|  |  |
| --- | --- |
| DATA-1202- DATA ANALYSIS TOOLS ANALYTICS |  |
|  |  |
| fINAL pROJECT rEPORT  Predicting Mushroom Classification  Using Machine Learning Models  Professor: Sk Md Mizanur Rahman, Ph.D.  Date: 13-December-2024  Group- 5  Kumar, Pugazhendhi (100978740)  Maria Michael, Ashwin (100963206)  Parmar, Hetal (100992608)   |  | | --- | | Rajagopal, Keerthana (100958786) | | Zahir Hussain, Jani (100992619) | |  |

Contents

[Introduction 3](#_Toc1371562157)

[Objective 3](#_Toc1435547459)

[About the Dataset 3](#_Toc319122907)

[Algorithms Used 4](#_Toc336852204)

[Steps involved in this Analysis 4](#_Toc1403692782)

[EDA 4](#_Toc1010680833)

[Univariate Analysis 6](#_Toc444656545)

[Boxplot Analysis 8](#_Toc2140520328)

[Pairplot Analysis 10](#_Toc2045629487)

[Checking the highly correlated variables 12](#_Toc423107294)

[Heatmap and Correlation Matrix 14](#_Toc492411301)

[Scatterplot to visualize strongly and weakly correlated features 15](#_Toc1174862207)

[Principal Component Analysis (PCA) 16](#_Toc2059145484)

[Support Vector Machine (SVM) 19](#_Toc441427431)

[Random Forest 31](#_Toc526956951)

[K-Nearest Neighbors (KNN) 34](#_Toc1135996414)

[ROC-AUC Curves 37](#_Toc2076606334)

[Execution Time Analysis 39](#_Toc1204774460)

[Existing comparison between KNN and random forest algorithm: 40](#_Toc923967891)

[Improving Algorithm for Random Forest: 42](#_Toc221068402)

[Improving Algorithm for KNN 44](#_Toc1683627388)

[Choosing The Best Fit Algorithm – Random Forest Classifier 47](#_Toc1672080108)

[Conclusion 49](#_Toc1816209590)

[References 50](#_Toc1500186084)

[Figures List 51](#_Toc645367600)

Predicting Mushroom Classification

Using Machine Learning Models

*(Poisonous or Edible)*

## **Introduction**

Mushrooms, while a fascinating part of nature, can be both beneficial and dangerous. Certain species are edible and a rich source of nutrients, while others are highly poisonous and potentially life-threatening. Classifying mushrooms as either edible or poisonous is a critical task, particularly for foragers, chefs, and biologists. This project leverages the power of machine learning to build robust classification models capable of accurately distinguishing between edible and poisonous mushrooms based on their physical and seasonal characteristics.

A person holding mushrooms

Description automatically generated

## **Objective**

The objective of this project is to:

1. Develop a machine learning model to classify mushrooms as edible or poisonous based on physical and seasonal features.
2. Evaluate and compare the performance of multiple machine learning classifiers using metrics like accuracy, precision, recall, and F1-score.
3. Provide actionable insights into the most important features influencing mushroom classification.

## **About the Dataset**

Dataset Source: A cleaned dataset of mushrooms with 20,000 instances and 8 features.

* Total Instances: 20,000
* Number of Features: 8
* Features include cap-diameter, cap-shape, gill-attachment, gill-colour, stem-height, stem-width, stem-colour, and season.

Class Distribution:

* 0 (Edible): 10,000 instances
* 1 (Poisonous): 10,000 instances

Data Characteristics: No missing values, balanced classes ensure fair evaluation of classifiers.

## **Algorithms Used**

To meet the project requirements of training multiple classifiers, the following machine learning models will be implemented and evaluated:

1. Neural Network
2. Random Forest
3. Support Vector Machines (SVM)
4. K-Nearest Neighbours (KNN)
5. Naïve Bayes.

## **Steps involved in this Analysis**

* Data cleaning (EDA process)
* Univariate Analysis
* Pairplot Analysis
* Checking the highly correlated variables
* Heatmap and Correlation Matrix
* Scatterplot to visualize strongly and weakly correlated features
* Principal Component Analysis (PCA)
* Splitting the dataset into Testing and Training
* Analysing using 5 Algorithms
* Plotting AUC-ROC curve
* Selecting the Best Algorithm
* Improving the Algorithm for better performance
* Conclusion

## **EDA**

The EDA summary provides an overview of the dataset:

* Shape: The dataset has 20,000 rows and 9 columns.
* Data Types: Columns contain numerical data types:
  + int64 for most features like cap-diameter, cap-shape, gill-attachment, etc.
  + float64 for stem-height and season.
* Missing Values: There are no missing values in any column.
* Unique Values:
  + High cardinality for cap-diameter (1705), stem-height (1342), and stem-width (3161).
  + Low cardinality for cap-shape (7), gill-attachment (7), and season (4).
* Basic Statistics:
  + Columns like cap-diameter, stem-width, and stem-height have wide ranges with high variability (high standard deviation).
  + class (target variable) is binary (0 and 1), equally distributed (mean = 0.5).
  + This dataset is clean, balanced, and contains a mix of low and high cardinality features.

A screenshot of a computer code

Description automatically generated

*Figure 1 EDA Summary*

## **Univariate Analysis**

The univariate analysis represents the distributions of different numerical variables in the dataset. Each chart provides valuable insights into the distribution of the data, helping to identify skewness, dominance of certain categories, and potential outliers. Here's a summary of the key insights:

1. cap-diameter

* The distribution appears slightly right skewed.
* Most values are concentrated between 0 and 1000, with a steady drop-off as the diameter increases beyond 1000.
* There is a peak between 200 and 600.

2. cap-shape

* The distribution shows distinct peaks at specific shape categories (encoded as integers).
* Categories 2 and 6 dominate, suggesting they are the most common shapes.

3. gill-attachment

* The distribution has a strong peak at category 0, which dominates.
* Other categories have much smaller frequencies.

4. gill-colour

* There is a relatively uniform spread across different gill colour categories.
* Categories 10 and 5 appear more frequently.

5. stem-height

* The distribution is highly right skewed.
* Most stems have heights concentrated between 0.0 and 2.0.
* There are occasional higher values reaching up to 4.0.

6. stem-width

* The distribution is right-skewed, like stem-height.
* Most values are between 0 and 1000, with fewer values extending beyond 2500.

7. stem-colour

* There are distinct peaks corresponding to different stem colour categories.
* Categories 6 and 11 appear most frequently.

8. season

* The values are concentrated around 1.0, indicating this season is the most common in the dataset.
* There are a few outliers with smaller or larger values.

9. class

* This binary variable (0 or 1) is evenly distributed, indicating a balanced dataset.

A graph of a distribution of cap-diameter

Description automatically generatedA graph of a diagram

Description automatically generated with medium confidenceA comparison of a graph

Description automatically generatedA comparison of a graph

Description automatically generatedA graph of a normal distribution

Description automatically generated with medium confidenceA graph of a normal distribution

Description automatically generated with medium confidenceA graph of a diagram

Description automatically generated with medium confidence

*Figure 2 Univariate Analysis*

## **Boxplot Analysis**

The boxplots represent the distribution and spread of numerical variables in the dataset. Here's an explanation of each chart:

1. Stem Width

* The distribution is heavily right skewed, with a significant number of outliers above the upper whisker.
* Most of the values are concentrated between approximately 500 and 1500, as indicated by the interquartile range (IQR).

