Mushroom Classification

Sai Ganesh Pendela
Maryland Applied Graduate Engineering
University of Maryland
Collge Park, MD, United States
spendela@umd.edu

***Abstract*—This paper explores the application of machine learning algorithms in classifying mushrooms as edible or poisonous. Five different machine learning algorithms are employed namely K-Nearest Neighbors, Logistic Regression, Decision Tree, Naive Bayes, and Linear Discriminant Analysis, their results are compared based on accuracy, recall, precision, f1 score, cross validation scores and area under the Receiver Operating Characteristic curve. Results indicate promising accuracy across all models, with K-Nearest Neighbours exhibiting remarkable accuracy and performance, providing a strong foundation for enhancing mushroom consumption safety..**

***Index Terms*—Mushroom dataset, Logistic Regression, Naive Bayes, Linear Discriminant Analysis, Decision Tree, K-Nearest Neighbors, accuracy, recall, precision, f1 score, cross validation, Receiver Operating Characteristic curve .**

I. Introduction

The decision problem focuses on the classification of mushrooms into edible or poisonous categories. In this paper, the Mushroom dataset is used to train and test different machine learning algorithms.

The Mushroom dataset [1] comprises of 8214 instances with 23 features. The objective is to predict whether the mushroom is poisonous or edible based on the mushroom features. Here the dependent variable is ***Class*** and the independent variables are ***cap-shape, cap-surface, cap-color, bruises, odor, gill-attachment, gill-spacing, gill-size, gill-color, stalk-shape, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, stalk-color-below-ring, veil-type, veil-color, ring-number, ring-type, spore-print-color***.

The rest of the paper is organized as follows, Section II describes the Methodology to build Logistic Regression [2], Decision Tree [3], K-Nearest Neighbours [4], Naive Bayes [5], Linear Discriminant Analysis Classifier [6]. and it also describes how to calculate various classification metrics [7]. The Results section provides comparison of the results of the models. In the Conclusion section, possible reasons for the Results obtained are explained.

II. Methodology

In this section, the necessary libraries required for the classification, the steps to load the dataset, preprocessing of the data, building of the different models i.e, Logistic Regression, Decision Tree, K-Nearest Neighbours, Naive Bayes, Linear Discriminant Analysis Classifiers on the Mushroom dataset is explained. Also, the functions used to calculate the metrics is discussed.

A. Setting up and importing necessary packages

The Machine Learning models are built using **Google Colab** [8]. The exact versions of them are mentioned in table I

Software	Version
Google Colab	Colab Pro

TABLE I
SOFTWARE AND THEIR VERSIONS

Other than Google Colab, any version of Python 3.8.* should work, but the exact version should be preferred to avoid any issues.

Other than the softwares mentioned, various other packages of Python are needed to ease the development process of the Classifiers. All the packages that are needed are mentioned in the table II.

Module	Version
Numpy [9]	1.24.4
Scikit-Learn [10]	1.3.0
Pandas [11]	2.0.3
Matplotlib [12]	3.7.2

TABLE II
MODULES USED AND THEIR VERSIONS

All the above mentioned packages can be imported using the Python syntax defined.

B. Loading the Mushroom dataset

The Mushroom dataset is readily available for download in CSV format from Kaggle. Once downloaded, you can import the dataset into Google colab as a DataFrame [13] using the Pandas *read_csv()* [14] function. This function accepts the *file path* as an input and returns a DataFrame object. Utilizing a DataFrame to load the data is advantageous because it enables us to access the dataset's columns individually, unlike the conventional row-wise approach commonly used by most software. The loading of the Mushroom dataset is same as

loading the iris data set mentioned in the previuos paper **Decision Tree Classifier** [15].

C. Data Preprocessing

Before building the models, the dataset is checked for any null values using the ***isnull()*** [16] function. The results show that the mushroom dataset does not contain any null values. After checking for null values, all the categorical columns are converted into numerical columns using the ***LabeEncoder()*** [17] function. The Label encoding is done to convert categorical data into numerical format, assigning unique integers to each category, enabling machine learning models to interpret and process the data.

After label encoding one hot encoding is performed on the categorical columns using ***get_dummies()*** [18] function. One-hot encoding is used to represent categorical variables numerically without introducing ordinal relationships, ensuring independence between categories in machine learning models. After performing one hot encoding, standaradization is done using ***StandaradScaler()*** [19] function. This function transforms the data to have a mean of 0 and a standard deviation of 1, making features comparable and suitable for certain machine learning algorithms.

