Mushroom Classification

Authors: Marko Erdelji (SV49/2020), Dušan Rožić (SV80/2020), Ognjen Radovanović (SV74/2020)

1. Motivation

The motivation for this work is to utilize a comprehensive and realistic dataset of mushrooms for the task of binary classification, distinguishing between edible and poisonous varieties. This work is driven by the potential applications in food safety and mycology, where accurate classification can prevent mushroom poisoning incidents.

2. Research questions

The specific problem we aim to address is the classification of mushrooms into edible and poisonous categories using a comprehensive and simulated dataset. The dataset, available from the UCI Machine Learning Repository, was created to support binary classification tasks and includes 61,068 instances with 20 features each. It is inspired by the Mushroom Data Set of J. Schlimmer and aims to provide a more extensive and realistic dataset for machine learning applications [1].

3. Related work

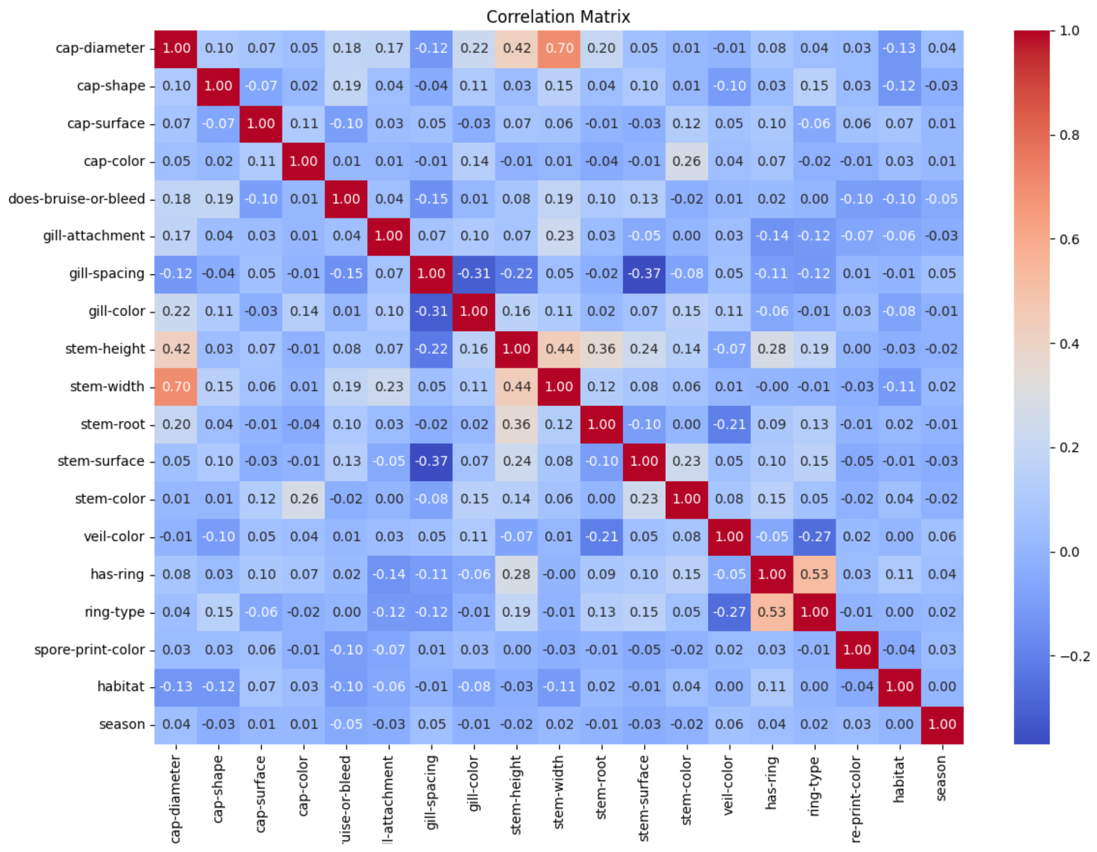
Previous studies have predominantly used the 1987 mushroom dataset from the UCI Machine Learning Repository [2]. While this dataset has been instrumental in demonstrating classification tasks, it is limited in scope and diversity. Recent work by Wagner et al. (2021) introduced a more comprehensive dataset that captures a broader range of mushroom species and attributes [1]. Our work builds on these efforts by applying machine learning techniques to this enhanced dataset.

4. Methodology

The solution was implemented in the Python programming language using a *flask* [3] server, *react* [4] frontend, the *pandas* [5] preprocessing library and helper libraries (*sckit-learn* [6], *matplotlib* [7] and *seaborn* [8]). Our approach to solving the problem includes the following steps:

1. **Exploratory Analysis:** We looked at the data to understand the structure and we noticed that a lot of the values are missing for some columns, we also noticed that the numerical columns have all their values present and are in the range of [0,20], we also didn't use the columns that had more than 90% of their values missing.
2. **Data Preprocessing:** We noticed that the categorical columns have a lot of missing data so we used the *SimpleImputer* class with the *most_frequent* strategy to fill in all the missing values. From the classes of the categorical columns we noticed that the data in itself is nominal, we tried using one-hot encoding, however the problem with this approach is that it produces a curse of dimensionality (more than 100 columns). We settled for *LabelEncoder* which at most produces values from 1-13 and we achieved good results with this approach. After that we used *StandardScaler* to normalize the data, however since the data is already numerically in the [0,20] range it doesn't change the final result.