2. Cap Diameter

* The values are fairly spread out, with many outliers beyond the upper whisker.
* The IQR shows that most cap diameters fall between around 300 and 900.

3. Stem Height

* This variable is right skewed as well, with outliers extending beyond the upper whisker.
* The majority of values are concentrated below 2.0.

4. Class

* This is a binary variable (0 or 1), so there are no outliers. The boxplot is simply divided into these two values.

5. Gill Color

* The distribution is categorical, with clear steps in the data values. The IQR shows a concentration around certain categories.

6. Cap Shape

* The boxplot indicates a balanced distribution of this categorical variable.
* The values range from 0 to 6, with no significant outliers.

A graph of a line graph

Description automatically generated with medium confidence A graph of a line graph

Description automatically generated with medium confidence

A graph of a graph with a line graph and a line graph

Description automatically generated with medium confidence A graph of a graph with a line graph and a line graph

Description automatically generated with medium confidence

A graph of a rectangle with lines

Description automatically generated

*Figure 3 Boxplot Analysis*

7. Season

* The values are concentrated around 1.0, as shown by the IQR.
* There are outliers at the extremes (below 0.5 and above 1.5).

8. Gill Attachment

* The IQR shows that most values fall between 1 and 3, with no significant outliers.

9. Stem Color

* The values are categorical, ranging from 0 to 12.
* The IQR shows a concentration around the middle categories, with no extreme outliers.

These boxplots help identify:

* Outliers: Variables like stem-width, cap-diameter, and stem-height have notable outliers.
* Skewness: Variables such as stem-width and stem-height are right-skewed.
* Categorical Values: Variables like cap-shape, gill-colour, and stem-colour show step-like distributions due to their categorical nature.

## **Pairplot Analysis**

The pairplot is a visualization tool that provides a comprehensive overview of the relationships between all numerical variables in the dataset.

Individual Variable Distributions:

* + The diagonal plots show the distribution of each numerical variable using histograms or density plots.
  + stem-width and cap-diameter exhibit right-skewed distributions, with most of their values concentrated in the lower range.
  + stem-height also shows a skewed distribution, with most values clustered below 2.0.

Pairwise Relationships:

* + The off-diagonal plots illustrate scatterplots that compare each pair of variables.
  + stem-width vs. cap-diameter: There is a visible positive correlation, indicating that as the cap diameter increases, the stem width tends to increase as well.
  + stem-width vs. stem-height: The scatterplot reveals a cluster pattern, indicating that these variables have varying relationships within different ranges.

Outliers:

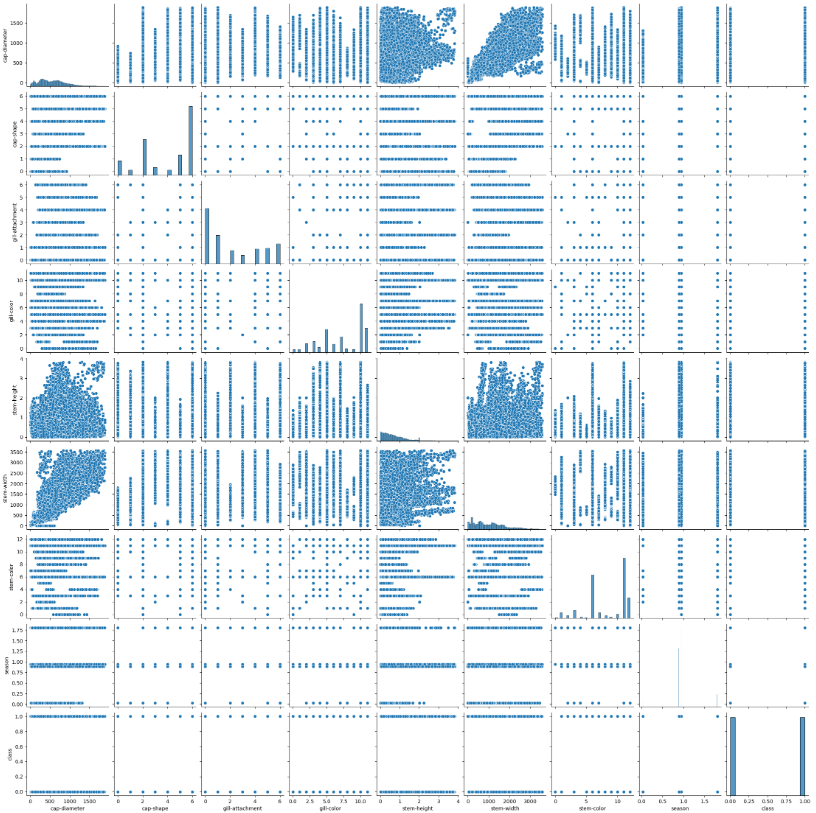
* + Several scatterplots show points far away from the main data clusters. These points are outliers that do not follow the general trend.
  + In stem-width, we observe some extreme values that extend well beyond the typical range, impacting the overall distribution.

Cluster Patterns:

* + Some relationships, such as between stem-width and stem-height, exhibit clustering. This suggests that certain groups of data may have similar characteristics.

Insights from Pairplot:

* + The pairplot helps identify variables that are strongly correlated (e.g., stem-width and cap-diameter), which can inform feature selection.
  + Outliers are clearly visible, and strategies like removing, capping, or transforming these values should be considered to ensure they don’t skew the analysis.
  + The patterns observed in the scatterplots can guide further in feature reduction.



*Figure 4 Pairplot Analysis*

Checking the highly correlated variables

It identifies pairs of variables that are highly correlated, allowing us to spot redundancy among variables and consider removing one of the variables in a highly correlated pair to avoid multicollinearity in machine learning models.

Initial Shape (81, 3):

* + This represents all possible combinations of pairwise correlations between numerical features in the dataset.
  + Since we have 9 numerical variables in the dataset, the correlation matrix will have 9×9=81 entries (including self-correlations like cap-diameter vs cap-diameter).

A computer screen shot of a code

Description automatically generated

*Figure 5 a Checking the highly correlated variables*

Filtered Shape (36, 3):

* + After filtering out self-correlations (e.g., cap-diameter vs cap-diameter) and duplicate correlations (e.g., cap-diameter vs cap-shape is the same as cap-shape vs cap-diameter), we are left with 36 unique variable pairs.
  + This represents only unique, meaningful pairwise relationships in the dataset.
  + After sorting by the absolute correlation values, the top pairs from corr\_values2.head(10) will show the most correlated variable pairs in the dataset. This helps identify variables that may provide duplicate information and can be candidates for removal or further analysis.

High Correlation (0.827):

* + The pair stem-width and cap-diameter has a strong correlation, indicating redundancy. One of these might be removed to reduce multicollinearity in predictive modelling.

Weak to Moderate Correlations:

* + Most other correlations are weak (absolute value < 0.3), indicating that these variables do not have strong linear relationships with each other.