By this time that dataframe has **5686** rows and **95** columns, to reduce the columns, dimensionality reduction is done using the ***PCA()*** [20] function with number of components ***n_components*** [21] as **2**. Dimensionality reduction reduces the number of columns by capturing the most important information in the data while reducing its complexity

D. Data partitioning

Data partitioning is done to separate a dataset into distinct subsets for training and testing machine learning models, enabling model evaluation on unseen data and assessing generalization performance. To perform data partitioning, the ***train_test_split()*** [22] function from Scikit-Learn is used. The function takes *data*, *ground truth* and *test_size* as input to return the *X_train*, *X_test*, *y_train* and *y_test*.

Data partitioning results in four variables:

X train (for training data), X test (for testing data), y train (ground truths for training), and y test (ground truths for evaluation).

A *30%* test split is chosen, offering a suitable number of samples for both training and testing, such as *100* training samples and *30* testing samples in the case of the Iris dataset.

Following the data split using these variables, you can build machine learning models, and perform model evaluation. The above process is same as mentioned in the previous paper **Decision Tree Classifier** [15].

E. Building models

Scikit-Learn [23]provides various classes and functions to build Machine learning models. Here a total of five models are built and their performance is evaluated.

To build a Logistic Regression model, ***LogisticRegression()*** [3] function in the Scikit-learn library is used. Logistic Regression is a linear classification algorithm used for binary and multiclass classification tasks. The steps to build the logistic regression model is same as mentioned in the paper **Understanding Logistic Regression** [24].

To build a decision tree model, ***DecisionTreeClassifier()*** [3] function in the Scikit-learn library is used. Decision Tree is a non-linear algorithm that recursively splits the data based on features to make decisions. The steps to build the decision tree model is same as mentioned in the paper **Decision Tree Classifier** [15]. Here, the criterion is set to Entropy and random_state is set to 42.

To build a K-Nearest Neighbour model, ***KNeighborsClassifier()*** [4] function in the Scikit-learn library is used. K-Nearest Neighbors is a non-parametric algorithm that classifies data points based on the majority class among their k-nearest neighbors. The steps to build KNN is same as mentioned in the paper **Nearest Neighbour Algorithm** [25]. Here, the number of neighbours is 5 which is a default value.

To build the Naive Bayes model, the ***GaussianNB()*** [5] is imported form the sklearn.naive_bayes module, the GaussianNB refers to the Gaussian Naive Bayes classifier, it a variant of the Naive Bayes algorithm that assumes the likelihood of the features is Gaussian (normal distribution). After importing the naive bayes classifier, an instance of the classifier is built, after that the model is fit to the training data. After training the model, predicitions are made on the test data.

Similarly, the LinearDiscriminantAnalysis (LDA) [6] is a classifier under the sklearn.discriminant_analysis module. LDA is a dimensionality reduction technique commonly used in the field of pattern recognition and machine learning. It's often used as a classification algorithm, especially in scenarios where the classes are well-separated and follow a normal distribution.

F. Evaluating the Performance

The *sklearn.metrics* is a module in the scikit-learn (sklearn) library, which is a popular machine learning library in Python. This module contains various functions and classes for evaluating the performance of machine learning models. The ***confusion_matrix()*** **[26],** ***accuracy_score()*** **[27],** ***classification_report()*** **[28],*recall_score()*** **[29],** ***F1 Score()*** **[30],** ***precision_score()*** [33] these functions should be imported to calculate the accuracy, recall, precision and to view the confusion matrix and classification report.

All the description and how to calculate these metrics is given in the previous paper **Classification Metrics** [7].

G. Generating ROC curves and comparing AUC metrics

ROC (Receiver Operating Characteristic) [31] curves are graphical representations of a binary classification model's performance. They illustrate the trade-off between a model's True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds. To calculate an ROC curve in Python, you can use libraries like scikit-learn.

The steps to build the ROC curve and find the area under the ROC curve is described in the previous paper ***Evaluation Metrics*** [32].

