Tree/Plant leaf classification

Amanpreet Singh
Graduate Applied Computer Science
The University of Winnipeg
Winnipeg, Canada
singh-a69@webmail.uwinnipeg.ca

***Abstract*— In this paper, we explore the problem of plant leaf classification using various machine learning algorithms, including Decision Trees, K-Nearest Neighbors, and Random Forest. Our dataset consists of features such as shape, texture, and margin of leaves belonging to multiple species, with the goal of accurately predicting the correct species given a new feature set (Shape, texture and margin). The paper describes the methodology that includes data exploration, preprocessing, model training, and evaluation. The models are trained and evaluated using cross-validation techniques, and their performance is measured using accuracy, precision, recall, and F1-score metrics. Through our experiments, we demonstrate the effectiveness of Random Forest in achieving the highest accuracy of 91% among the three algorithms. We also investigate the impact of dimensionality reduction and hyperparameter optimization techniques such as Principal Component Analysis (PCA), Bayesian optimization, and Grid search on model performance. Our results show that these techniques can help to improve model performance while reducing the computational complexity of the models. Overall, our study provides insights into the potential of machine learning for plant leaf classification based on their shape, texture, and margin features, which can have important applications in fields such as agriculture and environmental monitoring.**

Keywords— Pattern Recognition, Tree/Plant Leaves Classification, k-Nearest Neighbors, Random Forest, Decision Trees, Principal Component Analysis (PCA), Bayesian optimization, Grid search, k-fold cross-validation

I. INTRODUCTION

Plant classification is a fundamental problem in botany, and has been a subject of interest for several decades. The identification of plant species is critical for various applications such as monitoring plant growth, disease detection, plant breeding, ecosystem monitoring, and conservation. Traditional plant identification methods involve manual identification by experts, which is a time-consuming and error-prone process. Recent advances in machine learning and computer vision have enabled the development of automated plant identification systems, which can identify plant species with high accuracy.

In this paper, we present a study on the classification of plant leaves using machine learning algorithms. We use the dataset provided by Kaggle, which contains 990 samples of plant leaves belonging to 99 different species. The dataset is preprocessed and cleaned before being used for training and testing different machine learning models. The research domain of this study involves machine learning algorithms applied to a specific dataset. The main objective of the study was to analyze and compare the performance of different machine learning algorithms, namely Random Forest, K-Nearest Neighbors, and Decision Tree, in classifying the dataset. To achieve this objective, various techniques such as PCA for dimensionality reduction, Bayesian optimization and Grid Search for hyperparameter tuning, and cross-validation for model evaluation were employed. The results show that the Random Forest algorithm outperformed the other algorithms with an accuracy of 91%. The study contributes to the field of machine learning by providing insights into the effectiveness of different algorithms for classification tasks on this particular dataset. Additionally, the study demonstrates the impact of various techniques such as hyperparameter tuning and dimensionality reduction on the performance of the models. These findings can be applied to other similar datasets, providing a basis for choosing the most appropriate algorithm for classification tasks.

The rest of the paper is organized as follows. Section 2 provides a detailed description of the dataset used in our study. Section 3 presents the methodology followed for the study, including data preprocessing, model training and evaluation. Section 4 discusses the results obtained, and Section 5 concludes the paper with a summary of the findings and possible future directions for research.

II. RELATED WORK

In the past few years, several leaf identification approaches using machine learning have been proposed. By combining shape, morphological data, colour histogram, distance maps, and k-Nearest Neighbour classifier with the Flavia dataset, Munisami et al. [1] presented a model. Using the UoM medicinal plant dataset and modified LBP and 1-nearest neighbour classifier, Naresh and Nagendraswamy [2] proposed a classification system for medicinal plants. Yang et al. [3] build a shape-based system using multi-scale triangular centroid distance and shape dissimilarity measurement with Swedish leaves. A medicinal plant recognition system was proposed by Begue et al. [4] by combining shape and colour features using a Random Forest classifier with ten-fold cross-validation. Kan et al. [5] proposed an automatic plant classification method based on shape features, three geometrical features, and GLCM texture characteristics using SVM classifier. For classification using SVM, Ali et al. [6] proposed a combined feature set using LBP and a bag of features. A multi-model plant classification system was proposed by Salve et al. [7] by fusing LBP and GIST features using feature-level and score-level fusion techniques. A model was proposed by Mostajer Kheirkhah and Asghari [8] proposed a model. This model is classified using the Cosine KNN classifier. The multi-layer perception with backpropagation (ML-BP) classifier was used by Pacifico et al. [9] to propose an automatic plant classification system based on colour and texture features. By combining shape and texture features, Sujith and Neethu [10] proposed a feature combination method to classify plants using an ANN classifier. Zhao et al. [11] proposed a growing convolution neural network (GCNN) method with variable topology structure for leaf recognition. Depending on the specific classification task, the GCNN can automatically adjust to the proper size. At the same time, they also provide a progressive sample

