**Abstract**

Have you ever been walking down the streets of Windsor and been plagued by the question of determining the edibility of a mushroom? [1] While mushroom foraging has a rich tradition and a community of enthusiasts, it also poses inherent risks. This comes as some species can be toxic and potentially fatal if consumed. Reliable identification of edible and non-edible mushrooms is crucial for the safety and enjoyment of foragers and nature enthusiasts alike. This research endeavor delves into the mixture of machine learning methods with the natural world. Therefore, the main goal of this research endeavor is to develop a robust and accurate method for classifying the edibility of gilled mushrooms, thereby, providing a valuable tool for those exploring the world of mycology.

1. **Introduction**

The root of the problem lies in the difficulty of reliably distinguishing between edible and non-edible mushrooms. This challenge is compounded by the wide variety of mushroom species with subtle visual differences. Machine learning presents a compelling opportunity for tackling the task of classifying the edibility of mushrooms. Its capacity to analyze extensive datasets, detect intricate patterns, and provide data-driven insights aligns seamlessly with the complex task of mushroom identification. Through this research, we intend to harness the power of machine learning to revolutionize the way we approach the age-old practice of mushroom foraging, making it safer, more accessible, and more enjoyable for all. As we examine the details of our dataset and data preprocessing in the following sections, we will highlight the steps taken to prepare the data for our machine learning models.

1. **Dataset**

The dataset encompasses various mushroom characteristics, including cap shape, cap surface, cap color, bruises, odor, gill attachment, gill spacing, gill size, gill color, stalk shape, stalk root, stalk surface above and below the ring, stalk color above and below the ring, veil type, veil color, ring number, ring type, spore print color, population, and habitat. These attributes provide a rich and diverse set of features for our machine learning models to analyze. With a substantial collection of 8416 mushroom samples, this dataset provides a wealth of information to explore. The dataset contains two classes, "EDIBLE" and "POISONOUS”, indicating whether a mushroom is safe to consume or not. [ADD fig ]

1. **Data Preprocessing**

**Cleaning and Feature Selection**

An initial visual inspection of the data presented columns that had null values. Pandas was used to load the comma separated file (csv). The dataset was explored for the existence of duplicate values. To address the number of duplicate rows in the dataset the *duplicated()* function was used. This identifies rows with the same values in all columns. These duplicate rows are removed from the dataset by dropping the rows with the *drop\_duplicates()* function.

Continuing our data refinement process, a small function [ADD pic] was used on the data to quickly check for null values. The conclusion was that only 1 column had missing values. The one column was labelled ‘stalk-root’ with a total of 2480 missing values. Since, the entire dataset contained 8416 samples this resulted in a significant portion of the data that would require data imputation. This is the process of filling in missing or null values with estimated or predicted values. Given that the 29% of the data in this column would need to be altered and inevitably affect the model’s precision, the column was dropped.

The next step was to determine what columns would help in classifying the data and which columns would not contribute. This was done by checking for the number of unique values in each of the remaining columns. [ADD table pic] This exposed a column that remained constant throughout the dataset. A column that is constant are considered “zero-variance” features because they do not change over the dataset and would not affect our classification. Because of this justification the ‘veil-type’ column was subsequently dropped.

**Encoding and Processing Data**

In order for the models to be trained, the data needs to be processed in a way that is easy for the models to perform tasks on. This is useful for machine learning models, as they require numerical input. The dataset consisted of string data which is not ideal, so utilizing *LabelEncoder()*, from the Sci-kit learn library, to convert the categorical/text data into numerical values. The mappings from the original values to numbers are stored in a dictionary structure(map) to utilize them for future tasks.

To continue with preprocess the data, *StandardScaler()* from scikit-learn is used to standardize features. This process involves transforming the data into a standardized form, where it exhibits a mean of 0 and a standard deviation of 1, effectively scaling it to have unit variance. This step is both valuable and crucial because the model may struggle when working with features of varying scales during training.

In the final step, the dataset is divided into three distinct sets: training data, validation data, and testing data. The training data is utilized to teach the model and allow it to learn the underlying patterns in the dataset. The validation data is employed to fine-tune and optimize the model during its training phase. This helps to ensure that the model will generalizes well to unseen data. Lastly, the testing data serves as an independent assessment to evaluate the model's performance on new, unencountered examples, providing a convincing measure of its predictive capabilities.

Since, the goal is to classify if a mushroom is edible or poisonous, 2 separate sets of data are made. One with the labels will contain the class column which contains the "EDIBLE" and "POISONOUS" values. This is known as the target column. The other set contains the remain columns known as the feature column. The division was done using an 80-20 split into four sets labelled x\_train, x\_test, y\_train, y\_test. This means that 80% is used for training and 20% for the testing. This makes it so that the models are exclusively trained on the feature data and evaluated against the actual labels.

