Project #1

COMP 4730

Intro to Machine Learning

Classification of Edibility of Mushrooms Using Machine Learning Techniques

Authors:
George Kaceli
Justin Neal
Akshat Sharma

Submitted to:
Dr. Robin Gras

October 18th, 2023

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.

I. Introduction

The root of the problem lies in the difficulty of reliably distinguishing between edible and non-edible mushrooms.[2] 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.

II. 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" (Fig. 1), indicating whether a mushroom is safe to consume or not. [3]

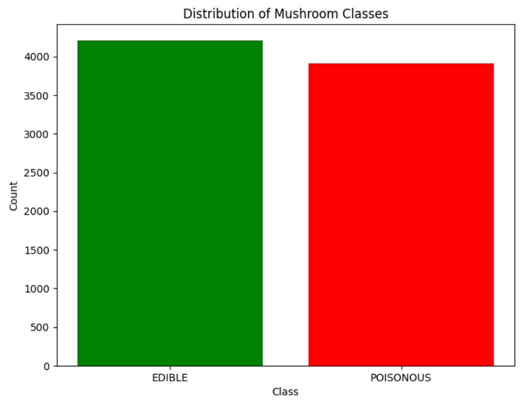


Fig. 1 Distribution of Mushrooms Classes

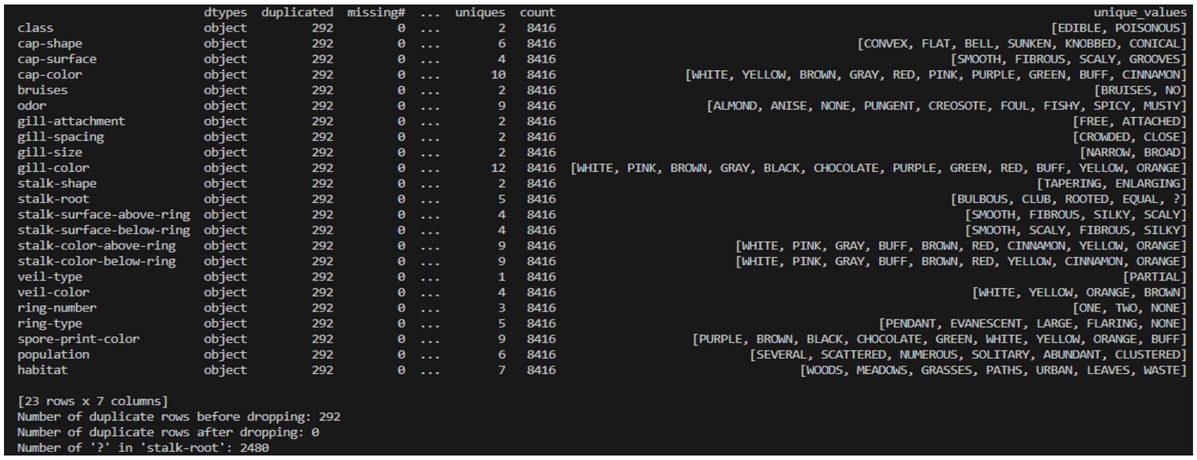
III. Data Preprocessing

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 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.[4]