learning approach to automatically determine the size of training data, so as to avoid overfitting or underfitting in model training. Liu et al. [12] proposed a hybrid neural network for leaf identification, which combines the advantages of an autoencoder neural network and a convolutional neural network (CNN). They begin by extracting leaf characteristics using a hybrid neural network, and when it comes time to identify plants, they use an SVM classifier. Mallah C et al [15] proposed Plant leaf classification using probabilistic integration of shape, texture and margin features. Wu et al. [13] have introduced PNN technique for plant leaf recognition. Principal component analysis (PCA) was used to extract 12 features, which were then tested on 1800 leaves with a 90% accuracy rate. Saleema et al. [14] introduced various extraction methods and leaf visual cues in 2019 to obtain an optimal value of geometric, morphological, and texture features. In their work initially, the selected features were normalized and then transformed to the components of principal for reducing the redundant dimensions. To various plant species, the features are used to classify the images of leaf. Including multi-SVM, decision tree, naıve Bayes and k-nearest neighbor, different machine learning classifiers were evaluated.

III. Problem Statement and Techniques Used

The classification of plant leaves is a fundamental task in botany and ecology, with numerous applications in areas such as plant identification, species conservation, and ecosystem monitoring. However, manually identifying plant species based on their leaf characteristics can be time-consuming and error-prone.

To address this challenge, we aim to develop a machine learning model that can accurately classify plant species based on their leaf features like shape, texture and margin. The dataset used in this project is "train.csv", which contains information on the leaf features of various plant species. The dataset has a total of 990 observations and 194 features. Specifically, we will explore the performance of four different classifiers: Decision Tree, K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GaussianNB), and Random Forest. We will compare the performance of these models based on various evaluation metrics such as accuracy, precision, recall, F1-score, and loss function.

Our goal is to identify the most accurate and efficient classifier for the task of plant leaf classification, while also examining the impact of different feature selection and dimensionality reduction techniques on model performance. Through this project, we hope to contribute to the development of automated plant species identification systems that can improve efficiency and accuracy in botany and ecology research.

Below are the techniques/methods we have used in this work.

A. *Random Forest algorithm*

Random Forest [16] can be used for both classification and regression problems. It is based on the concept of ensemble learning, which combines multiple classifiers to improve the accuracy of the model and solve complex problems.

Random Forest consists of multiple decision trees trained on different subsets of the dataset. The method predicts the final result based on majority votes rather than depending just on one decision tree by averaging the predictions from all the trees. The accuracy increases and the risk of overfitting decreases as the number of trees in the forest increases. To ensure a better Random Forest classifier, there should be actual values in the feature variable of the dataset, and the predictions from each tree should have low correlations. Random Forest works in two phases: the first phase involves creating the forest by combining multiple decision trees, and the second phase involves making predictions for each tree created in the first phase.

The Random Forest algorithm tough to overfitting or high dimensionality of the data. Its ability to handle both classification and regression problems makes it a versatile tool for machine learning practitioners.

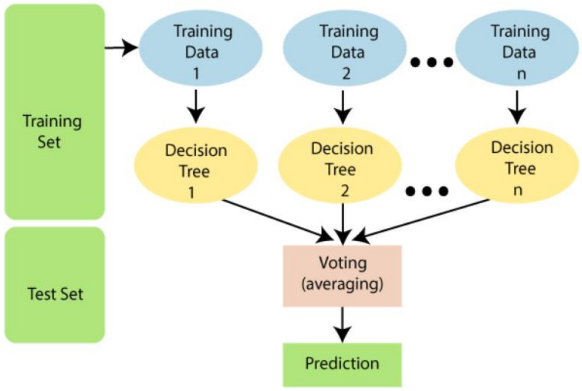


Fig. Random Forest

B. *Decision Tree algorithm*

Decision Tree [17] is a supervised learning technique that is widely used for solving classification and regression problems. It is represented by a tree-like structure, where the internal nodes depict the features of a dataset, branches represent the decision rules, and each leaf node signifies the outcome. In a decision tree, there are two types of nodes: decision nodes and leaf nodes. Decision nodes are responsible for making decisions based on multiple branches, while leaf nodes are the final output of those decisions and do not contain any further branches.