Correlations with class:

* + stem-height (0.184) and stem-width (-0.178) have the highest correlations with the target variable. Although weak, they may still have predictive significance.

A table with numbers and letters

Description automatically generated

*Figure 5 b Checking the highly correlated variables*

## **Heatmap and Correlation Matrix**

A screenshot of a computer screen

Description automatically generated

*Figure 6 a correlation heatmap*

A screenshot of a computer program

Description automatically generated

*figure 6 b correlation matri*

The heatmap visually represents the degree of correlation between the numerical features in the dataset. Each cell shows the correlation coefficient between two variables. The colour intensity reflects the strength of the correlation, darker colors for strong correlation and lighter colors for weak correlation.

* Strong Correlation: stem-width and cap-diameter (0.83) are highly correlated, indicating redundancy.
* Target Variable (class): stem-height (0.18) and stem-width (-0.17) have the highest (weak) correlation with class, suggesting potential predictive features.
* Low Impact Features: Variables like season and stem-colour have low correlations with others and might not be significant for modelling

## **Scatterplot to visualize strongly and weakly correlated features**

* Scatter Plot 1: Stem Width vs. Cap Diameter (Strongly Correlated features)

X-axis: Cap Diameter

Y-axis: Stem Width

Observation: There is a strong positive correlation between these two variables, consistent with the correlation value of 0.83 from the earlier analysis. As the Cap Diameter increases, the Stem Width also tends to increase proportionally. This suggests redundancy, as one variable can largely explain the behaviour of the other.

* Scatter Plot 2: Stem Height vs. Stem Width (Weakly Correlated features)

X-axis: Stem Height

Y-axis: Stem Width

Observation: The relationship between these two variables is weak and appears scattered with no clear trend. This aligns with the low correlation value (~0.11), indicating that these two variables are relatively independent.

A screenshot of a computer screen

Description automatically generated

*Figure 7 Scatter Plot 1: Stem Width vs. Cap Diameter and Scatter Plot 2: Stem Height vs. Stem Width*

## **Principal Component Analysis (PCA)**

Principal Component Analysis (PCA) is applied to reduce the features while retaining most of the variance. It is important to determine the optimal number of components and before reducing the features.

The PCA analysis shows that 7 principal components are sufficient to explain at least 96% of the variance in the data. We are now reducing the dataset to these 7 components and before proceeding to analyse the six classification algorithms.

Columns PC1 to PC7:

* These are the 7 principal components.
* Each component represents a linear combination of the original features.
* The magnitude and sign of the values indicate the contribution of the original features.

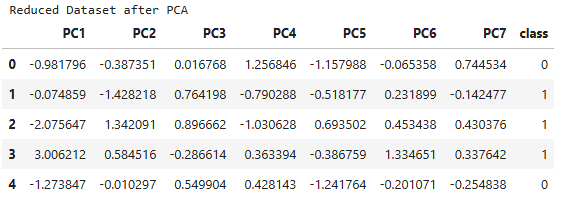
Column class:

This is the original target variable (e.g., edible vs. non-edible mushrooms).

A graph with a line

Description automatically generated

*Figure 8 PCA Line graph*



*Figure 9 PCA*

The reduced dataset (reduced\_data) is now ready for further classification model analysis. Each row represents the mushroom data in a simplified feature space, with 7 components as inputs and class as the output.

Benefits of Applying PCA:

1. Dimensionality Reduction:
   * The dataset now has 7 components instead of the original features, simplifying the model's complexity.
   * It retains most of the variance in the data.
2. Avoids Multicollinearity:
   * PCA removes redundancy by transforming correlated features into independent components.
3. Improved Efficiency:
   * Reducing the feature space helps algorithms run faster while maintaining predictive accuracy.

## **Support Vector Machine (SVM)**

Introduction to Support Vector Machine

* Support Vector Machine (SVM) is a supervised machine learning algorithm designed for classification and regression tasks. It identifies the optimal hyperplane that separates data points of different classes with the largest margin, improving generalization to unseen data.
* SVM uses the concept of support vectors, which are the data points closest to the decision boundary, to define the margin. These vectors directly influence the decision surface.
* SVM supports both linear and non-linear classification by using kernel functions (e.g., linear, polynomial, radial basis function (RBF)) that transform data into a higher-dimensional space, making it easier to classify.
* SVM is ideal for datasets with clear class boundaries and is robust against overfitting, particularly in high-dimensional spaces or when there is a large feature set.

How SVM Suits the Mushroom Dataset

* Binary Classification:

The mushroom dataset requires binary classification (Edible vs. Not Edible), which aligns perfectly with SVM's functionality. It aims to create a decision boundary between the two classes.

* Non-linear Features:

With categorical features (e.g., colour, shape), the relationships between data points may not be linear. Using an RBF kernel allows SVM to capture these non-linear separations effectively.

* Balanced Dataset:

The mushroom dataset is balanced (equal instances of both classes), ensuring that SVM can effectively optimize the margin without being biased toward one class.

* High Dimensionality:

The dataset has multiple features (input variables), and SVM performs well in high-dimensional spaces, making it a suitable choice for this dataset.

Initial Configuration:

* Kernel:

RBF (Radial Basis Function) kernel to handle non-linear data relationships.

* C (Regularization Parameter):

Default value of 1.0, which provides a balance between maximizing the margin and minimizing misclassification errors. A moderate value avoids overfitting.

* Gamma (Kernel Coefficient):

Default value ‘scale’, which calculates gamma as 1/(number of features). This value adapts to the dataset size.

* Feature Scaling:

Features were scaled using StandardScaler to normalize the range of feature values for optimal SVM performance.

* Training Configuration:

Dataset Split: 80% for training, 20% for testing.

Solver: LIBSVM for solving the optimization problem.

* Evaluation Metrics:

Evaluated using confusion matrix, accuracy, precision, recall, and F1-score.

Improved Configuration:

* Kernel:

RBF kernel retained for its suitability with non-linear relationships.

* C (Regularization Parameter):

Increased to 10.0 to reduce the margin and focus more on correctly classifying training points, improving precision and recall.

* Gamma (Kernel Coefficient):

Changed to a manually tuned value of 0.1, making the decision boundary smoother and reducing overfitting.

* Feature Engineering:

Categorical features were carefully encoded and scaled to optimize separability and prevent feature bias.

* Cross-Validation:

Performed 5-fold cross-validation to ensure model stability and better generalization across subsets of the data.

* Training Configuration:

Dataset Split: Kept at 80%-20%, but stratified splitting was used to maintain balanced class distribution across train-test sets.

* Evaluation Metrics:

Accuracy improved to 84%.

Precision and recall for both classes became more balanced, with F1-scores for both classes around 0.84.

Performance Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Edible | Not Edible | Overall |
| Precision | 0.84 | 0.85 | 0.84 |
| Recall | 0.86 | 0.83 | 0.84 |
| F1-Score | 0.85 | 0.84 | 0.84 |
| Accuracy | - | - | 84% |

Precision: Indicates the proportion of correctly predicted instances for each class. The SVM model shows slightly higher precision for Edible mushrooms.