RESULTS

A. Comparison based on classification metrics

The performance of various classification models on the mushroom dataset is assessed based on metrics such as accuracy, precision, recall, and F1 score. K-Nearest Neighbors demonstrates robust classification with an accuracy of **92.12%**, showcasing a balanced trade-off between precision **95.49%** and recall **87.89%**. Logistic Regression also performs well, achieving an accuracy of **90.28%**, with a balanced precision **96.18%** and recall **83.23%**. Decision Tree, with an accuracy of **89%**, exhibits high precision **88.78%** and recall **89.08%**. Naive Bayes maintains competitive accuracy at **89.75%**, with balanced precision **96.13%** and recall **82.13%**. Linear Discriminant Analysis achieves an accuracy of **88.11%**, emphasizing higher precision **97.65%** over recall **77.31%**. The variability in model performance highlights the importance of considering specific classification goals and trade-offs when selecting an appropriate model for mushroom classification.

Model	Accuracy	Recall	Precision
Decision Tree	0.89	0.89	0.89
Logistic Regression	0.90	0.83	0.96
K-Nearest Neighbors	0.92	0.87	0.95
Naive Bayes	0.89	0.82	0.96
Linear Discriminant Analysis	0.88	0.77	0.97

TABLE III
PERFORMANCE OF ALL THE MODEL'S

K-Nearest Neighbors (KNN) performed well in mushroom classification due to its ability to capture the local structure of the data. In the context of mushroom classification, where the characteristics of edible and poisonous mushrooms may form distinct groups, KNN can accurately identify the class based on the similarity of features. So this is the reason why KNN performed well when compared to other algorithms.

B. Comparison based on cross validation results

The cross-validation results reveal the performance of various models in mushroom classification. K-Nearest Neighbors (KNN) exhibits the highest average accuracy at **92.82%**, demonstrating its robust predictive capabilities with low variability **1.08%** standard deviation. Logistic Regression achieves a solid accuracy of **90.66%** with low deviation **1.03%**, making it a reliable classifier. The Decision Tree performs well with an accuracy of **89.04%** and stable results **1.12%** standard deviation. Naive Bayes shows competitive performance average accuracy of **89.87%**, standard deviation of **1.13%**, while Linear Discriminant Analysis (LDA) provides respectable accuracy of **88.96%** with low variability **1.06%**. The choice of k in KNN likely contributes to its superior accuracy, making it a standout model for mushroom classification, although other models also exhibit strong and consistent performance.

Model	Average Accuracy	Standard Deviation
Decision Tree	0.89	0.0112
Logistic Regression	0.90	0.0103
K-Nearest Neighbors	0.92	0.0108
Naive Bayes	0.89	0.0113
Linear Discriminant Analysis	0.88	0.0106

TABLE IV
CROSS VALIDATION RESULTS OF ALL THE MODEL'S

The higher average accuracy of K-Nearest Neighbors (KNN) in mushroom classification can be attributed to its ability to capture intricate patterns and local clusters effectively. Given the nature of mushroom characteristics, where the proximity and similarity of features play a crucial role in classification, KNN excels by considering the nearest neighbors. This is especially relevant in a dataset where the relationships between different mushroom types might be well-defined within local regions. The model's performance is further substantiated by its low standard deviation, indicating consistency and reliability across different cross-validation folds.

C. Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) areas for the mushroom classification models provide valuable insights into their discriminatory power. K-Nearest Neighbors (KNN) leads with the highest ROC area of **0.92**, indicating its superior ability to distinguish between edible and poisonous mushrooms. Logistic Regression (LR) and Naive Bayes (NB) closely follow with ROC areas of **0.90**, highlighting their strong discriminatory capabilities. Decision Tree (DT) achieves a respectable ROC area of **0.89**, emphasizing its effectiveness in capturing intricate patterns within the data. Linear Discriminant Analysis (LDA) exhibits a slightly lower ROC area of **0.88**, suggesting a nuanced performance in discriminatory tasks. The variations in ROC areas align with the models' inherent characteristics, where KNN's local neighborhood approach excels in capturing subtle distinctions, while LR and NB leverage probabilistic frameworks. DT's performance is commendable, showcasing its ability to create discerning decision boundaries. LDA, with a focus on linear separation, may experience challenges in capturing more complex relationships, as reflected in its ROC area.