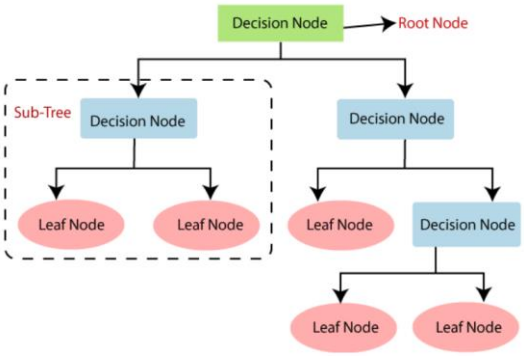


Fig. Decision tree

The root node represents the starting point of the decision tree and represents the entire dataset, which further gets divided into two or more homogeneous sets. The leaf nodes are the final output of the decision tree, and no further segregation is possible. Splitting is the process of dividing the decision node/root node into sub-nodes based on specific conditions. The branch or subtree is formed by splitting the decision tree. Pruning is the process of removing unwanted branches from the tree. The root node of the tree is called the parent node, and other nodes are referred to as child nodes.

The crucial issue while implementing a decision tree is selecting the best attribute for the root node and sub-nodes. Two popular techniques are Information Gain and Gini Index. Information gain is the reduction in entropy achieved by splitting the dataset based on a particular attribute, while Gini Index measures the degree of impurity of the sub-nodes created by splitting the dataset based on a particular attribute.

$$\text{Information Gain}(S,a) = \text{Entropy}(S) - \sum_{v \in values(a)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

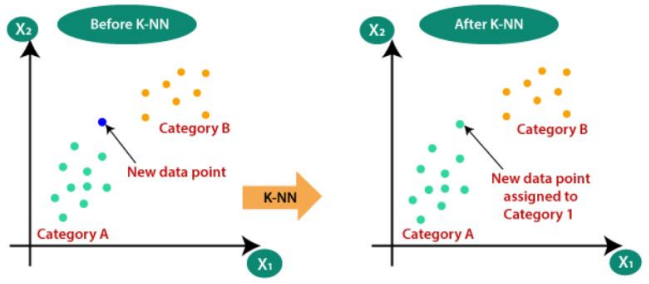
- a represents class label
- Entropy(S) is the entropy of dataset, S
- |Sv|/ |S| represents the proportion of the values in Sv to the number of values in dataset, S
- Entropy(Sv) is the entropy of dataset, Sv

$$\text{Entropy}(S) = -\sum_{c \in C} p(c)\log_2 p(c)$$

- S represents the data set that entropy is calculated
- c represents the classes in set, S
- p(c) represents the proportion of data points that belong to class c to the number of total data points in set, S

C. K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbor (K-NN) [18] is a powerful yet straightforward Machine Learning algorithm that operates based on the Supervised Learning technique. Its principle is to classify a new data point based on its similarity to previously stored data. K-NN is often referred to as a "lazy learner" algorithm because it does not learn from the training set immediately, but rather stores the dataset and performs an action on the dataset at the time of classification.



For instance, if we have two categories, A and B, and we encounter a new data point x1, we can use K-NN to determine which category it belongs to. To achieve this, we follow a series of steps. First, we select the number K of neighbors. Second, we calculate the Euclidean distance of the K number of neighbors. Third, we choose the K nearest neighbors based on the calculated Euclidean distance. Fourth, among these K neighbors, we count the number of data points in each category. Fifth, we assign the new data points to the category with the maximum number of neighbors. Finally, our model is ready.

K-NN is a versatile algorithm that can be used for both classification and regression problems. One of the advantages of K-NN is that it is easy to interpret and explain the results. However, the performance of K-NN depends heavily on the choice of K, and it can be computationally expensive for large datasets.

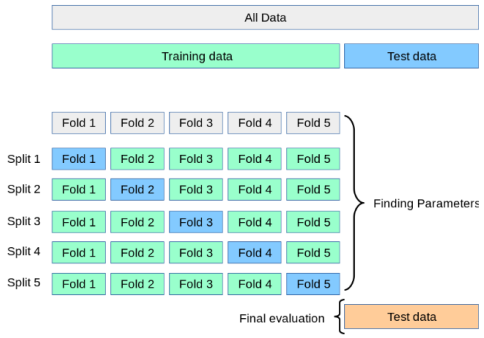
$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Euclidean distance formula

D. k-fold cross validation

When working with machine learning algorithms, it is common to fine-tune settings, also known as hyperparameters, for optimal performance. However, this can lead to overfitting, where the model performs well on the test set but poorly on new data. To address this issue, a validation set can be used, but this reduces the amount of data available for training the model.