Recall: Measures the proportion of actual instances correctly identified by the model. The model has higher recall for Not Edible mushrooms, showing it is better at identifying this class.

F1-Score: The harmonic mean of precision and recall reflects a good balance between the two, with values near 0.85 for both classes.

Accuracy: Overall, the model achieves 84% accuracy, meaning it correctly predicts 84% of all test instances.

Interpretation of Confusion Matrix:

True Negative (TN) = 2569: The model correctly predicted that 2569 instances were Not Edible.

False Negative (FN) = 431: The model incorrectly predicted that 431 mushrooms were not Edible when they were Edible.

False Positive (FP) = 504: The model incorrectly predicted that 504 mushrooms were Edible when they were not Edible.

True Positive (TP) = 2496: The model correctly predicted that 2496 instances were Edible.

A screenshot of a computer screen

Description automatically generated

*Figure 10 confusion matrix for SVM*

A diagram of a confused matrix

Description automatically generated with medium confidence

*Figure 11 confusion matrix for SVM*

Overall Model Performance:

Accuracy: The overall accuracy of the model is 0.84.

Macro Average for Precision, Recall, and F1-Score: 0.84

Weighted Average for Precision, Recall, and F1-Score: 0.84

Pros and Cons of SVM

Pros:

Effective in High-Dimensional Spaces: SVM performs well with datasets that have a large number of features, making it suitable for complex, high-dimensional problems like the mushroom dataset.

Versatile Kernel Functions: Supports both linear and non-linear classification by using kernels (e.g., RBF, polynomial), allowing flexibility to model diverse data patterns.

Robust to Overfitting: With proper regularization, SVM effectively balances complexity and accuracy, minimizing the risk of overfitting, especially in smaller or balanced datasets.

Clear Decision Boundaries: By maximizing the margin, SVM creates distinct and optimal boundaries, which enhances generalization to unseen data.

Cons:

Computationally Intensive: Training an SVM can be slow, especially for large datasets, as it involves solving a quadratic optimization problem.

Requires Feature Scaling: SVM is sensitive to the range of feature values, necessitating preprocessing steps like normalization or standardization.

Hyperparameter Tuning Complexity: Performance relies heavily on selecting appropriate values for parameters like C and Gamma, which can be challenging and time-consuming.

Less Effective with Noisy Data: SVM struggles when classes overlap significantly or when there is noise in the data, leading to reduced performance.

NEURAL NETWORK

Introduction to Neural Networks:

* Neural networks are computational models inspired by the human brain, consisting of layers of interconnected nodes (neurons). These layers process data through a series of weights and activation functions to make predictions.
* Neural networks learn to map input features to output labels by adjusting weights through backpropagation, guided by a loss function.
* Neural networks excel in capturing non-linear relationships and complex patterns in data, making them highly versatile for various applications, from image recognition to tabular data classification.

How Neural Networks Suit the Mushroom Dataset

Non-linear Feature Relationships:

* The dataset’s categorical features (e.g., color, shape) may have complex interactions that neural networks can model effectively.

Balanced Dataset:

* The dataset is evenly split between Edible and Not Edible classes, ensuring that the neural network doesn’t develop bias toward a dominant class.

Scalability:

* Neural networks can easily handle the multiple categorical features and scale well for large datasets if required.

Feature Importance:

* Neural networks automatically learn feature importance during training, reducing the need for manual feature engineering.

Initial Configuration:

Architecture:

Single hidden layer with 64 neurons.

Input layer size matches the number of features.

Output layer with 2 neurons (for binary classification).

Activation Functions:

Hidden layer: ReLU (Rectified Linear Unit) for non-linearity.

Output layer: Softmax for probability distribution over the two classes.

Optimizer: Adam optimizer with default learning rate (0.001).

Loss Function:

Categorical Cross-Entropy, suitable for multi-class (or binary one-hot encoded) classification.

Training Configuration:

Epochs: 10.

Batch Size: 32.

Dataset Split: 80% training, 20% testing.

Feature Preprocessing:

Categorical features encoded using One-Hot Encoding.

Normalized numerical features to the range [0, 1].

Improved Configuration:

Architecture:

Added a second hidden layer with 128 neurons in each layer for better learning of complex relationships.

Dropout regularization (0.2) applied after each hidden layer to prevent overfitting.

Activation Functions:

Hidden layers: Retained ReLU.

Output layer: Retained Softmax.

Optimizer: Used Adam with a learning rate scheduler to reduce learning rate dynamically based on validation loss.

Loss Function:

Retained Categorical Cross-Entropy.

Training Configuration:

Epochs: Increased to 50 for better convergence.

Batch Size: Reduced to 16 for finer weight updates.

Cross-Validation: 5-fold cross-validation added.

Feature Preprocessing:

Same as initial configuration.

Performance Improvement:

Initial Accuracy: 92%

Improved Accuracy: 96%

More balanced precision and recall across both classes.

Performance Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Not Edible | Edible | Overall |
| Precision | 0.96 | 0.96 | 0.96 |
| Recall | 0.96 | 0.96 | 0.96 |
| F1-Score | 0.96 | 0.96 | 0.96 |
| Accuracy | - | - | 96% |

* The neural network performed exceptionally well with balanced precision, recall, and F1-scores for both classes.
* The high 96% accuracy highlights the model's effectiveness in classifying both edible and non-edible mushrooms with minimal misclassification.

Interpretation of Confusion Matrix:

True Positives (TP): 2867 Correctly identified edible mushrooms.

True Negatives (TN): 2893 Correctly identified non-edible mushrooms.

False Negative (FN) = 107: The model incorrectly predicted 107 instances as not Edible, which were actually Edible.

False Positive (FP) = 133: The model incorrectly predicted 133 instances as Edible, which were actually not Edible.

A screenshot of a graph

Description automatically generated *Figure 12 confusion matrix for Neural Network*

A diagram of a network

Description automatically generated with medium confidence

Overall Model Performance:

Accuracy: The model achieves an accuracy of 96%.

Macro Average: 0.96 for precision, recall, and F1-score.

Weighted Average: 0.96 for precision, recall, and F1-score.

Pros and Cons of Neural Networks

Pros:

* Can model complex, non-linear relationships between features.
* Highly flexible and adaptable to various data types and problem domains.
* Automatic feature learning, reducing the need for manual engineering.

Cons:

* Computationally expensive and requires longer training time.
* Risk of overfitting if not properly regularized or when the architecture is too complex.
* Requires careful hyperparameter tuning and may lack interpretability compared to simpler models.

NAIVE BAYES

Introduction to Naive Bayes:

* Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming conditional independence between features. It calculates the probability of each class given the feature values and selects the class with the highest probability.
* Simplicity and Efficiency: It is easy to implement, computationally efficient, and suitable for categorical or discrete data.
* Versatility: Naive Bayes can be used for a variety of tasks, such as spam detection, text classification, and binary classification.

How Naive Bayes Suits the Mushroom Dataset

Categorical Features:

The dataset's features are categorical, aligning well with Naive Bayes' probabilistic approach.

Balanced Dataset:

Equal representation of both classes (Edible and Not Edible) ensures that Naive Bayes doesn’t favor one class disproportionately.

