# A Comparative Analysis of Mushroom Edibility Prediction
Data Set: **Mushroom**

**Fangqi Yuan**, fangqiyu@usc.edu
**Yuan Chang**, yuanchan@usc.edu

May 8, 2023

## 1 Abstract

This report presents a comprehensive analysis and evaluation of various classification methods applied to the mushroom data set. The primary objective of this project is to train different models and achieve high classification accuracy on the test set in order to accurately distinguish between edible and poisonous mushrooms. To achieve this goal, the project explores three distinct approaches for handling the data set, which comprises a mix of both numerical and categorical features: one-hot encoding, label encoding, and statistical feature engineering.

One-hot encoding serves as a relatively straightforward approach to manage categorical features, while improving results using label encoding requires more effort, and feature engineering is used to reversely transform the data set into a purely statistical one. In this study, we employ a range of classification methods, including Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), neural networks, and several other classifiers beyond the scope of the EE559 course.

Throughout the project, we systematically analyze the performance of each classification method on the mushroom dataset under different preprocessing and feature engineering strategies. Our results demonstrate that the dataset, albeit containing a mixture of feature types, is one that could get classified with a relatively high accuracy. By employing the appropriate preprocessing techniques and classification models, it is possible to achieve robust and reliable classification results on this dataset.

## 2 Introduction

The primary objective of this project is to develop an effective classification model capable of accurately distinguishing between edible and poisonous mushrooms based on their features. The dataset is composed of a mix of both numerical and categorical features, which presents a unique challenge in the preprocessing and feature engineering stages. To achieve a high classification accuracy on the test set, several key aspects need to be considered:

**Preprocessing and Feature Engineering**: A crucial step in handling the mixed dataset is to preprocess the data effectively. This includes dealing with missing values, encoding categorical features appropriately, and potentially creating new features that may improve the model's performance. Techniques such as one-hot encoding, label encoding, or aggregating categorical variables based on their statistical properties can be employed to transform the data into a suitable format for machine learning algorithms.

**Model Selection and Evaluation**: Various classification algorithms, including both parametric and distribution-free classifiers, should be explored as required. These models should be evaluated using appropriate performance metrics, such as accuracy, precision, recall, and F1 score.

**Hyperparameter Tuning and Model Optimization**: As different approaches are required in this project, equal efforts must be made to optimize each of the models by adjusting available parameters.

By addressing these key points and carefully selecting the appropriate techniques, models, and evaluation methods, this project aims to compare robust classification models capable of accurately distinguishing between edible and poisonous mushrooms.

Before work on this project formally begun, some code on Kaggle has been browsed by us [1][2], most of which put their focus on the second route in this project, i.e. starting with label encoding. In this report, the focus is also on the second and the third route (See 3.2 and 3.3). A small part of code was based on their ideas, but most was developed on the basis our own discussions and the guide written by our TA.

# 3 Approach and Implementation

As is mentioned above, the data set is composed of categorical features as well as numerical features, which is not convenient for any kind of classifier.

To demonstrate that, we could use a set of histograms(Figure 1) on each of the categorical features. Denote poisonous mushrooms as p, edible as e:

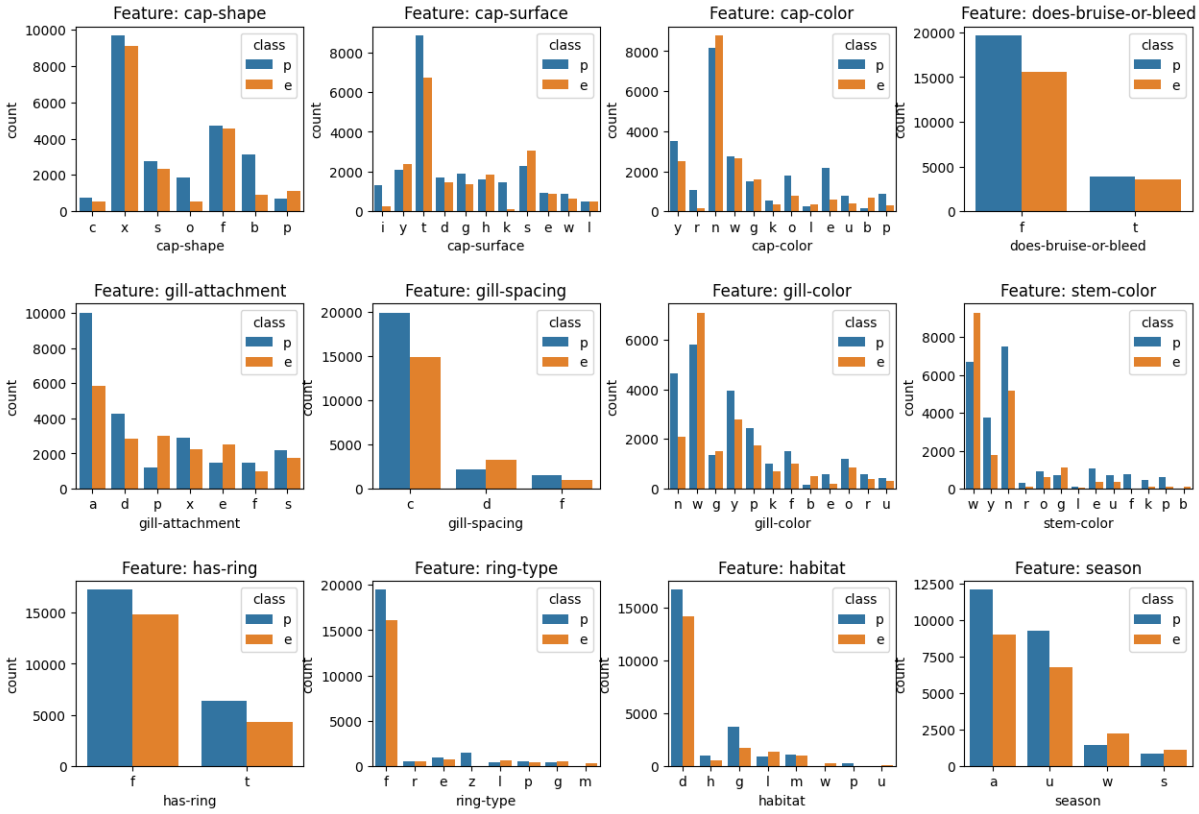


Figure 1: Distribution of Categorical Features for Poisonous and Edible Mushrooms

It could be seen that none of the features could be a decisive factor for the p or e class. Some quantitative analysis will be mentioned later.

Based on these results, three routes are introduced. Within each route, 4 or more specific approaches are employed according to the requirements as a team project.

## 3.1 One-hot Encoding

One-hot encoding is a plain method to transfer categorical features into numerical ones. In this project, `pd.get_dummies` is applied to change the data set into this form.

Before evaluating each specific approach, performance of some reference systems should be first evaluated as a basic standard.

### 3.1.1 Reference Systems

A random classifier, as its name implies, assigns random results to each data point of the test set according to the frequency in the training set. The result in terms of Accuracy and F1-Score is:

`Accuracy:` `0.5052671797390972`

`F1-score:` `0.5552502453385671`

A nearest mean classifier classifies points based on the respective distance between two means. The result is:

`Accuracy:` `0.6168331422957262`

`F1-score:` `0.6536412078152753`

### 3.1.2 SVM

A support vector machine is directly called using `svm.SVC()` in this part. After some trials, the rbf kernel with $C = 100$ presents the best result.