One solution to this problem is to use a procedure called cross-validation (CV). In k-fold CV [19], the training set is divided into k smaller sets. For each fold, a model is trained using k-1 of the sets as training data and the remaining set as validation data. This process is repeated for each fold, with a different subset used for validation each time. The final performance measure is the average of the measures computed in each fold.



While k-fold CV can be computationally expensive, it has the advantage of using all available data for training and testing, which is particularly important when working with limited sample sizes. Other variations of

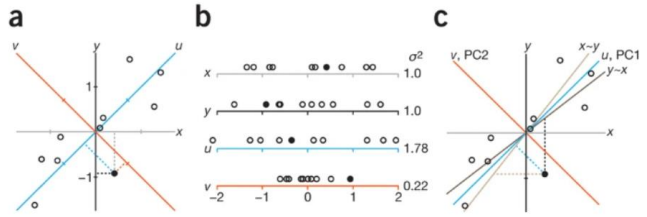
CV exist, but they follow the same basic principles. Ultimately, the goal of CV is to evaluate the model's performance in a way that generalizes well to new data, rather than just the specific data used for training and testing.

E. Principal component analysis (PCA)

PCA [20] is a useful method for analyzing high-dimensional data, such as those encountered in biology, where multiple features are measured for each sample. High-dimensional data can be computationally intensive and require correction for multiple tests when assessing associations between features and outcomes. PCA simplifies this process by transforming the data into a lower-dimensional space, where the most important features are retained. This unsupervised learning method is similar to clustering and finds patterns in the data without prior knowledge about sample groups or phenotypic differences.

PCA reduces the data by projecting it onto principal components (PCs), which are geometrically orthogonal and uncorrelated with all previous PCs. The first PC is selected to minimize the total distance between the data and its projection, while maximizing the variance of the projected points. Subsequent PCs are selected in a similar manner, with the added requirement of being uncorrelated with all previous PCs. This selection process maximizes the correlation between the data and its projection, equivalent to multiple linear regression on the projected data against each variable of the original data.

By reducing the data in this way, PCA simplifies complex datasets and retains the most important information. It is a valuable tool for analyzing high-dimensional data in a computationally efficient manner, and it can be used to identify patterns and trends that would otherwise be difficult to discern.



F. Bayesian optimization

Bayesian optimization is a powerful technique that combines exploration and exploitation in a sequential search framework to efficiently optimize a function. Unlike traditional methods such as grid search or random search, Bayesian optimization builds a probabilistic model of the underlying function, which helps it determine the best places to sample. This probabilistic model is initially uncertain about the function, but with each observation of the function, the model becomes more precise.

In addition to the probabilistic model, Bayesian optimization also uses an acquisition function that determines which points should be sampled next. This function is computed from the posterior distribution over the function and is defined on the same domain as the function itself. Depending on how the acquisition function is defined, it can encourage exploration or exploitation. Overall, Bayesian optimization is a powerful tool for optimizing complex functions, particularly when the number of function evaluations is limited.

IV. METHODOLOGY

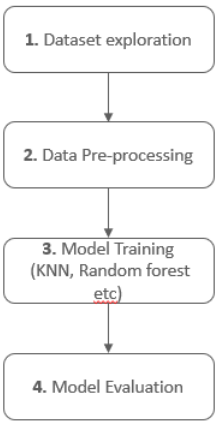
The methodology involves evaluating four different machine learning algorithms for leaf classification, including k-Nearest Neighbor, Naive Bayes, Decision Trees, and Neural Networks. The methodology includes data exploration, data preprocessing, model Training, model evaluation.

1) Dataset exploration

The dataset exploration process involved understanding the dataset and its features, identifying any missing or erroneous values, and gaining insights into the distribution of target variable (leaf species) to identify class imbalance which may affect the performance of our models.

Our dataset (Leaf.csv) consists of around 990 leaf samples (10 samples for each plant/tree). Three sets of features are given for each leaf sample: a shape of a leaf, texture of a leaf, and margin of a leaf. For each feature, a 64-attribute vector is given per leaf sample.

There is no missing value in our dataset. We checked this using “isnull().sum()”. Our classes are also balanced. We checked that our classes are balanced or not using “value_counts()”. In total we have 99 classes (99 types/species of leaves). For each leaf there are 10 samples.



2) Data Pre-processing

Data preprocessing involves several steps such as handling missing values, encoding categorical variables, and handling class imbalance (if any).

As we checked in Dataset exploration step, there are/is no missing values in the our dataset. The species column of our dataset has categorical variables i.e names of species of leaves. So, we have encoded them using label encoding

technique. For this we have used “LabelEncoder().fit()” in our script. Finally, there is no class imbalance to handle in our dataset as per the step 1.

By performing these preprocessing steps, we can ensure that our machine learning models are able to make accurate predictions on unseen data.