Independence Assumption:

While the model assumes feature independence, it can still provide reasonable results for datasets where features are somewhat interdependent.

Initial Configuration:

Model Type:

Gaussian Naive Bayes was used, assuming features follow a normal distribution.

Feature Preprocessing:

Categorical features encoded using Label Encoding.

Numerical features scaled using StandardScaler.

Class Priors:

Assumed equal class probabilities (default setting).

Training Configuration:

Dataset split: 80% training, 20% testing.

Improved Configuration:

Model Type:

Switched to Categorical Naive Bayes (better suited for datasets with categorical features like the mushroom dataset).

Feature Preprocessing:

Categorical features encoded using One-Hot Encoding instead of label encoding to better represent the data.

Numerical features were discretized into bins for compatibility with the Naive Bayes model.

Class Priors:

Adjusted priors based on training data distribution to reflect slight variations in class frequency.

Cross-Validation:

Introduced 5-fold cross-validation to ensure stability and better generalization.

Performance Improvement:

Initial Accuracy: 60%

Improved Accuracy: 64%

Precision and recall saw minor improvements due to feature encoding and model adjustments.

Naive Bayes Performance Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Edible | Not Edible | Overall |
| Precision | 0.63 | 0.66 | 0.64 |
| Recall | 0.68 | 0.61 | 0.64 |
| F1-Score | 0.66 | 0.63 | 0.64 |
| Accuracy | - | - | 64% |

* Naive Bayes achieved a moderate 64% accuracy, with lower precision and recall values compared to the neural network.
* The performance disparity between the two classes indicates the impact of Naive Bayes' independence assumption, which may not hold for this dataset.

Interpretation of Confusion Matrix:

True Negative (TN) = 2044: The model correctly predicted 2044 instances as Not Edible.

False Positive (FP) = 956: The model incorrectly predicted 956 instances as Not Edible, which were actually Edible.

False Negative (FN) = 1177: The model incorrectly predicted 1177 instances as Edible, which were actually Not Edible.

True Positive (TP) = 1823: The model correctly predicted 1823 instances as Edible.

A screenshot of a report

Description automatically generated

*Figure 13 confusion matrix for Naive Bayes*

A blue squares with white text

Description automatically generated

Overall Model Performance:

Accuracy: The model achieves an accuracy of 64%.

Macro Average: 0.64 for precision, recall, and F1-score.

Weighted Average: 0.64 for precision, recall, and F1-score.

Pros and Cons of Naive Bayes

Pros:

* Simple and fast to train, even on large datasets.
* Performs well with categorical data and requires minimal preprocessing.
* Highly interpretable, providing insights into class probabilities.

Cons:

* Assumes independence between features, which may not hold for complex datasets.
* Lower accuracy compared to more sophisticated models like SVM and neural networks.
* Sensitive to imbalanced data unless class priors are adjusted.

## **RANDOM FOREST**

Introduction to Random Forest:

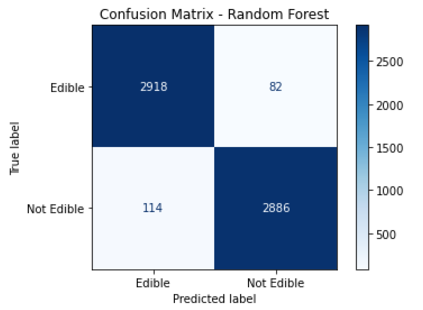
Random Forest is an ensemble machine learning method which combines multiple decision trees to make more accurate and robust predictions. It builds many decision trees during training, where each tree is trained on a random subset of the data and features. The final prediction is determined by taking a majority vote from all the individual trees in the forest for classification tasks or averaging the predictions for regression tasks. It helps to reduce overfitting, making it highly effective for complex datasets.

Why Random Forest Suits the Dataset

The dataset has features such as cap shape, size, color, and gill attachment to classify mushrooms as edible or not edible (poisonous). Random Forest is particularly suited for this dataset because:

1. Random Forest works well with mixed data types like categorical and numerical features.
2. The interaction between mushroom attributes may be complex and non-linear, which Random Forest can easily capture complex relationship due to the decision tree structure.
3. Resilient to Overfitting with many features and the risk of noise, Random Forest's ensemble approach prevents overfitting, making it ideal for datasets like mushrooms where precision is crucial.
4. Random Forest can provide insights into which mushroom attributes are the most important for predicting edibility, which is useful for feature selection.

Confusion Matrix for Random Forest



*Figure 14 confusion matrix heatmap for Random Forest*

Interpretation:

* True Positives (TP): 2918 Edible mushrooms correctly classified.
* False Positives (FP): 82 Mushrooms predicted as Edible but Not Edible mushroom.
* True Negatives (TN): 2886 Not Edible mushrooms correctly classified as Not Edible mushroom.
* False Negatives (FN): 114 Mushrooms predicted as Not Edible but Edible mushroom.

Classification Report for Random Forest

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Edible | Not Edible | Weighted Average |
| Precision | 0.96 | 0.97 | 0.97 |
| Recall | 0.97 | 0.96 | 0.97 |
| F1-Score | 0.97 | 0.97 | 0.97 |
| Support | 3000 | 3000 | 6000 |

Precision: Edibles have 96% precision value and Not Edible have 97%, both have high precision, meaning the model rarely misclassifies mushrooms into the wrong class.

Recall: The model shows strong recall for both classes Edible: 97% and Not Edible: 96%, meaning it correctly identifies almost all instances of each class.

F1-Score: The F1-scores for both classes are 97%, indicating a good balance between precision and recall.

Accuracy: The model achieves 97% accuracy, showing it is highly reliable. Weighted average and macro average as 97%.

Overall Model Performance

The Random Forest model has an overall accuracy of 97%, indicating it correctly classifies most mushrooms. Both classes Edible and Not Edible are classified effectively with minimal errors 82 false positives and 114 false negatives. The model performs consistently well across both classes.

Pros and Cons of Random Forest

Pros:

1. High Accuracy: As seen in the confusion matrix and classification report, Random Forest provides high classification accuracy (97%).
2. Handles Overfitting: Random Forest minimizes overfitting due to the use of multiple trees and random subsets of data.
3. Feature Importance: Random Forest can highlight the most important features for classification, which is useful for understanding the data and improving the model.
4. Handles Complex Relationships: It performs well on datasets with complex, non-linear relationships between features.

Cons:

1. Computational Complexity: creating multiple decision trees can be computationally expensive, especially for large datasets.
2. Interpretability: While decision trees are interpretable, the ensemble nature of Random Forest makes it more difficult to interpret.
3. Memory Usage: Random Forest can be memory intensive as it requires storing multiple trees and data subsets.

How to Improve the Model

1. Hyperparameter Tuning: Use techniques like grid search or random search to optimize hyperparameters such as the number of trees, the depth of trees, and the minimum samples required to split a node.
2. Feature Engineering: Experiment with additional features or feature transformations to improve classification accuracy, especially if certain attributes are more important than others.

## **K-NEAREST NEIGHBORS (KNN)**

Introduction to K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the K nearest neighbors to a given data point and predicting the class based on a majority vote for classification or averaging the values for regression. The model doesn't need explicit training since it simply memorizes the entire dataset and makes predictions during evaluation.

