Identify the Quality of Mushrooms Using Machine Learning

Name:	**Sushovit Nanda**
Registration No./Roll No.:	21283
Institute/University Name:	IISER Bhopal
Program/Stream:	EECS
Problem Release date:	August 17, 2023
Date of Submission:	November 19, 2023

1 Introduction

A mushroom is the fleshy, spore-bearing fruiting body of a fungus. Fungi belong to a separate kingdom in the classification of living organisms, distinct from plants, animals, and bacteria. While mushrooms can vary widely in appearance, they typically have a cap on top of a stalk. The cap may have gills, pores, or other structures on it is underside, where the mushroom produces and releases spores for reproduction. This paper contains various methods and techniques to extract relevant parameters from the Mushroom Dataset and use them to classify mushrooms as "Edible" or "Poisonous." It provides methods for data classification, data cleaning and various models for classification.

- **mushroom_trn_data.csv:** This file contains training feature vectors with 7311 data points and 22 features.
  The feature class 'stalk-root' has 2235 missing values.
- **mushroom_trn_class_labels.csv:** This file contains class labels of all training data points.
  It has 3787 edible labels and 3524 poisonous labels
- **mushroom_tst_data.csv:** This file contains test feature vectors with 813 data points and 22 features.
  The feature class 'stalk-root' has 245 missing values.

2 Methods

The **Github Page** of the project can be found here The project has been carried out in the following order :

2.1 Data Processing

There are 3 data files as discussed in the above section. All 3 were imported to form databases using the Pandas module in Python. *Mushroom_train_data* contains the training database. *Mushroom_train _label* contains the class label database and the true_test contains the testing database. The models are supposed to predict the class labels for the *true_test*.

2.2 Data Visualization

- The shape of all 3 databases have been printed in the code using the .shape code.
  The shape of the *Mushroom_train_data* is (7311, 22). Shape of *Mushroom_train _label* is (7311,1). Shape of *true_test* is (813,22)
- A countplot has been plotted using the *sns.countplot* to visualize how many counts of each unique feature belongs to either class label - 'Edible' or 'poisonous.'
  Looking through this countplot shows us that the 'stalk-root' feature class has missing values as '?'.

2.3 Missing Data Feature Removal

1. From the code, we find out that there are 2480 missing data points out of the total 8124 data points in the entire 'stalk-root' feature class.

2. Since this is still a categorical data as of now, the imputation method available to us to fill in the missing values is to put the mode of the entire feature class but since almost one-third of the data points are missing, imputing the missing values with mode will introduce a bias which will result in a loss of information and impact the model performance.

3. Hence to tackle this issue, it is better to drop the 'stalk-root' feature class as it is of no use due to it is high variability of missing values.

4. We also removed the 'veil-type' feature class as it has only 1 unique feature so it will be correlating with every other feature and will be redundant for our model.

2.4 Conversion into Numerical Data

Before beginning to conversion the data, we see that there are 2 extra unique features in our *Mushroom_train_data* in its 'cap-shape, cap-surface' feature class compared to the *true_test*, hence we merge them both and convert them together to avoid unequal columns in the later stages of the project.

To convert the categorical database into numerical one, we do OneHot Encoding using the get_dummies function from the Pandas library. OneHot encoding is used to represent categorical variables as binary vectors. It converts each unique feature in a categorical variable into a new binary column and denotes the presence of the category with a 1 and the absence with a 0. For each categorical variable, a binary vector is created with a length equal to the number of unique features. Each unique feature is then represented by a unique combination of 0s and 1s.

After the encoding, the merged dataset is spliced back to *Mushroom_train_data* and *true_test*. The class labels in *Mushroom_train _label* have been converted to numerical data by 'Poisonous' being replaced with 0 and 'edible' being replaced with 1.

2.5 Correlation Calculation

Correlation is a statistical measure that describes the degree to which two variables change together. It indicates the strength and direction of a linear relationship between two variables.