3) Model Training

In this phase, we used three machine learning algorithms: Decision tree, KNN, and Random forest. For each algorithm, we trained the model on the training set, and then evaluated its performance on the test set. We used different hyperparameters for each algorithm and performed hyperparameter tuning using Grid Search and Bayesian Optimization to achieve the best possible performance. More information regarding hyperparameters corresponding to models is given in experiment and result section.

During the training phase, we also applied different techniques to address the problem of overfitting, such as cross-validation and feature selection. We made sure that the models were not overfitting by monitoring the training and testing accuracy scores and adjusting the hyperparameters accordingly.

After training the models, we saved the best performing model to predict the leaf species for the unseen data in the evaluation phase.

4) Model Evaluation

This step involves calculating different performance metrics such as accuracy, precision, recall, and F1-score. We will also use classification report to get an in-depth understanding of our models' performance. We will first evaluate each model individually, comparing their performance on the test set. Finally, we will summarize our results and provide insights into the strengths and limitations of our selected model. This will help us to draw conclusions and make recommendations for future work on this problem.

V. Experiment and Results

Before the training of our classifiers on input data, we preprocessed input data using Principal Component Analysis (PCA). The PCA algorithm is dimensionality reduction technique which reduces the dimensionality of the input data by projecting it onto a lower-dimensional space. For “n_components”, we have assigned “n_components='mle'”. This automatically choose the number of principal components. determined by the MLE (maximum likelihood estimation) method. We have passed “svd_solver='full'“ indicates that PCA should use the full SVD solver to compute the principal components. This helps in avoiding the overfitting.

Next, we have splitted our input data into training, validation, and testing set. Out of 100%, we have taken 64% for training, 16% for validation and 20% for testing. Then we trained our classifiers using training dataset. We performed the prediction on both validation set and testing set. We have done hyperparameter tuning using Bayesian Optimization, Grid search etc. The evaluation metrics we have used are 5 fold cross validation, F1 score, precision, recall, accuracy, and loss function.

Below are the results from the random forest, decision tree algorithm.

A. Random Forest

Below is the accuracy and log loss of random forest on validation data set. This random forest has given this accuracy without any optimization technique.

- Without “Bayesian Optimization”

```
Validation Accuracy : 86.7925%
Validation Log Loss: 1.775114226246516
```

Below is the output of 5 Fold cross validation. We have calculated the average accuracy of 5 fold cross validation which is close to the above validation accuracy. This ensures us that input dataset has very little biasing as there is not very much variation in the accuracies given by 5 folds.

```
print("Cross Validation Avg Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores

RandomForestClassifier() [0.88050314 0.89308176 0.88607595 0.84810127 0.86708861]
Cross Validation Avg Accuracy: 0.87 (+/- 0.03)
```

- With “Bayesian Optimization”

Now below are the outputs which model has given after we apply Bayesian Optimization technique.

For the hyperparameters, we chose the range of “n_estimators”, 'max_depth' and 'min_samples_split'. Below is the snapshot for the same.

```
# Define the search space
search_space = [space.Integer(10, 170, name='n_estimators'),
                space.Integer(2, 34, name='max_depth'),
                space.Integer(2, 12, name='min_samples_split')]
```

From the above ranges, Bayesian optimization gives the best hyperparameters with the F1 score. Below is the snapshot for the same.

```
  warnings.warn("The objective has beer
Best hyperparameters:  [170, 32, 2]
Best F1 score:  0.8269360269360269
```

Next, we predicted the test data using the above optimized model given by Bayesian optimization. Below is the accuracy we got. We can see that accuracy is increased from 86% to 91%.

```
print("BO Test data Accuracy: {:.

BO Test data Accuracy: 91.4141%
```

We also validated this optimized model using 5 fold cross validation. Below is the output for the same. In

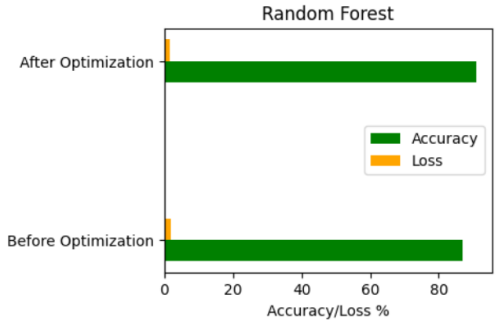
the below output we can see that model is actually optimized as it is now giving higher prediction accuracy than previous unoptimized model.

```
RandomForestClassifier(max_depth=32, n_estimators=170) [0.91194969 0.90566038 0.91139241 0.90506329 0.90506329]
Cross Validation Avg Accuracy: 0.91 (+/- 0.01)
```

Below is the output showcasing precision, recall, f1-score and support respectively. This is just a cropped version of classification report. Full report is present in .ipynb file.

viburnum_tinus	1.00	1.00	1.00	2
tidophylloides	0.67	1.00	0.80	2
elkova_Serrata	1.00	0.50	0.67	2
accuracy			0.89	198
macro avg	0.91	0.89	0.88	198
weighted avg	0.91	0.89	0.88	198

From the experimentation of random forest, we can conclude that we got the final accuracy of 91%. Below is the plot of accuracy and loss for random forest (Before optimization and after optimization).