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 (Fig.2). 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.



```
                          dtypes  duplicated  missing#  ...  uniques  count                                                                          unique_values
class                     object         292         0  ...        2   8416                                                                    [EDIBLE, POISONOUS]
cap-shape                 object         292         0  ...        6   8416                                         [CONVEX, FLAT, BELL, SUNKEN, KNOBBED, CONICAL]
cap-surface               object         292         0  ...        4   8416                                                      [SMOOTH, FIBROUS, SCALY, GROOVES]
cap-color                 object         292         0  ...       10   8416                 [WHITE, YELLOW, BROWN, GRAY, RED, PINK, PURPLE, GREEN, BUFF, CINNAMON]
bruises                   object         292         0  ...        2   8416                                                                          [BRUISES, NO]
odor                      object         292         0  ...        9   8416                    [ALMOND, ANISE, NONE, PUNGENT, CREOSOTE, FOUL, FISHY, SPICY, MUSTY]
gill-attachment           object         292         0  ...        2   8416                                                                       [FREE, ATTACHED]
gill-spacing              object         292         0  ...        2   8416                                                                       [CROWDED, CLOSE]
gill-size                 object         292         0  ...        2   8416                                                                        [NARROW, BROAD]
gill-color                object         292         0  ...       12   8416 [WHITE, PINK, BROWN, GRAY, BLACK, CHOCOLATE, PURPLE, GREEN, RED, BUFF, YELLOW, ORANGE]
stalk-shape               object         292         0  ...        2   8416                                                                  [TAPERING, ENLARGING]
stalk-root                object         292         0  ...        5   8416                                                      [BULBOUS, CLUB, ROOTED, EQUAL, ?]
stalk-surface-above-ring  object         292         0  ...        4   8416                                                        [SMOOTH, FIBROUS, SILKY, SCALY]
stalk-surface-below-ring  object         292         0  ...        4   8416                                                        [SMOOTH, SCALY, FIBROUS, SILKY]
stalk-color-above-ring    object         292         0  ...        9   8416                        [WHITE, PINK, GRAY, BUFF, BROWN, RED, CINNAMON, YELLOW, ORANGE]
stalk-color-below-ring    object         292         0  ...        9   8416                        [WHITE, PINK, GRAY, BUFF, BROWN, RED, YELLOW, CINNAMON, ORANGE]
veil-type                 object         292         0  ...        1   8416                                                                              [PARTIAL]
veil-color                object         292         0  ...        4   8416                                                         [WHITE, YELLOW, ORANGE, BROWN]
ring-number               object         292         0  ...        3   8416                                                                       [ONE, TWO, NONE]
ring-type                 object         292         0  ...        5   8416                                            [PENDANT, EVANESCENT, LARGE, FLARING, NONE]
spore-print-color         object         292         0  ...        9   8416                  [PURPLE, BROWN, BLACK, CHOCOLATE, GREEN, WHITE, YELLOW, ORANGE, BUFF]
population                object         292         0  ...        6   8416                          [SEVERAL, SCATTERED, NUMEROUS, SOLITARY, ABUNDANT, CLUSTERED]
habitat                   object         292         0  ...        7   8416                                 [WOODS, MEADOWS, GRASSES, PATHS, URBAN, LEAVES, WASTE]

[23 rows x 7 columns]
Number of duplicate rows before dropping: 292
Number of duplicate rows after dropping: 0
Number of '?' in 'stalk-root': 2480
```

Fig. 2 Values in the dataset columns.

Encoding and Processing Data

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. [5]

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.

Another preprocessing alternative that we use is one hot encoding where all the categorical features in the matrix x are one-hot encoded using OneHotEncoder from Scikit-learn, which converts categorical variable(s) into a form that could be provided to the algorithms we use to do a better job in prediction. The resultant is a DataFrame where each unique category in the original features is expanded into a new binary feature. [5]

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.[5]

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.

IV. 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 (RFC), Logistic Regression, Support Vector Machine (SVM), Decision Tree Classifier, and Gaussian Naive Bayes—will be explored.[6]

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 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. [6]

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.

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.

V. 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.

The following metrics were also used to determine how the model performed:

1. **Precision:** It is a measure of the accuracy of the positive predictions, it indicates how many of the positively predicted instances were positive. A higher precision indicates a lower false positive rate.

$$Precision = \frac{True\ Positives(TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

2. **Accuracy:** it is a metric that generally measures the overall correctness of the model, regardless of the class label. In a binary or multiclass classification, accuracy represents the ratio of the correctly predicted instances to the total number of instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

or

$$Accuracy = \frac{True\ Positives(TP) + True\ Negatives\ (TN)}{True\ Positives(TP) + True\ Negatives\ (TN) + False\ Positives(FP) + False\ Negatives\ (FN)}$$

