer Perceptron Neural Network. Our experiments demonstrated that no single algorithm is universally best. Each model has its strengths and limitations. MLP was a close contender, showing exceptional performance in capturing complex patterns, though it required more intensive tuning and computational time.

Logistic Regression, while intuitive and fast, underperformed due to its linear nature and inability to handle non-linearity. KNN showed moderate accuracy but was computationally inefficient at this dataset scale. Decision Trees were interpretable and performed well with tuning but tended to overfit without pruning. MLP, representing a more complex model architecture, handled non-linearity very well but required thoughtful regularization and training strategy to avoid overfitting.

Throughout the project, we relied on Python-based tools such as Scikit-learn, NumPy, Pandas, and Matplotlib for building models, processing data, and visualizing results. GridSearchCV was used extensively to find the best hyperparameters, and evaluation was carried out using stratified 5-fold cross-validation to ensure generalizability and fairness. Our approach involves building and implementing classification models from the ground up using Scikit-learn. We establish a solid experimental framework with cross-validation and fair performance comparisons. Each model undergoes hyperparameter tuning to find the best settings, and we evaluate the results using confusion matrices and classification reports.

**7 FUTURE WORK**

While our current approach has delivered strong results, there are several promising avenues for future exploration that could further enhance performance and expand the model’s capabilities. One of the key directions we plan to pursue is the integration of all data in order to evaluate their impact in a more powerful training environment. This will allow us to leverage enhanced computational resources, potentially leading to better model accuracy and robustness.

Another area we’re focusing on is addressing class imbalance. Despite the overall success of the model, we’ve noticed that certain minority classes, such as Class 4 and Class 5, have been underperforming, especially in terms of recall and precision. To remedy this, we plan to implement synthetic oversampling techniques to balance the dataset and improve model performance on these underrepresented classes. This adjustment will help the model recognize and predict these minority classes more effectively.

Looking ahead, one of the most important objectives is to take the best-performing model and deploy it in real-world settings. We plan to wrap it as a RESTful API, making it easily accessible for various applications. This API will be hosted on a cloud platform, ensuring that the model can be used in real-time for forest type prediction. With this deployment, users will be able to quickly input data and receive accurate predictions, making the model a valuable tool for ecological research. By continuing to refine our model and exploring these new strategies, we’re confident that we can make significant improvements in both accuracy and real-world usability.

**8 TEAM JUSTIFICATION**

This project was completed through the equal and collaborative efforts of both of us. From the initial stage of dataset selection to the final preparation of the report and visualizations, we maintained continuous communication and coordination to ensure fair participation and distributed responsibility.

Firstly, we reviewed the dataset options in UCI Machine Learning Repository and decided to use the Covertype dataset. We analyzed the structure and characteristics of the dataset. We identified challenges such as class imbalance. Each team member was responsible for implementing and tuning two machine learning models. Ahmet focused on Logistic Regression and KNN, while Ayeshkant worked on Decision Trees and MLP. Once the models were developed, we cross-checked each other’s implementations to ensure consistency and accuracy. We collaborated closely on designing and executing grid searches using GridSearchCV, analyzing cross-validation results, and selecting the best hyper-parameter for each approach. Together, we reviewed model performance and classification reports, and each team member contributed to visualizing the results for better clarity and interpretation.

The report was co-authored, with each member contributing to different sections including Introduction, Methodology, Results, Discussion, and Conclusion. We reviewed each other’s work and revised sections for clarity, consistency, and academic tone. In terms of experimentation and project involvement, we followed a team-based approach to design controlled experiments and analyze outcomes, and refine our models iteratively.

**REFERENCES**

[1] UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci. edu/ml/datasets/covertype, [Accessed: June 20, 2025].

[2] J. A. Blackard and D. J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," Computers and Electronics in Agriculture, vol. 24, no. 3, pp. 131-151, 1999. [Online]. Available: https://doi.org/10.1016/S0168-1699(99)00046-0. [Accessed: Jun. 11, 2025].

[3] P. Gong, R. Pu, and J. Chen, "Mapping ecological land systems and classification uncertainties from digital elevation and forest-cover data using neural networks," *Photogrammetric Engineering & Remote Sensing*, vol. 62, no. 11, pp. 1249-1260, Nov. 1996.

[4] P. Maurya and A. Kumar, "Performance assessment of K-nearest neighbor algorithm for classification of forest cover type," in *Proceedings of the [Book Title]*, 2022, pp. 43–51, doi: 10.1007/978-981-16-4538-9\_5

[5] A. Kumar, P. Maurya, S. Tiwari, A. Ali, H. Vasisht, and A. Baghel, "Classification of forest cover-type using ensemble of decision tree, random forest and K nearest neighbor," *JIMS8I - Int. J. Inf. Commun. Comput. Technol.*, vol. 10, pp. 615-619, Jan. 2023, doi: 10.5958/2347-7202.2022.00011.1

[6] Scikit-learn: Machine Learning in Python. [Online]. Available: <https://scikit-learn.org/>, [Accessed: Jun. 18, 2025]

[7] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with ReLU activation," *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017