1. **Model Selection**

Model selection is a crucial step in machine learning where different algorithms are chosen to train on your dataset. Subsequently, the performance of each model was assessed with precision, aiming to identify the best one for the task. The distinct characteristics of the five major classification machine learning models employed in this project—Random Forest Classifier, Logistic Regression, Support Vector Machine, Decision Tree Classifier, and Gaussian Naive Bayes—will be explored.

Random Forest is used in classification because it combines the strengths of decision trees with ensemble techniques, offering high accuracy, robustness, and flexibility in handling different types of data and problem domains.

Logistic Regression is useful in scenarios where the connection between features and the target variable is predominantly linear. when the probabilistic aspects of the outcomes are crucial. However, its performance might be less impressive compared to more complex models when dealing with data that exhibits non-linear patterns.

Support Vector Machines (SVM) are a popular choice for classification due to their ability to handle high-dimensional data, robustness to overfitting, versatility in handling non-linear problems, and the emphasis on maximizing the margin between classes. They are particularly useful when a clear margin of separation exists between different classes.

Decision tree classifiers are popular in classification tasks because of their interpretability, ease of use, versatility, and ability to handle non-one-dimensionality data. The reason they are such a valuable tool it that this model strikes a balance between complexity and transparency.

Gaussian Naive Bayes is used for classification due to its simplicity, efficiency, and effectiveness in handling data. It's particularly well-suited for text classification, multiclass problems, and applications where quick decisions are needed. [ADD pic of models used]

Neural Network classifiers are the standard now in the realm of classification models, for their remarkable ability to model complex relationships, non-linear patters within high large scale and high dimensional data. These models are composed of interconnected nodes (neurons) organized into layers, where each connection has an associated weight that is adjusted during training to minimize the error in predictions. NN’s can self-learn features, reducing the need for manual feature engineering.

1. **Model Training and Evaluation**

The performance of multiple classification models to determine their capability with the dataset were evaluated. The process involves training each model, making predictions, and evaluating their effectiveness using a loop that iterates through each model in our predefined list. This systematic approach to model training and evaluation allows for assessing the strengths and weaknesses of each of the classification algorithm used. After training, each model makes predictions on previously unseen data, represented by the test dataset (X\_test). The goal was to understand how well these models could generalize to new, unseen data samples. This rigorous evaluation process enables making an informed decisions about model selection for classification task, also ensuring that the chosen model would yield accurate and reliable predictions.

Cross-validation was employed as a pivotal technique to rigorously evaluate the performance of classification models. This approach involved systematically partitioning the dataset into a predefined number of folds using *StratifiedKFold()*. Through an iterative process, each classification model underwent training and testing phases on different combinations of these folds. During each iteration, a model was trained on one subset and tested on another, ensuring comprehensive learning and assessment. The results, including accuracy scores, confusion matrices, and classification reports, were stored across all folds. Like the evaluation above this not only offered valuable insights into each model's consistency but also allowed for a strong evaluation of its ability to generalize to unseen data.

1. **Hyper-Parameter Tuning**
2. **Initial Training**

Logistic regression:

This model was run using default parameters and it achieved a good performance with an accuracy of 95%, with an average cross validation score of 94%.

Decision Tree Classifier:

This model was trained with initial parameters, criterion as ‘gini’, maxdepth of tree as 3. These initial parameters were chosen to determine how the decision tree would model relationships and do it quickly as gini is computationally less expensive. It achieved a very high accuracy of 97%, and an average cross validation score of 97% ± 1% accuracy.

Random Forest Classifier:

This model was trained with initial parameters, criterion as ‘gini’ and maxdepth of tree as 4. For the same reasons as decision tree, gini was used for a faster computation result and small depth for the trees in order to not categorize any of the noise in the data as a pattern. It achieved an initial accuracy of 98% which was already high and an average cross validation score of 99% accuracy.

Support Vector Machine:

This model was trained with initial parameters, kernel as linear, and C as 0.025, considering our large training split of 80% and testing of 20%, it made sense to use C as 0.025 considering we didn’t want to overfit or be sensitive to outliers.

It ended up achieving a high initial accuracy of 94%, and an average cross validation score of 94% accuracy.

1. **Hyperparameters**

**Definition:**

Hyperparameters are parameters whose values are set prior to the training phase and govern the learning process of a machine learning algorithm. Unlike model parameters, which are learned directly from the training data, hyperparameters are configured to control the overall behavior of the learning algorithm. The prefix 'hyper' indicates that they operate at a level above parameters, influencing the learning process and consequently the resulting model parameters. Correctly tuning hyperparameters can be crucial for optimizing a model's predictive performance.[3]

**Methods used:**

**GridSearchCV** from the sci-kit learn library was used in order to conduct the hyperparameter search, as this allows us to create a predefined grid of hyperparameters and explore all the combinations.