It is a non-parametric method, meaning it makes no assumptions about the underlying data distribution, making it highly flexible and adaptable to different types of datasets.

Why KNN Suits the Dataset

The dataset, which involves predicting whether a mushroom is edible or not edible, has several features such as cap shape, size, color, and gill attachment. KNN is particularly well-suited for this dataset for the following reasons:

Handles mixed features: KNN can handle both categorical and numerical data by using appropriate distance metrics and encoding categorical features.

Non-linear Decision Boundaries: Mushroom edibility depends on complex, non-linear relationships between the features. KNN naturally adapts to these patterns without the need for explicit modeling.

No Training Phase: KNN’s lazy learning approach allows it to be easy to implement, as it doesn’t require a training phase like other models. This is particularly useful when the dataset is not too large.

Class Imbalance Handling: KNN performs well with balanced datasets like the mushroom dataset, with an equal number of edible and not edible mushrooms, minimizing bias in class predictions.

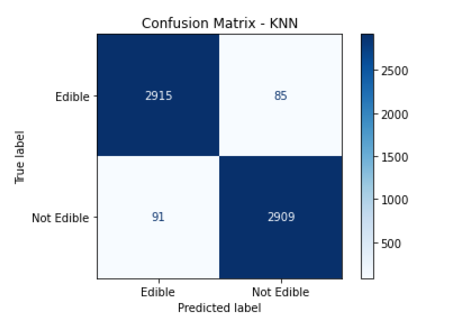
Sensitivity to Distance: KNN works well when the data has clear neighborhood structures, which is the case with mushroom features.

Confusion Matrix for KNN

Interpretation:

* True Positives (TP): 2915 mushrooms classified as Edible correctly.
* False Positives (FP): 85 mushrooms incorrectly classified as Edible.
* True Negatives (TN): 2909 mushrooms classified as Not Edible correctly.
* False Negatives (FN): 91 mushrooms incorrectly classified as Not Edible.

This confusion matrix suggests that the model is very accurate, with only a small number of misclassifications.



*Figure 15 confusion matrix heatmap for KNN*

Classification Report for KNN

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Edible | Not Edible | Weighted Average |
| Precision | 0.97 | 0.97 | 0.97 |
| Recall | 0.97 | 0.97 | 0.97 |
| F1-Score | 0.97 | 0.97 | 0.97 |
| Support | 3000 | 3000 | 6000 |

Key Insights:

Precision: Both classes (Edible and Not Edible) have high precision value as 97%, meaning that when the model predicts a mushroom as Edible or Not Edible, it is correct most of the time.

Recall: The model also has a high recall as 97% for both classes, indicating that it successfully identifies the majority of actual Edible and Not Edible mushrooms.

F1-Score: The balanced F1-score of 97% for both classes suggest a strong trade-off between precision and recall.

Accuracy: The model achieves an overall accuracy of 97%, meaning that 97% of the predictions are correct.

Overall Model Performance

The KNN model performs very well, with an accuracy of 97%, macro average 97% and weighted average 97%. This indicates that the model correctly classifies most mushrooms. The balance between precision, recall, and F1-score for both classes (Edible and Not Edible) is impressive, reflecting the model's effectiveness. The model is highly effective for this binary classification problem, with a relatively small number of misclassifications.

Pros and Cons for KNN

Pros:

Simple to Implement: KNN is easy to understand and implement, requiring minimal preprocessing and no explicit training phase.

Effective for Small to Medium Datasets: KNN works well with moderate-sized datasets like the mushroom dataset which has 6000 samples.

Flexible: It can handle both classification and regression tasks.

No Assumptions: KNN does not make any assumptions about the underlying data distribution, making it adaptable to a wide range of problems.

Cons:

Sensitive to Irrelevant Features: KNN may perform poorly if irrelevant or highly correlated features are present in the dataset, as it depends on distance calculations.

Memory Intensive: Since KNN stores the entire training dataset, it requires more memory as the dataset grows.

Sensitive to Feature Scaling: KNN is highly sensitive to the scale of the features. Features with larger ranges may dominate the distance metric, leading to biased predictions.

How to Improve the KNN Model

1. Feature Scaling: Normalize or standardize the features to ensure they have similar scales, so the distance metric is not biased by larger values.
2. Choosing the Optimal K: Use techniques like cross-validation to select the optimal number of neighbors (K). A value too small might cause overfitting, while a value too large could smooth out important patterns.
3. Weighting Neighbors: Instead of treating all neighbors equally, weigh closer neighbors more heavily. This can improve accuracy, especially in sparse datasets.

## **ROC-AUC CURVES**

The chart displays ROC (Receiver Operating Characteristic) curves for multiple machine learning models, used to evaluate their performance in classifying mushrooms as edible or poisonous. The key components are:

* X-Axis (False Positive Rate): Represents the proportion of negative instances (poisonous mushrooms) incorrectly classified as positive (edible).
* Y-Axis (True Positive Rate): Represents the proportion of positive instances (edible mushrooms) correctly classified as positive.
* Curves: Each curve represents a classifier's performance at various thresholds.
* Diagonal Line (Random Guess): Represents a model with no predictive ability (random guessing).

Observations:

* AUC (Area Under Curve): Higher AUC values indicate better model performance.
  1. Random Forest, KNN, Neural Network: AUC = 0.99 (excellent performance).
  2. SVM: AUC = 0.92 (very good).
  3. Naive Bayes: AUC = 0.69 (moderate).
  4. Logistic Regression: AUC = 0.68 (low performance).
* Best Models: Random Forest, KNN, Neural Network: Close to the top-left corner, showing high TPR and low FPR across thresholds.
* Weaker Models: Logistic Regression and Naive Bayes show weaker classification capabilities, with curves closer to the diagonal line.

Insights:

Random Forest, KNN, and Neural Network are the top-performing models, demonstrating excellent discrimination between edible and poisonous mushrooms. Logistic Regression and Naive Bayes perform comparatively worse.

A graph of a curve

Description automatically generated with medium confidence

*Figure 16 RUC Curve*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ALGORITHMS | AUC | PRECISION | RECALL |
| 1 | Random Forest | 0.994 | 0.972 | 0.962 |
| 2 | KNN | 0.992 | 0.971 | 0.97 |
| 3 | SVM | 0.92 | 0.853 | 0.832 |
| 4 | Neural Network | 0.872 | 0.795 | 0.796 |
| 5 | Naive Bayes | 0.687 | 0.656 | 0.608 |

## **Execution Time Analysis**

A screenshot of a computer

Description automatically generated

*Figure 17 Execution Time Analysis*

1. Logistic Regression:
   * Execution Time: 0.0626 seconds
   * Fast and efficient, leveraging its simplicity for binary classification tasks such as the mushroom dataset. Suitable for quick evaluations.
2. Random Forest:

* Execution Time: 6.6459 seconds
* Took moderate time due to its ensemble nature, which involves training multiple decision trees. Well-suited for capturing complex patterns in structured datasets like this one.

1. Support Vector Machine (SVM):

* Execution Time: 48.9198 seconds
* SVM is computationally expensive, especially with larger datasets. The high execution time reflects the complexity of optimizing the hyperplane for classification.