B. Decision Tree

Decision tree algorithm has not given good prediction accuracy on validation set or test set. We tried multiple optimization approaches like fitting the tree with different depths and check the best value using the validation set or by iterating through the range of samples used for splitting a node. The classifier showed little significant change in results.

Below is the accuracy and log loss of decision tree on validation data set.

```
print("Validation Log Loss: {}".format(l

Validation Accuracy: 42.7673%
Validation Log Loss: 20.628757600060784
```

Below is the F1 score for training and testing.

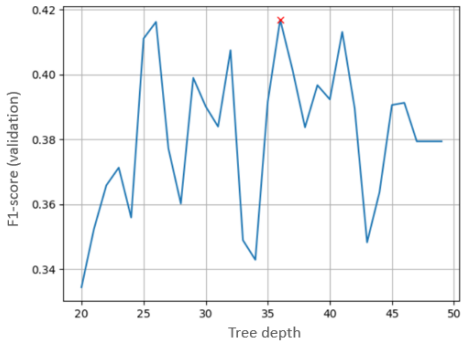
```
f1scoretest = f1_score( y_val, predictions_val_DT , aver

print("F1-score for training:{}, for test:{}".format(f1s

F1-score for training:1.0, for test:0.37123617123617125
```

Then we validated the model performance/results using 5 fold cross validation. Below is the output of the same. As we can see from the results of 5 folds that accuracy of model is increasing upto 48%.

```
DecisionTreeClassifier() [0.51572327 0.43396226 0.48101266 0.51265823 0.44303797]
Cross Validation Avg Accuracy: 0.48 (+/- 0.07)
```

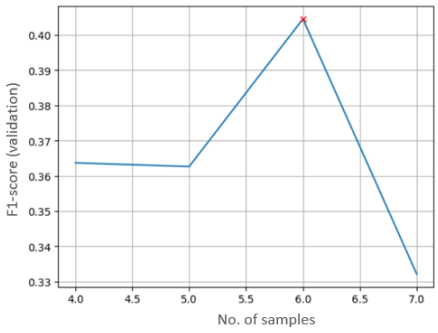
- Fit the tree with different depths

So, we applied optimization approaches on classifier. First we fit the tree with different depths and check the best value using the validation set. Below plot shows the F1 score for different depth of tree. Depth range from 20 to 50, within this range maximum F1 score achieved is 0.41. When depth > 46, F1 score remains constant on 0.38. So, changing the depth of tree increased the F1 score from 37% to 41% . This is because with the change in depth of tree, the effect of overfitting is reduced.



- Effect of number of samples for spliting a node

Next, we apply one more approach to optimize the decision tree classifier. We tried with range of samples used for splitting a node. There are 8 sample for each class in our training set. So, we passed the range “np.arange(4,8)” to “min_samples_split”. Below is the output plot for the same operation. In the below plot we can see that for the 6 number of samples, F1 score is highest. Otherwise low or decreasing. But there is no increase in value of F1 score than previously.



Below is the output showcasing precision, recall, f1-score and support respectively. This is just a cropped version of classification report. Full report is present in .ipynb file.

ytidophylloides	0.00	0.00	0.00	1
Zelkova_Serrata	1.00	0.50	0.67	2
accuracy			0.43	159
macro avg	0.36	0.44	0.37	159
weighted avg	0.38	0.43	0.37	159

Hence, decision tree algorithm does not provide good prediction accuracy on the validation data set. Its final accuracy after the optimization is 48%. Below is the plot of accuracy and loss for KNN (Before optimization and after optimization).

C. K-Nearest Neighbors algorithm (KNN)

k-nearest neighbors' algorithm has given good prediction accuracy on validation set or test set. For more accurate results and to reduce overfitting/underfitting, we employed hyperparameter tuning method named "Grid Search" to fit the model with best parameters. The sequence of experimentation steps/outputs is as follows.