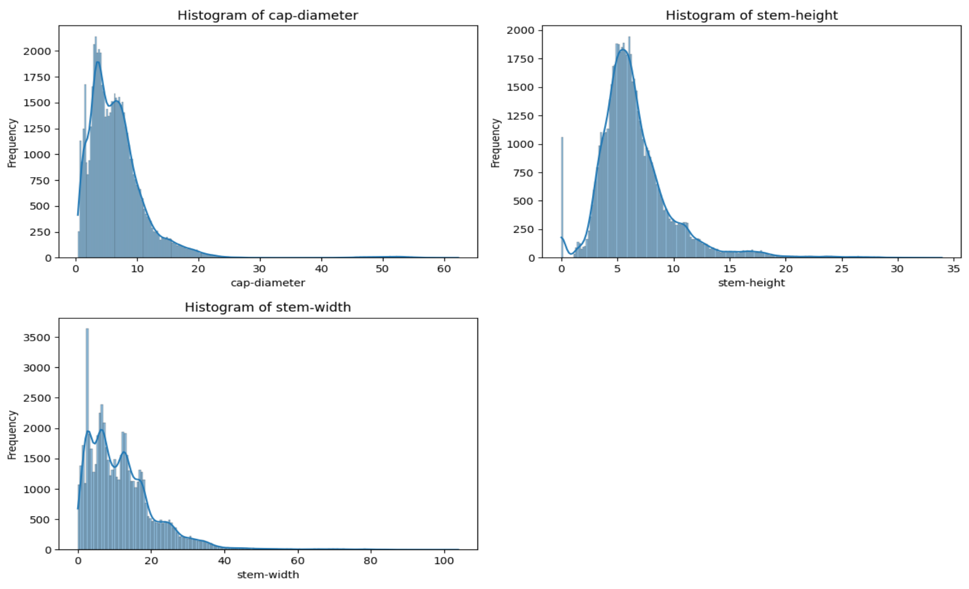
3. **Feature Selection:** To handle multicollinearity amongst the features, we used a correlation matrix to check which features are correlated more than 80%.



Picture 1. Correlation matrix

The correlation between the cap-diameter and stem-height was 0.7 but this wasn't enough to have any significant impact on the final result.

4. **Outliers:** Since the numerical data was synthetically generated to not have outliers, there aren't any outliers to remove for the continuous variables as can be seen from the histograms.

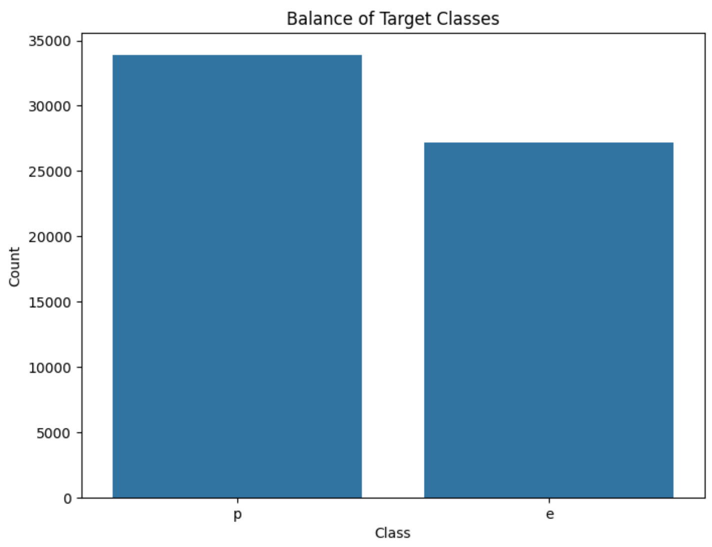


Picture 2. Histograms of numerical data

5. **Data splitting:** We split the data into train and test (80% train, 20% test).
6. **Models:** We fitted multiple models with the training data. The models we used are: Random Forest [9], Logistic Regression [10], KNN [11], Naïve Bayes [12], Gradient Boosting [13], Decision Tree [14]

5. Discussion

To evaluate the previously mentioned models, we used the **macro F1 score** [15]. The score was used since the target classes are relatively balanced as can be seen in the graph.



Picture 3. Distribution of target classes (e-edible, p-poisonous)

We used the **precision** and **recall** metrics as well.

Model	Precision	Recall	Macro F1
Random forest	0.9999	0.9999	0.9999
Logistic Regression	0.6502	0.6402	0.6402
KNN	0.9999	0.9999	0.9999
NaiveBayes	0.6513	0.6301	0.5995
GradientBoosting	0.9218	0.9209	0.9213
DecisionTree	0.9947	0.9948	0.9948

Table 1. Performance of models

6. References

[1] Wagner,Dennis, Heider,D., and Hattab,Georges. (2023). Secondary Mushroom. UCI Machine Learning Repository. https://doi.org/10.24432/C5FP5Q.
[2] Mushroom. (1987). UCI Machine Learning Repository. https://doi.org/10.24432/C5959T.
[3] Flask documentation (https://flask.palletsprojects.com/en/3.0.x/)
[4] React documentation (https://react.dev/)
[5] Pandas documentation (https://pandas.pydata.org/)
[6] Scikit-learn documentation (https://scikit-learn.org/stable/index.html)
[7] Matplotlib documentation (https://matplotlib.org/)
[8] Seaborn documentation (https://seaborn.pydata.org/)
[9] Random forest classifier (https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/?ref=gcse_ind)
[10] Logistic regression (https://www.geeksforgeeks.org/understanding-logistic-regression/)
[11] KNN (https://www.geeksforgeeks.org/k-nearest-neighbours/)
[12] Naïve bayes (https://www.geeksforgeeks.org/naive-bayes-classifiers/)
[13] Gradient boosting (https://www.geeksforgeeks.org/ml-gradient-boosting/)
[14] Decision tree (https://www.geeksforgeeks.org/decision-tree/)
[15] F1 score (https://www.geeksforgeeks.org/f1-score-in-machine-learning/)