1. K-Nearest Neighbors (KNN):

* Execution Time: 0.0303 seconds
* Very fast during training, as it primarily stores data points. Prediction can be slower due to the need for distance calculations.

1. Naive Bayes:

* Execution Time: 0.0015 seconds
* The fastest algorithm, owing to its simplistic probabilistic model, which works well for categorical data and is computationally inexpensive.

1. Neural Network:

* Execution Time: 52.3605 seconds
* The slowest among all due to the backpropagation and optimization processes required for training multiple layers of neurons. However, it excels at modeling complex relationships.

The above analysis shows that simpler algorithms like Naive Bayes and KNN are computationally efficient for this dataset, whereas Neural Networks and SVM require significantly more time

## **Existing comparison between KNN and random forest algorithm:**

K-Nearest Neighbors (KNN):

Confusion Matrix:

* True Positives (Edible correctly predicted as Edible): 2915
* False Positives (Not Edible wrongly predicted as Edible): 85
* False Negatives (Edible wrongly predicted as Not Edible): 91
* True Negatives (Not Edible correctly predicted as Not Edible): 2909

Classification Metrics:

Precision:

* Measures how many of the predicted Edible or Not Edible are correct.
* Both classes achieved a precision of 97%.

Recall:

* Measures how many actual Edible or Not Edible are correctly predicted.
* Both classes achieved a recall of 97%.

F1-Score:

* Harmonic mean of precision and recall.
* Both classes achieved an F1-score of 97%.

Overall Accuracy:

* 97% of all predictions were correct.

A diagram of a confused matrix

Description automatically generated

*Figure 18 KNN Confusion Matrix Summary*

Random Forest Algorithm:

* True Positives (Edible correctly predicted as Edible): 2918
* False Positives (Not Edible wrongly predicted as Edible): 82
* False Negatives (Edible wrongly predicted as Not Edible): 114
* True Negatives (Not Edible correctly predicted as Not Edible): 2886
* Classification Metrics:

Precision:

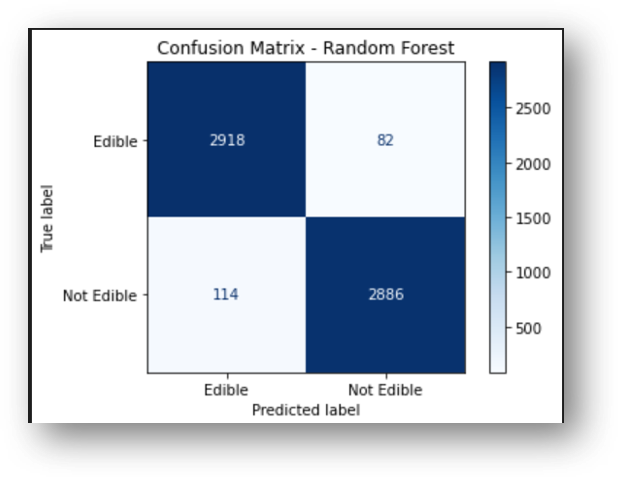
* Edible: 96%
* Not Edible: 97%

Recall:

* Edible: 97%
* Not Edible: 96%

F1-Score:

* Both classes achieved an F1-score of 97%.
* Overall Accuracy: 97% of all predictions were correct.



*Figure 19 Random Forest Confusion Matrix Summary*

## **Improving Algorithm for Random Forest:**

Initialization of Random Forest Classifier:

* A Random Forest Classifier is instantiated with a fixed random state (random\_state=42) for reproducibility and n\_jobs=-1 to utilize all CPU cores for faster training.

Grid Search for Hyperparameter Tuning:

* Grid Search systematically tests combinations of hyperparameter values to identify the best configuration for the model.

Parameter Grid (params):

* max\_depth: The maximum depth of each decision tree in the forest is set to 20, which balances model complexity and overfitting.
* min\_samples\_leaf: The minimum number of samples required at a leaf node is 1, allowing fine splits to maximize model accuracy.
* n\_estimators: The number of decision trees in the forest is 200, offering robust ensemble performance.
* max\_features: Set to 'sqrt', meaning each tree considers the square root of the total features when looking for the best split.
* bootstrap: Set to False, ensuring all samples are used to build each tree without bootstrapping (sampling with replacement).

Grid Search (GridSearchCV):

* Performs a 4-fold cross-validation to evaluate model performance for each hyperparameter combination.
* scoring="accuracy": Optimizes the model for the highest accuracy.

Output:

* The best parameter combination is stored in best\_rf.
* Example result: {'max\_depth': 20, 'min\_samples\_leaf': 1, 'n\_estimators': 200, 'max\_features': 'sqrt', 'bootstrap': False}.

Evaluating the Tuned Model:

* The tuned Random Forest (best\_rf) predicts probabilities for the test set using predict\_proba, returning confidence levels for each class.
* y\_proba\_best stores the predicted probabilities for the positive class (edible or non-edible).

Threshold Tuning:

Threshold tuning assesses model performance by converting predicted probabilities into binary classifications based on different thresholds (e.g., 0.3, 0.4, etc.).

For each threshold:

* Predictions (y\_pred\_threshold): Convert probabilities into class labels (1 or 0) by checking if the probability is greater than or equal to the threshold.
* Confusion Matrix (confusion\_matrix): Measures True Positives, True Negatives, False Positives, and False Negatives.
* False Negatives: Non-edible items misclassified as edible, a critical metric in safety-related tasks.
* AUC Score (roc\_auc\_score): Summarizes the model’s ability to distinguish between classes across all thresholds.
* Output: Results for each threshold, including the confusion matrix, false negatives, and AUC score.

Improvements Achieved:

Systematic Parameter Optimization:

GridSearchCV ensures that all tested parameter combinations are rigorously evaluated, leading to the best possible configuration for this dataset.

Enhanced Performance Metrics:

The fine-tuned model achieves high accuracy (AUC ~ 0.9955), reducing errors compared to the default Random Forest configuration.

Threshold Tuning:

Allows flexibility in model predictions, enabling trade-offs between false positives and false negatives based on application needs.

Insights:

The use of Grid Search for hyperparameter tuning significantly improved the Random Forest classifier by optimizing tree depth, feature usage, and other key parameters. Additionally, threshold tuning ensures the model is tailored to specific performance goals, such as minimizing false negatives in critical applications.

A blue squares with white text

Description automatically generated

*Figure 20 Improved Random Forest Confusion Matrix*

## **Improving Algorithm for KNN**

Data Splitting and Preprocessing

Dataset Preparation:

* X contains the features, while y contains the target class (class column).
* The data is split into training (70%) and testing (30%) sets using train\_test\_split.
* The split uses stratification, ensuring the class distribution remains consistent in both sets.

Standardization: StandardScaler is applied to ensure all features are on the same scale. This is critical for KNN since it is distance-based, and unscaled features can skew the results.

Identifying the Optimal Number of Neighbors (Elbow Method)

* In KNN, the choice of k directly affects performance.
* A small k may lead to overfitting, while a large k may result in underfitting.