3. **Recall:** it calculates the ratio of the correctly predicted positive observations to all the actual positives, also indicates the model's ability to find and correctly predict all relevant cases. $Recall = \frac{(True\ Positives\ (TP))}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$

4. **F1 score:** it is the harmonic mean of precision and recall, it considers both false positives and false negatives, and is useful when class distribution is imbalanced.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

VI. Hyper-Parameter Tuning

A. Initial Training

Logistic regression:

This model was run using parameters {C=0.01, penalty='l1', solver='liblinear'}(Fig.3)(Table 1) and it achieved the following results:

Logistic Regression CV Accuracy: 1.00 +/- 0.00

From the results we can determine that the model wasn't overfitting considering the above metrics. The cross validation and training seem consistent.

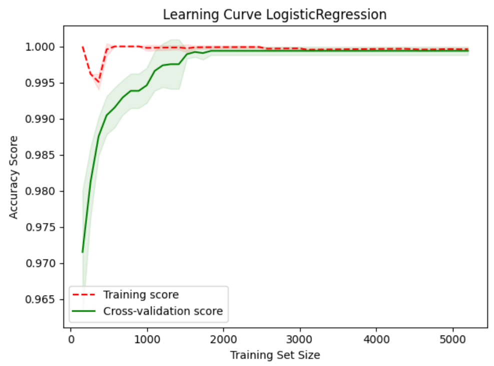


Fig.3: Logistic Regression Training and CV curve

	Training	Testing
Accuracy	99.97%	99.94%
Precision	100.00%	100.00%
Recall	99.94%	98.94%
F1 Score	99.97%	99.946%

Table 1: Logistic Regression Training and CV curve

Decision Tree Classifier:

This model was trained with initial parameters {criterion="gini", max_depth=3}(Fig.4)(Table 2). These initial parameters were chosen to determine how the decision tree would model relationships and do it quickly. The reason for this is that gini is computationally less expensive. The results were as follows:

	Training	Testing
Accuracy	96.72%	95.88%
Precision	99.46%	100.00%
Recall	93.71%	91.44%
F1 Score	96.50%	95.53%

Table 2: Decision Tree Training and CV curve

Decision Tree Classifier CV Accuracy: 0.97 +/- 0.0

From the above results we can determine that the model was not overfitting.

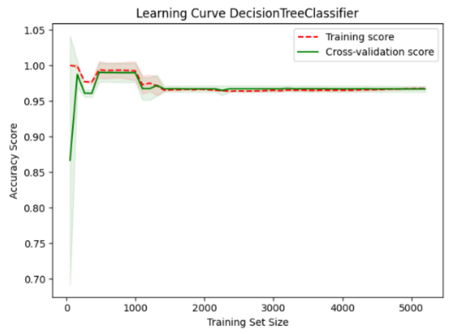
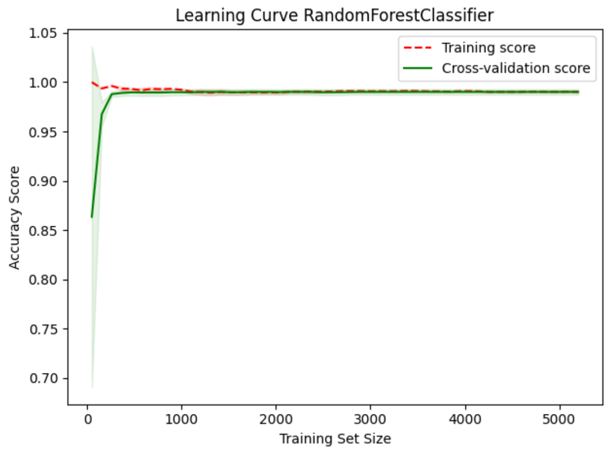


Fig.4: Decision Tree Training and CV curve

Random Forest Classifier:
This model was trained with initial parameters {criterion="log_loss", max_depth=4} (Fig.5)(Table 3). The results were as follows:
Random Forest Classifier CV Accuracy: 0.99 +/- 0.00
From the above results, the precision being 100% is suspicious as it seems to be the case that every instance the model predicted as positive(poisonous) was indeed positive, low recall score also indicates that model is not capturing all the actual positive test cases.