- Without "Grid Search"

Below is the output of accuracy and Log Loss of KNN with default parameters.

```
ll_test_KNN = log_loss(y_val
print("Log Loss: {}".format(
```

```
Accuracy: 82.3899%
Log Loss: 2.282210142592443
```

Below is the output of 5 fold cross validation score.

```
print( cross validation Avg Accuracy: %0.2f (+/- %0.2f) % (scores.mean(), scores.std() 2))
```

```
KNeighborsClassifier(n_neighbors=3) [0.86163522 0.79874214 0.87341772 0.79113924 0.89240506]
Cross Validation Avg Accuracy: 0.84 (+/- 0.08)
```

- With "Grid Search"

Below is the output of best parameter (Neighbors) and best score calculated by GridSearchCV.

```
# print(grid.best_params_)
print(grid.best_score_)
```

```
{'n_neighbors': 1}
0.8616352201257862
```

Below is the output of accuracy given by KNN classifier on test data prediction.

```
int("G Test data Accuracy: {:.4%}"
```

```
) Test data Accuracy: 90.9091%
```

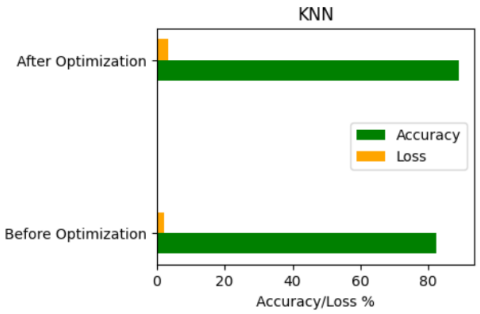
Below is the output of 5 fold cross validation score of classifier using Grid Search

```
Cross-validation scores: [0.91823899 0.87421384 0.89873418 0.84810127 0.91772152]
Average cross-validation score: 0.8914019584427992
```

Below is the output showcasing precision, recall, f1-score and support respectively. This is just a cropped version of classification report. Full report is present in .ipynb file.

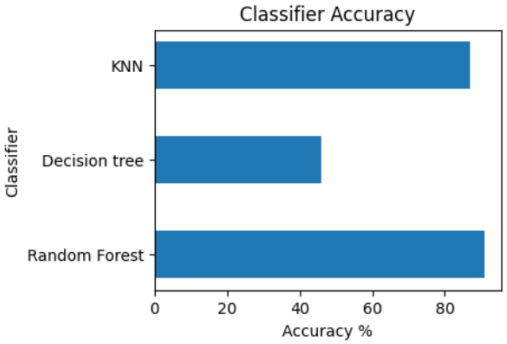
viburnum_rinus	1.00	1.00	1.00	2
ytidophylloides	1.00	1.00	1.00	2
Zelkova_Serrata	1.00	1.00	1.00	2
accuracy			0.91	198
macro avg	0.93	0.91	0.90	198
weighted avg	0.93	0.91	0.90	198

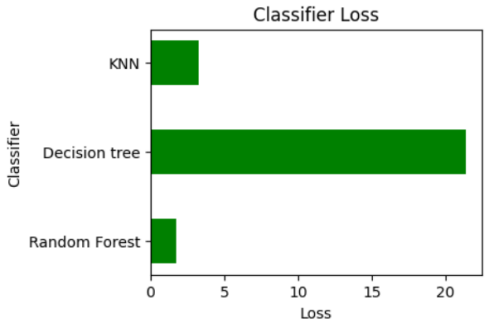
From the results of KNN, specially from cross validation score, we can conclude that GridSearchCV has increase the accuracy of KNN classifier from 84% to 89%. Below is the plot of accuracy and loss for KNN (Before optimization and after optimization).



D. Results

Below figures show final accuracy and loss of each classifier.





VI. Limitations of this research

While this study provided promising results in the classification of the dataset using machine learning techniques, there is still room for advancement in terms of increasing the accuracy of the models. The use of other techniques for dimensionality reduction, such as t-SNE or LDA, could be explored in future work. Additionally, it would be beneficial to explore a wider range of hyperparameters and apply more advanced optimization techniques, such as gradient-based optimization, to further enhance the performance of the models. Furthermore, it would be useful to investigate the performance of other algorithms such as neural networks, support vector machines, and ensemble methods. Finally, it would be interesting to apply these models to a larger and more diverse dataset to validate their effectiveness in other contexts.

VII. Conclusion

In this study, we aimed to improve the accuracy of three classification algorithms, namely Random Forest, KNN, and Decision Tree. We employed several techniques including PCA for dimensionality reduction, Bayesian optimization, hyperparameter tuning, and grid search to optimize the algorithms. The results showed that the Random Forest algorithm had the highest accuracy of 91%, followed by KNN with an accuracy of 89%, and Decision Tree with an accuracy of 48%.