The result is:

`Accuracy:` `0.9960700835107253`

`F1-score:` `0.9965119658947776`

Confusion Matrix:

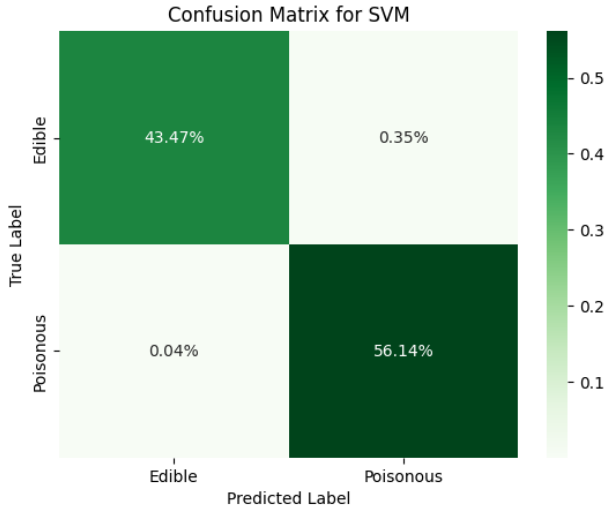


Figure 2: Confusion Matrices for SVM

### 3.1.3 Probabilistic Methods

We will take a k-NN classifier here. Since the data set is transferred from a mostly categorical one, when calculating distances, the Manhattan distance is assumed to be a better rubric than Euclidean ones. The result is:

`Accuracy:` `0.9993995960919164`

`F1-score:` `0.9994659416419868` After that, PCA is also tried in hope to see if an accuracy of 100% could be reached, with the set having been standardized. The parameter is `n_components=0.97`, meaning the variance is kept as 0.97.

The result is very close but still less than fully accurate, which is a little pity.

`Accuracy:` `0.9994541782653785`

`F1-score:` `0.9995142801632019`

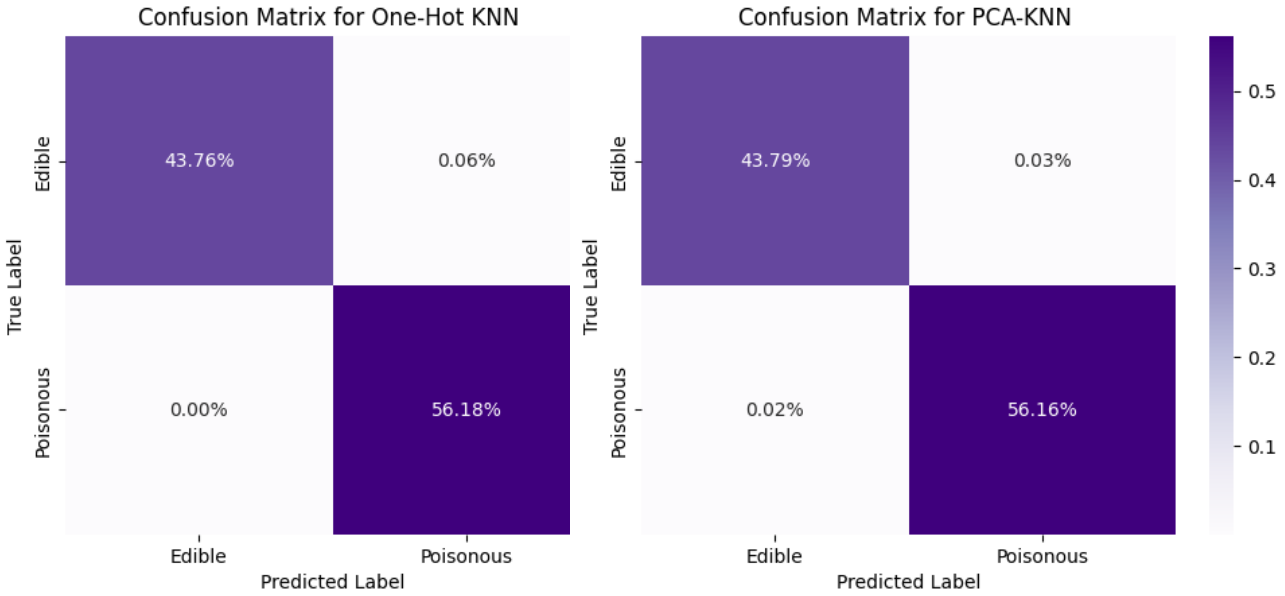


Figure 3: Confusion Matrices for k-NN

### 3.1.4 Non-probabilistic Methods

It is reckoned that non-probabilistic methods learnt is not a valid choice for this data set, since the dimension is high enough. The result is within our expectation.

```
Accuracy:  0.6149227662245511
F1-score:  0.6910985594815885
```