	Training	Testing
Accuracy	99.02%	99.02%
Precision	100.00%	100.00%
Recall	97.96%	97.96%
F1 Score	98.97%	98.97%

Table 3: Random Forest Train vs CV score

Fig.5: Random Forest Train vs CV score
From the above graph we can also see that the cross validation and testing scores converge.

Support Vector Machine:
This model was trained with initial parameters {kernel="linear", C=0.001}(Fig.6)(Table 4)., considering the large training split of 80% and testing of 20%, it made sense to use C as 0.001 considering the aim was to avoid overfitting or being sensitive to outliers.
SVC Classifier CV Accuracy: 1.00 +/- 0.0
The results are as follows:

	Training	Testing
Accuracy	99.94%	99.75%
Precision	100.00%	100.00%
Recall	99.87%	99.49%
F1 Score	99.94%	99.74%

Table 4: SVC curve for Training vs CV score

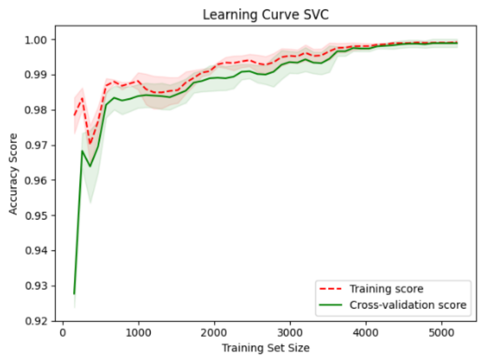
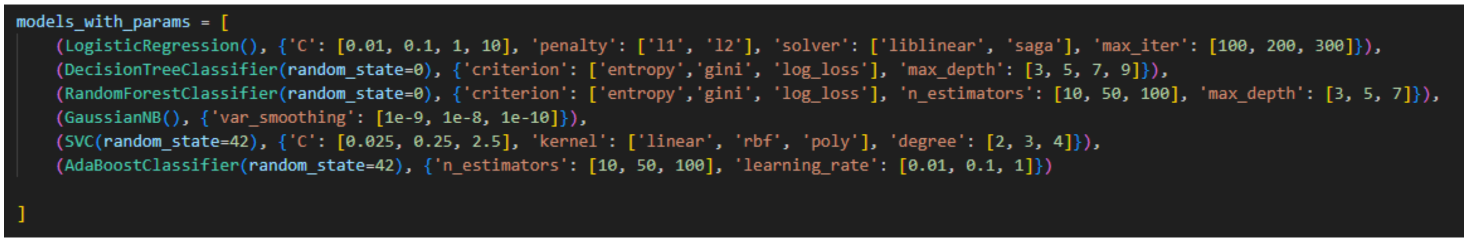


Fig.6: SVC curve for Training vs CV score
From the above results, we can say that the model learned well since it performed better on testing, and it can be further improved via a hyperparameter search.

B. Hyperparameters

Hyperparameters are parameters (Fig.7) 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 behaviour of the learning algorithm. The prefix 'hyper' indicates that they operate at a



```
models_with_params = [
    (LogisticRegression(), {'C': [0.01, 0.1, 1, 10], 'penalty': ['l1', 'l2'], 'solver': ['liblinear', 'saga'], 'max_iter': [100, 200, 300]}),
    (DecisionTreeClassifier(random_state=0), {'criterion': ['entropy','gini', 'log_loss'], 'max_depth': [3, 5, 7, 9]}),
    (RandomForestClassifier(random_state=0), {'criterion': ['entropy','gini', 'log_loss'], 'n_estimators': [10, 50, 100], 'max_depth': [3, 5, 7]}),
    (GaussianNB(), {'var_smoothing': [1e-9, 1e-8, 1e-10]}),
    (SVC(random_state=42), {'C': [0.025, 0.25, 2.5], 'kernel': ['linear', 'rbf', 'poly'], 'degree': [2, 3, 4]}),
    (AdaBoostClassifier(random_state=42), {'n_estimators': [10, 50, 100], 'learning_rate': [0.01, 0.1, 1]})

]
```

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.[7]

Fig.7: Hyperparameters used in tuning.