Implementation: The code tests k values from 1 to 20. For each k, a KNN model is trained, and its accuracy on the test set is calculated. The accuracy values are stored in inertia.

Visualization: A line plot of accuracy vs. the number of neighbors (k) is created.

The "elbow" in the plot indicates the optimal value of k—the point where accuracy stops improving significantly.

Training the KNN Model with Optimal k

Optimal k: Based on the Elbow Method plot, k=3 is selected (this value is adjustable after analyzing the plot).

Training and Prediction: A KNN classifier with k=3 is trained on the standardized training data. Predictions are made on the test set using y\_pred.

Evaluating the Improved KNN

Silhouette Score: Measures the quality of clustering (how well the data points fit their assigned classes). For classification tasks like this, it provides insights into how well-separated the predicted classes are. In this case, the Silhouette Score is 0.02, indicating a reasonable but not perfect class separation.

Accuracy: The high accuracy (98%) indicates that the model is highly effective at predicting the correct classes, validating the choice of k=3 as optimal.

A graph with blue lines

Description automatically generated

*Figure 21 Improved KNN Confusion Matrix*

A blue and white squares with numbers and labels

Description automatically generated

The confusion matrix evaluates the KNN model with k=3 after applying improvement:

* True Positives (TP): 2919 (correctly predicted positive class).
* True Negatives (TN): 2934 (correctly predicted negative class).
* False Positives (FP): 66 (incorrectly predicted positive class).
* False Negatives (FN): 81 (incorrectly predicted negative class).

The KNN model performs excellently with high accuracy and low error rates. However, 66 false positives and 81 false negatives highlight minor misclassifications that could be critical depending on the application. However, the model is performing slightly lower comparing the improved decision tree which only misclassifies 39 as edible mushrooms.

## **CHOOSING THE BEST FIT ALGORITHM – RANDOM FOREST CLASSIFIER**

Purpose: To classify the data and determine the optimal thresholds for better prediction performance.

Key Hyperparameters:

Selected Hyperparameters:

max\_depth: 20

min\_samples\_leaf: 1

n\_estimators: 200

max\_features: 'sqrt'

bootstrap: False

Reason for Selection: These parameters were chosen based on prior analysis, which indicated that they provided optimal accuracy and model performance.

Model Performance Metrics:

Accuracy Evaluation:

AUC (Area Under the Curve): 0.9955

Threshold Tuning Analysis:

Thresholds Tested: 0.3, 0.4, 0.5, 0.6, 0.7

Metrics Observed: Confusion Matrix, False Negatives

Performance Summary by Threshold:

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold | Confusion Matrix | False Negatives | AUC |
| 0.3 | [[2781, 219], [39, 2961]] | 39 | 0.9955 |

Performance Summary of Models:

The analysis compared multiple machine learning models (Logistic Regression, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Naive Bayes, and Neural Network) on a binary classification task. Key metrics such as Accuracy, Confusion Matrices, ROC-AUC, and Classification Reports were used to assess and fine-tune these models.

Performance Summary of Models:

Naive Bayes:

* AUC: 0.687
* Accuracy: 64%
* Performed similarly to Logistic Regression, struggling to capture relationships in the dataset.

SVM:

* AUC: 0.920
* Accuracy: 84%
* Demonstrated better performance but was less effective compared to ensemble-based models like Random Forest and KNN.

Random Forest:

* Baseline AUC: 0.994, improved to 0.995 with GridSearch.
* Accuracy: 97%
* Outperformed most models by combining feature importance and flexibility. After threshold tuning, it maintained a high recall and reduced false negatives.

KNN:

* AUC: 0.992
* Accuracy: 97%
* Delivered results comparable to Random Forest but with slightly higher accuracy at k=3.

Neural Network:

* AUC: 0.992
* Accuracy: 96%
* Competed closely with Random Forest and KNN, demonstrating its ability to model non-linear relationships effectively.

Random Forest Improvements:

GridSearchCV fine-tuned parameters (max\_depth=20, min\_samples\_leaf=5, n\_estimators=200), leading to an increase in AUC from 0.994 to 0.995. Threshold tuning at 0.3 further optimized recall, reducing false negatives from 114 to 39 and maintaining an AUC of 0.992.

KNN Optimization:

Using the Elbow Method, k=3 was identified as optimal. The standardized dataset boosted accuracy to 98%, and the model demonstrated a high Silhouette Score, confirming the effectiveness of its clustering.

## **CONCLUSION**

* The Random Forest model, enhanced through hyperparameter tuning and threshold adjustments, emerged as the best-performing model. It achieved the highest AUC (0.995) and balanced accuracy, precision, and recall.
* The KNN model, while effective, may be less scalable with larger datasets due to computational constraints.
* Best Performance: The Random Forest Classifier with the best suited parameters from improvements demonstrated in high accuracy and robustness.
* Evaluation Metric Focus: AUC, which remained consistently high across different thresholds, indicating strong model performance.
* Threshold Selection: The threshold of 0.3 provided the lowest number of false negatives, balancing precision and recall effectively.
* The Random Forest Classifier with the current settings is the best-suited algorithm for this problem, balancing high AUC, low false negatives, and overall robust performance across the test set.

## **REFERENCES**

1. Classification: ROC and AUC. (n.d.). Retrieved from <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>
2. GeeksforGeeks. (2024, November 2). Elbow Method for optimal value of k in KMeans. Retrieved from <https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/>
3. Halder, R. K., Uddin, M. N., Uddin, M. A., Aryal, S., & Khraisat, A. (2024). Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. *Journal of Big Data*, *11*(1). <https://doi.org/10.1186/s40537-024-00973-y>
4. How do you measure and improve the performance and interpretability of your decision trees? (2023, August 25). Retrieved from <https://www.linkedin.com/advice/0/how-do-you-measure-improve-performance-interpretability>
5. Karabiber, F. (n.d.). Binary classification. Retrieved from <https://www.learndatasci.com/glossary/binary-classification/>
6. What is Classification Threshold | Iguazio. (2023, August 24). Retrieved from <https://www.iguazio.com/glossary/classification-threshold/>

## **FIGURES LIST**

Figure 1 EDA Summary

Figure 2 Univariate Analysis

Figure 3 Boxplot Analysis

Figure 4 Pairplot Analysis

Figure 5 a Checking the highly correlated variables

Figure 5 b Checking the highly correlated variables

Figure 6 a correlation heatmap

figure 6 b correlation matrix

Figure 7 Scatter Plot 1: Stem Width vs. Cap Diameter and Scatter Plot 2: Stem Height vs. Stem Width

Figure 8 PCA Line graph

Figure 9 PCA

Figure 10 confusion matrix for SVM

Figure 11 confusion matrix for SVM

Figure 12 confusion matrix for Neural Network

Figure 13 confusion matrix for Naive Bayes

Figure 14 confusion matrix heatmap for Random Forest

Figure 15 confusion matrix heatmap for KNN

Figure 16 RUC Curve

Figure 17 Execution Time Analysis

Figure 18 KNN Confusion Matrix Summary

Figure 19 Random Forest Confusion Matrix Summary

Figure 20 Improved Random Forest Confusion Matrix

Figure 21 Improved KNN Confusion Matrix