**Logistic regression:**

For logistic regression, the hyperparameters tuned were:

**C:** Inverse of regularization strength explored were [0.01, 0.1, 1, 10].

A smaller value applies more regularization, which helps with overfitting, however if it is too small, model may underfit.

**Penalty:** Norm used in penalization considered were ['l1', 'l2'].

L1 regularization tends to produce sparser models by driving certain parameters to 0, L2 regularization doesn’t on the other hand, but instead shrinks them toward zero.

**Solver:** Algorithm for optimization examined included ['liblinear', 'saga'].

Different solvers have different computational efficiencies depending on data size and structure. Liblinear, performs better on smaller datasets, saga is better suited for larger ones.

**Max Iter:** Maximum iterations allowed for solvers to converge were [100, 200, 300**].**

Higher maximum of iterations allows optimization algorithm more attempts to find the minimum cost function and converge to the solution.

After grid search and cross-validation process, the best parameters for the Logistic Regression model were identified as follows:

**Best parameters for Logistic Regression: {'C': 0.01, 'max\_iter': 100, 'penalty': 'l1', 'solver': 'saga'}.**

They did not help increase the accuracy of the model as much.

**Decision Tree Classifier:**

For Decision tree, the hyperparameters tuned were:

**Criterion:** A function to measure the quality of a split. Explored values were ['entropy', 'gini', ‘log\_loss’].

Gini tends to be faster to compute and may lean towards larger partitions (broader trees), whereas Entropy can produce more balanced trees but is computationally expensive due to calculation of logarithms.

**Max Depth:** The maximum depth of the tree explored depths were [3, 5, 7, 9].

Deeper trees (higher ‘max\_depth’) can model more complex patterns and accurate on the training data, but risk overfitting to noise or outliers in the training set.

After grid search and cross-validation, the best parameters for Decision Tree Classifier were identified as follows:

Best parameters for DecisionTreeClassifier: {'criterion': 'entropy', 'max\_depth': 7}

These parameters were optimal as the model achieved perfect accuracy with them.

**Support Vector Classifier/Machine:**

For SVC, the hyperparameters tuned were:

**C (Regularization Parameter):**  this parameter trades off correct classification of training examples against maximization of decision function’s margin. Parameters explored were ‘[0.025, 0.25, 2.5]’.

**Kernel:** choice of kernel affects the decision boundary created by the SVM. Linear kernel can model linear decision boundaries, while the rbf and poly kernels can model non=linear boundaries. Parameters explored were ['linear', 'rbf', 'poly'].

**Degree:** this is only applicable when using poly kernel and determines polynomial degree. Higher degree = model more complex relationships, but risk of overfitting increases. Lower degree is less computationally expensive and may generalize better. Parameters explored were [2, 3, 4].

After grid search and cross-validation, the best parameters for SVC were identified as follows:

Best parameters for SVC: {'C': 0.25, 'degree': 3, 'kernel': 'poly'}

These parameters were optimal as the model achieved perfect accuracy with them.

From this we can say that since the C value is small, it used a lower margin, therefore a simpler decision function but it did not end up affecting the accuracy as much and using the poly boundary also helped in this case with moderate degree of 3.

**Random Forest Classifier (RFC):**

For RFC, the hyperparameters tuned were:

**Criterion:** Same as decision tree A function to measure the quality of a split. Explored values were ['entropy', 'gini', ‘log\_loss’].

Gini tends to be faster to compute and may lean towards larger partitions (broader trees), whereas Entropy can produce more balanced trees but is computationally expensive due to calculation of logarithms.

**n\_estimators (number of trees in the forest):** this parameter determines number of trees in the forest. More trees = increase in robustness and stability of model, makes sure that outliers and noise don’t impact the model in training, but increasing computational cost. Parameters tuned were ‘[10, 50, 100]’.

**max\_depth** **(maximum depth of tree):** this parameter controls the maximum depth of each tree, more depth to each tree, helps model further complex patterns. Going to deep also risks overfitting. Parameters tuned were ‘[3, 5, 7]’.

After grid search and cross-validation, the best parameters for RFC were identified as follows:

Best parameters for RandomForestClassifier: {'max\_depth': 5, 'n\_estimators': 10}.

The model achieved perfect accuracy with the above parameters.

1. Comparison of Results
2. Conclusion and Findings
3. Description of Student Participation

References

[1] – fun guyz

dataset

<https://archive.ics.uci.edu/dataset/73/mushroom>

[3]. <https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>