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 in order to find the best possible parameters.

Logistic regression

For logistic regression, the hyperparameters tuned were:
C: Inverse of regularization strength explored were [0.01, 0.1, 1, 10].
Smaller C values apply more regularization, which can mitigate overfitting. However, excessively small C values may lead to underfitting.
Penalty: Norm used in penalization considered were ['l1', 'l2'].
L1 regularization tends to produce sparser models by driving certain parameters to 0. On the other hand, L2 regularization does not, but instead shrinks them toward 0.
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 to [100, 200, 300**].**
Higher maximum iterations allow optimization algorithms more attempts to find the minimum cost function and converge to the solution.
Best parameters for Logistic Regression: {'C': 10, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}.
They did not help increase the accuracy of the model.

Decision Tree Classifier

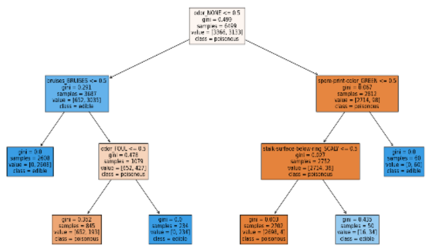


Fig.8: Decision Tree Representation

For Decision tree (Fig.8), the hyperparameters tuned were:
Criterion: A function to measure the quality of a split. The values explored 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 due to noise or outliers in the training set.
Best parameters for Decision Tree Classifier: {'criterion': 'entropy', 'max_depth': 5}
These parameters were optimal as the model achieved perfect accuracy with them.

Support Vector Machine

For SVC, the hyperparameters tuned were:
C: This parameter balances between accurately classifying training examples and maximizing the margin of the decision function. Parameters explored were ‘[0.025, 0.25, 2.5]’.
Kernel: The choice of kernel affects the decision boundary created by the SVM. A linear kernel can model linear decision boundaries, while the Radial Basis Function (rbf) and polynomial (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 means that the model can handle more complex relationships, but the risk of overfitting increases. Lower degrees are less computationally expensive and may generalize better. Parameters explored were [2, 3, 4].
Best parameters for SVC: {'C': 2.5, 'degree': 2, 'kernel': 'linear'}
These parameters were found to be optimal, resulting in a model that achieved perfect accuracy. Despite the relatively small C value, the 'linear' kernel was chosen, which typically results in a larger margin and a simpler decision function. The 'degree' parameter set to 2 indicates a polynomial kernel, and the combination of these parameters led to a high level of accuracy.

Random Forest Classifier

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’].
Explored values included 'entropy,' 'gini,' and 'log_loss.' 'Gini' tends to be faster to compute and may result in broader trees, while 'Entropy' can produce more balanced trees but is computationally expensive due to logarithmic calculations.
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]’.
Best parameters for Random Forest Classifier: {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 50}.
Hence, with the hyperparameters {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 50} in place, it is expected that the Random Forest Classifier will excel in generating balanced trees, effectively capturing intricate patterns, and ensuring resilience and stability when encountering outliers and noise during training.

VII. Comparison of Results

The logistic regression model was optimized with hyperparameters C=10, max_iter=100, penalty='l1', and solver='liblinear', although these parameters did not improve model accuracy. In contrast, the Decision Tree Classifier, with parameters criterion='entropy' and max_depth=5, achieved perfect accuracy and proved to be optimal. The SVM was fine-tuned with parameters C=2.5, degree=2, and kernel='linear,' resulting in a highly accurate model. Despite the relatively small C value, the 'linear' kernel choice, combined with degree=2, contributed to this success. Lastly, the Random Forest Classifier, configured with criterion='entropy,' max_depth=7, and n_estimators=50, is expected to excel in generating balanced trees and capturing complex patterns while remaining resilient to noise and outliers during training.