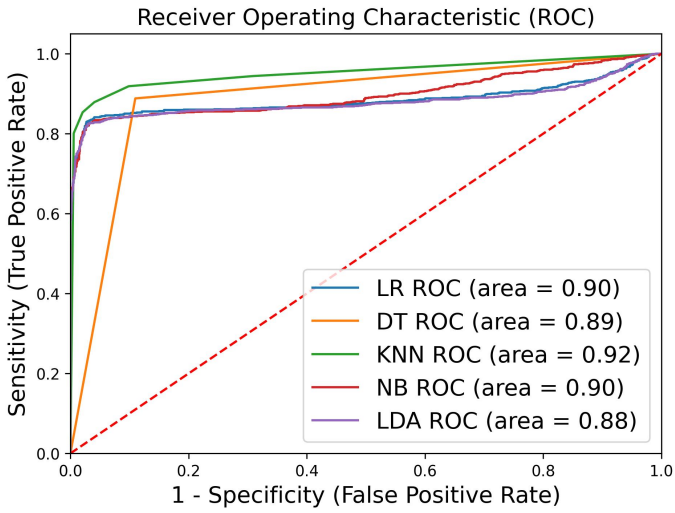

Fig. 1. Receiver Operating Characteristic Curve.

DISCUSSION

The results of the mushroom classification models offer valuable insights into their performance and significance. K-Nearest Neighbors (KNN) emerges as the top-performing model when compared with other four algorithms, with a remarkable accuracy of **92.12%** and with an average accuracy of **92.82%** and a very low standarad deviation of **0.0108**. This aligns with expectations, given KNN's ability to capture local patterns and clusters effectively. KNN can accurately identify the class based on the similarity of features. In the context of mushroom classification, where the characteristics of edible and poisonous mushrooms may form distinct groups, so this is the reason why KNN performed well when compared to other algorithms.

This work is important as it addresses the crucial task of mushroom classification, providing a reliable means to distinguish between edible and poisonous varieties. Accurate classification is vital for ensuring the safety of individuals consuming wild mushrooms, preventing potential health hazards. The use of machine learning models in this context streamlines the identification process, offering a quick and efficient solution.

In the future, potential improvements to this work include exploring additional features, employing advanced feature engineering techniques, and optimizing models through ensemble methods and hyperparameter tuning. Expanding the dataset to encompass a broader range of mushroom species and attributes could enhance model diversity. Additionally, incorporating deep learning approaches or more sophisticated algorithms may reveal intricate patterns not captured by traditional machine learning models. These continuous refinements contribute to the ongoing enhancement of mushroom classification systems, improving their reliability and real-world applicability.

REFERENCES

[1] https://www.kaggle.com/datasets/uciml/mushroom-classification
[2] https://en.wikipedia.org/wiki/Logistic_regression
[3] https://en.wikipedia.org/wiki/Decision_tree
[4] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[5] https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[6] https://en.wikipedia.org/wiki/Linear_discriminant_analysis
[7] https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[8] https://colab.research.google.com/?utm_source=scs-index
[9] https://numpy.org/doc/stable/
[10] https://scikit-learn.org/stable/tutorial/index.html
[11] https://pandas.pydata.org/docs/getting_started/index.html
[12] https://matplotlib.org/stable/tutorials/pyplot.html
[13] https://www.geeksforgeeks.org/python-pandas-dataframe/
[14] https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/.
[15] https://github.com/ganeshpendela/ENPM808L/blob/master/All%20papers/ENPM808L_HW02.pdf
[16] https://www.w3schools.com/python/pandas/ref_df_isnull.asp
[17] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[18] https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html
[19] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[20] https://en.wikipedia.org/wiki/Principal_component_analysis
[21] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[22] https://www.geeksforgeeks.org/how-to-split-the-dataset-with-scikit-learns-train_test_split-function/
[23] https://scikit-learn.org/stable/
[24] https://github.com/ganeshpendela/ENPM808L/blob/master/All%20papers/Pendela%2C%20Sai%20Ganesh_Week3.pdf
[25] https://github.com/ganeshpendela/ENPM808L/blob/master/All%20papers/Pendela%2C%20Sai%20Ganesh_Report_Week%205.pdf
[26] https://www.w3schools.com/python/python_ml_confusion_matrix.asp
[27] https://www.javatpoint.com/accuracy_score-in-sklearn
[28] https://www.statology.org/sklearn-classification-report/
[29] https://en.wikipedia.org/wiki/Precision_and_recall
[30] https://www.v7labs.com/blog/f1-score-guide
[31] https://en.wikipedia.org/wiki/Receiver_operating_characteristic
[32] https://github.com/ganeshpendela/ENPM808L/blob/master/All%20papers/PendelaSai_Ganesh_Report_Week_7.pdf
[33] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html