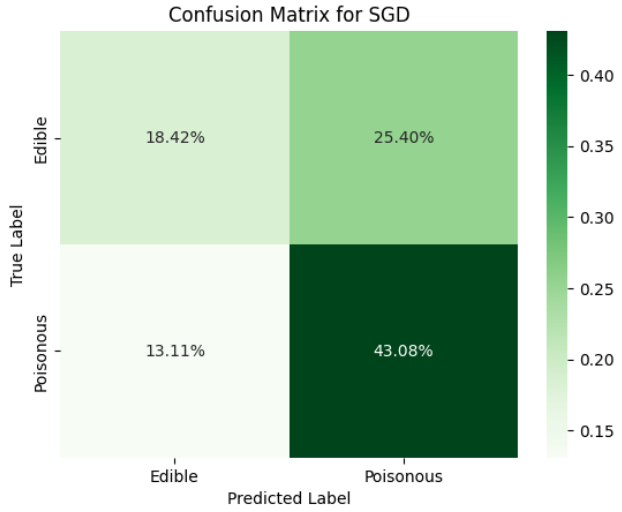


Figure 4: Confusion Matrices for SGD

## 3.2 Label Encoding

As a matter of fact, this is one of the most common measures to deal with this mushroom data set, as is mentioned above. Namely, each categorical feature as a string is labeled as integers in a certain order. This set could then be processed in its original dimension.

As for its implementation, `sklearn.preprocessing.LabelEncoder()` is applied to the original data set.

### 3.2.1 Reference Systems

Random classifier: The result in terms of Accuracy and F1-Score is
`Accuracy: 0.5120899514218656`
`F1-score: 0.5598936536851953`
Nearest mean classifier: The result is
`Accuracy: 0.6014409693794007`
`F1-score: 0.6487396574947085`

### 3.2.2 SVM

A similar SVM with rbf kernel is used to train the categorical set. The result is far from good compared with 3.1.2.
`Accuracy: 0.7132252606298782`
`F1-score: 0.7545089243995887`

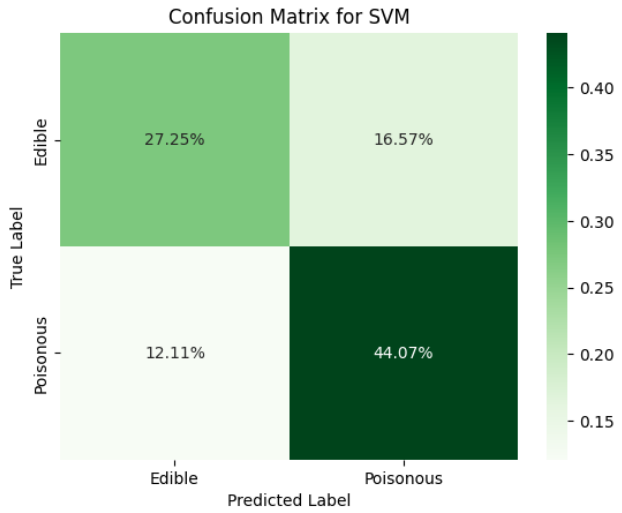


Figure 5: Confusion Matrices for SVM

### 3.2.3 Probabilistic Methods

A naive bayes classifier is first applied as a probabilistic approach, namely
`sklearn.naive_bayes.CategoricalNB()`.
Surprisingly, the result is not good as expected.
`Accuracy: 0.72572457835271`
`F1-score: 0.7519376018166559`
k-NN is then used again, but the result is still not so good as in one-hot encoding. After all, the integers are to tell features apart, but not the feature itself.
`Accuracy: 0.8625075050488511`
`F1-score: 0.8783032996763128`
Euclidean distance is then tried again, and it gets even worse:
`Accuracy: 0.7890944817422629`
`F1-score: 0.8131708732230926`