VIII. Conclusion

In summary, this study undertook an exhaustive investigation into machine learning techniques applied to the classification of gilled mushrooms' edibility. The processed dataset, encompassing 8,416 samples, served as the foundation for enhancing model performance. Five distinguished classification models, comprising the Random Forest Classifier, Logistic Regression, Support Vector Machine, Decision Tree Classifier, and Gaussian Naive Bayes, were subject to rigorous evaluation. The outcomes unequivocally demonstrated the Decision Tree Classifier's flawless accuracy, achieved through the configuration of hyperparameters, specifically, criterion='entropy' and max_depth=5. Concurrently, the Support Vector Machine exhibited remarkable performance, following parameter fine-tuning involving C=2.5, degree=2, and kernel='linear.' Lastly, the Random Forest Classifier, after optimization with criterion='entropy,' max_depth=7, and n_estimators=50, is poised to excel in constructing well-balanced trees capable of capturing intricate patterns. These discoveries provide invaluable insights into the untapped potential of machine learning within the realm of mycology.
Happy foraging.

IX. Future Work

In the context of future work, this Python script lays the foundation for several potential enhancements and extensions. The Python script utilizes TensorFlow/Keras to construct, train, and evaluate a neural network model for the classification of mushrooms as either poisonous or non-poisonous, based on a dataset containing various mushroom features. This script serves as a foundation for using deep learning techniques for binary classification tasks in the context of mushroom toxicity prediction.

X. Description of Student Participation

The successful completion of our project was a true collaboration of teamwork. Every team member actively participated and contributed their expertise to achieve our common goal. Our journey began with the selection of a dataset, and as with any project, we encountered some initial issues with the data. These challenges prompted us to create a dedicated repository folder named "old_files," where the original dataset, along with its imperfections, was carefully preserved.

This project was split off into several stages of developments which include, selection of the dataset, preprocessing the dataset that was selected, encoding the data so that it could be fed into the model, selection and hyper parameter tuning of said models, and in between gathering valuable metrics and insights to add to the report.

To make our work even more accessible and transparent, we also established a GitHub page for our project. Our GitHub page became an invaluable resource for the team project. It allowed us to showcase our progress, share our codebase, and provide a platform to work collectively on a single project.

Link to our complete project available on GitHub.
https://github.com/nealj1/comp-4730-1st-project

References

[1] - Concerns raised as “Magic mushroom” store opens in windsor, https://windsorstar.com/news/local-news/concerns-raised-as-magic-mushroom-store-opens-in-windsor (accessed Oct. 16, 2023).

[2] - “Mushroom - UCI Machine Learning Repository.” https://archive.ics.uci.edu/dataset/73/mushroom (accessed: Oct. 15, 2023).

[3] - Ottom, M. A., Alawad, N. A. & Nahar, K. M. Classification of mushroom fungi using machine learning techniques. Int. J. Adv. Trends Comput. Sci. Eng. 8, 2378–2385 (2019).

[4] - Hossin, M. & Sulaiman, M. A review on evaluation metrics for data classification evaluations. Int. J. Data Min. Knowl. Manag. Process 5, 1 (2015).

[5] - Whipple, T. (2020, October 7). Mushroom labelling. Medium. https://mrtjwhipple.medium.com/mushroom-labelling-db941388f0d0

[6] - Pinky, N. J., Islam, S. M., & Alice, R. S. (2019). Edibility detection of mushroom using ensemble methods. International Journal of Image, Graphics and Signal Processing, 11, 55-62.

[7] - “Parameters, Hyperparameters, Machine Learning | Towards Data ..” https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac (accessed: Oct. 15, 2023).