The use of PCA was effective in reducing the number of features, and thus increasing the efficiency of the algorithms. Bayesian optimization and hyperparameter tuning allowed us to find the best set of hyperparameters for the algorithms, which ultimately increased their accuracy. The use of a validation set and a test set helped us avoid overfitting, and the 5-fold cross-validation approach allowed us to assess the generalizability of our models.

Overall, our study demonstrates that employing various techniques such as PCA, Bayesian optimization, and hyperparameter tuning can significantly improve the performance of classification algorithms. The findings of this study have useful/practical implications for plant classification and could potentially be applied in various fields of plant biology and ecology.

VIII. References

[1] Munisami, T., Ramsurn, M., Kishnah, S., Pudaruth, S. (2015). Plant leaf recognition using shape features and colour histogram with K-nearest neighbour classifiers. Procedia Computer Science, 58: 740-747. https://doi.org/10.1016/j.procs.2015.08.095

[2] Naresh, Y.G., Nagendraswamy, H.S. (2016). Classification of medicinal plants: An approach using modified LBP with symbolic representation. Neurocomputing, 173: 1789-1797. https://doi.org/10.1016/j.neucom.2015.08.090

[3] Yang, C., Wei, H., Yu, Q. (2016). Multiscale triangular centroid distance for shape-based plant leaf recognition. In ECAI, pp. 269-276. https://doi.org/10.3233/978-1- 61499-672-9-269

[4] Begue, A., Kowlessur, V., Mahomoodally, F., Singh, U., Pudaruth, S. (2017). Automatic recognition of medicinal plants using machine learning techniques. International Journal of Advanced Computer Science and Applications, 8(4): 166-175

[5] Kan, H.X., Jin, L., Zhou, F.L. (2017). Classification of medicinal plant leaf image based on multi-feature extraction. Pattern Recognition and Image Analysis, 27(3): 581-587. https://doi.org/10.1134/S105466181703018X

[6] Ali, R., Hardie, R., Essa, A. (2018). A leaf recognition approach to plant classification using machine learning. In NAECON 2018-IEEE National Aerospace and Electronics Conference, Dayton, OH, USA, pp. 431-434. https://doi.org/10.1109/NAECON.2018.8556785

[7] Salve, P., Sardesai, M., Yannawar, P. (2018). Classification of plants using GIST and LBP score level fusion. In International Symposium on Signal Processing and Intelligent Recognition Systems, pp. 15-29. https://doi.org/10.1007/978-981-13-5758-9_2

[8] Mostajer Kheirkhah, F., Asghari, H. (2019). Plant leaf classification using GIST texture features. IET Computer Vision, 13(4): 369-375. https://doi.org/10.1049/ietcvi.2018.5028

[9] Pacifico, L.D., Britto, L.F., Oliveira, E.G., Ludermir, T.B. (2019). Automatic classification of medicinal plant species based on color and texture features. In 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), Salvador, Brazil, pp. 741-746. https://doi.org/10.1109/BRACIS.2019.00133

[10] Sujith, A., Neethu, R. (2021). Classification of plant leaf using shape and texture features. In Inventive Communication and Computational Technologies, pp. 269-282. https://doi.org/10.1007/978-981-15-7345-3_22

[11] Z.-Q. Zhao, B.-J. Xie, Y.-m. Cheung, X. Wu, Plant leaf identification via a growing convolution neural network with progressive sample learning, in: Asian Conference on Computer Vision, 2014, pp. 348–361.

[12] Z. Liu, L. Zhu, X.-P. Zhang, X. Zhou, L. Shang, Z.-K. Huang, Y. Gan, Hybrid deep learning for plant leaves classification, in: International Conference on Intelligent Computing, Springer, 2015, pp. 115–123.

[13] S. Wu, F. Bao, E. Xu, Y. Wang, Y. Chang, Q. Xiang, A leaf recognition algorithm for plant classification using probabilistic neural network, in 2007 IEEE International Symposium on Signal Processing and Information Technology (2007)

[14] G. Saleem, M. Akhtar, N. Ahmed, W. Qureshi, Automated analysis of visual leaf shape features for plant classification. Comput Electron Agric 157, 270–280 (2019)

[15] Mallah C, Cope J, Orwell J. Plant leaf classification using probabilistic integration of shape, texture and margin features. Signal Processing, Pattern Recognition and Applications. 2013 Feb;5(1):45-54.

[16] https://www.javatpoint.com/machine-learning-random-forest-algorithm

[17] https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

[18] https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[19] https://scikit-learn.org/stable/modules/cross_validation.html

[20] https://www.nature.com/articles/nmeth.4346