Correlation was calculated for each of the feature in the *Mushroom_train_data*. Those with highly correlated variables in a model may not provide additional information and can lead to multi-collinearity issues.

2.6 Feature Selection

For feature selection, the features correlating a particular threshold were dropped. The threshold assigned for this project is 0.8, that is to say features that show more than 80% correlation.

A high threshold has been chosen for this project because correlation does not imply causation. A strong correlation between two variables does not necessarily mean that one causes the other. It could also be due to a third variable, coincidence, or other factors.

2.7 Modelling

The *Mushroom_train_data* has been split into *x_train*, *y_train*, *x_test*, *y_test*; using the train_test_split function having an 80/20 split.

The following models were performed on the training set and the corresponding confusion matrix and classification report are made for each of the training models. The models run on the dataset were:

1. Logistic Regression

2. Decision Tree
3. Random Forest Classifier
4. K-Nearest Neighbors Classifier
5. Support Vector Classifier

GridSearchCV is being used for the hyperparameter tuning of all the above models. It also uses Cross-Validation to obtain a robust estimate of a model's performance. It divides the dataset into multiple folds, trains the model on subsets of the data, and evaluates performance on the remaining portions. This process is repeated for each combination of hyperparameters. This avoids the issue of over-fitting in the model.

Once all combinations are evaluated, GridSearchCV identifies and returns the hyperparameters that resulted in the best performance according to the specified evaluation metric. Using these hyperparameters, we predict the class labels of *x_test*.

3 Experimental Analysis

The trained model has been tested on the *x_test* data points and predictions of class labels have been made by all the 5 trained models. Below is the performance of each model when the predicted labels have been compared with the true labels from *y_test*.

Table 1: Performance Of Different Classifiers Using All Features

Classifier	Precision	Recall	F-measure
Logistic Regression	1.00	1.00	1.00
Decision Tree	1.00	1.00	1.00
K-Nearest Neighbor	1.00	1.00	1.00
Random Forest	1.00	1.00	1.00
Support Vector Machine	1.00	1.00	1.00

Table 2: Confusion Matrices of Different Classifiers

Actual Class	Predicted Class	
	Poisonous	Edible
Poisonous	715	1
Edible	0	747

Decision Tree

Actual Class	Predicted Class	
	Poisonous	Edible
Poisonous	716	0
Edible	0	747

K-Nearest Neighbor

Actual Class	Predicted Class	
	Poisonous	Edible
Poisonous	716	0
Edible	0	747

Logistic Regression

Actual Class	Predicted Class	
	Poisonous	Edible
Poisonous	716	0
Edible	0	747

Random Forest

Actual Class	Predicted Class	
	Poisonous	Edible
Poisonous	716	0
Edible	0	747

SVM

As validated by our code, every model is giving us an accuracy and F1-score of 100%. All models are correctly classifying the data points in *x_test*, except *Decision_tree* where 1 poisonous mushroom has been incorrectly classified as edible.

The best model proposed for this problem statement is *Support Vector Classifier*. Using this model, we predict the class labels of *true_test* database.

4 Discussions

The proposed model can handle both linear and non-linear decision boundaries[1]. By using different kernel functions[2] (e.g., radial basis function, polynomial), SVCs can capture complex relationships in the data. It aims to find a decision boundary (hyperplane) that maximizes the margin between different classes. A larger margin often leads to a more robust and generalizable model and less prone to overfitting. They perform well in high-dimensional spaces, making them suitable for datasets with many features.

SVCs too have their limitations. Its computationally expensive, especially with large datasets. The choice of the kernel (e.g., linear, polynomial, radial basis function) can significantly impact the performance[3] as each different kernel has its own mathematical complexities. Using it on large datasets may lead to memory issues. SVC may not handle datasets with NaN or infinite values, therefore thorough data processing needs to be done.

References

[1] Sancho Salcedo-Sanz, José Luis Rojo-Álvarez, Manel Martínez-Ramón, and Gustavo Camps-Valls. Support vector machines in engineering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):234–267, 2014.

[2] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[3] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.