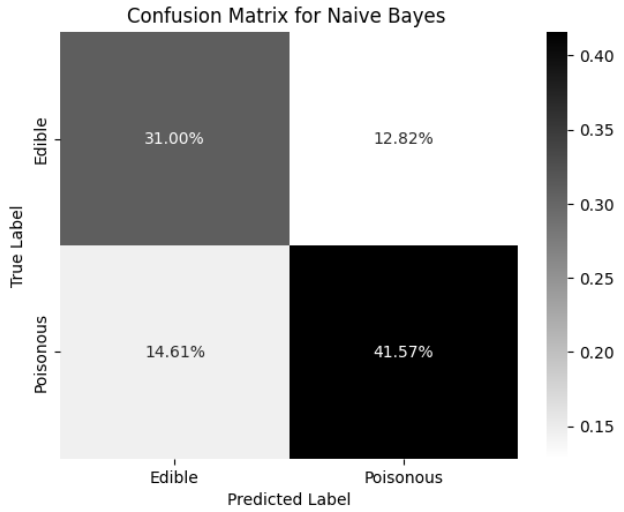


Figure 6: Confusion Matrix for Naive Bayes

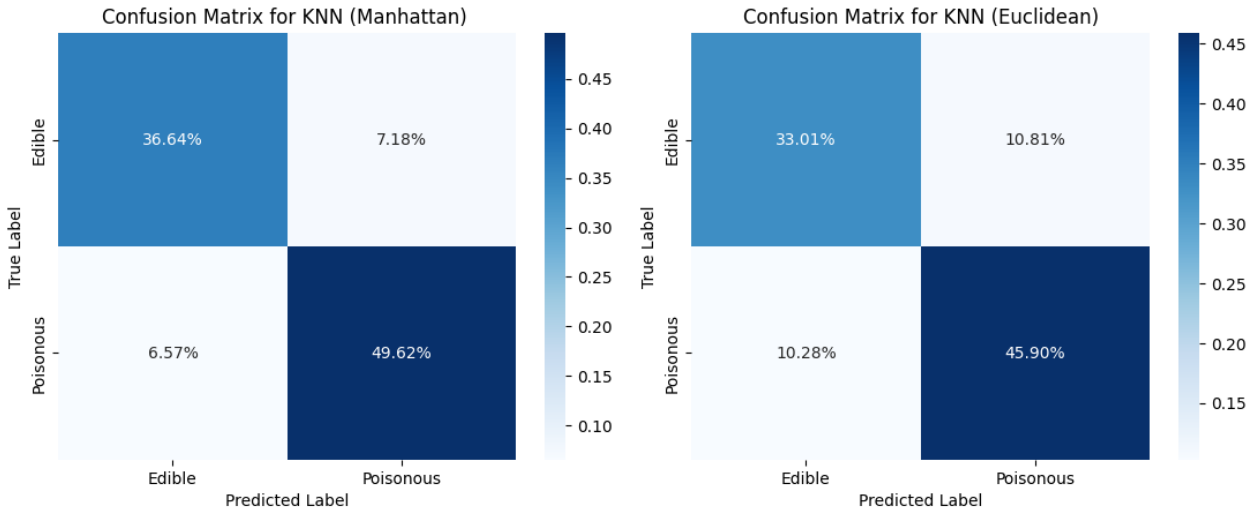


Figure 7: Confusion Matrices for k-NN

### 3.2.4 Neural Network

A neural network is built in this section to improve performance, since that in 3.2.3 is not so decent. In detail, a network composed of 2 ReLu layers (64 nodes) and a Sigmoid output layer (1 node) is constructed, with 50 epochs.

The result is as below:

```
Accuracy:  0.9715626876262212
F1-score:  0.9743992924180629
```

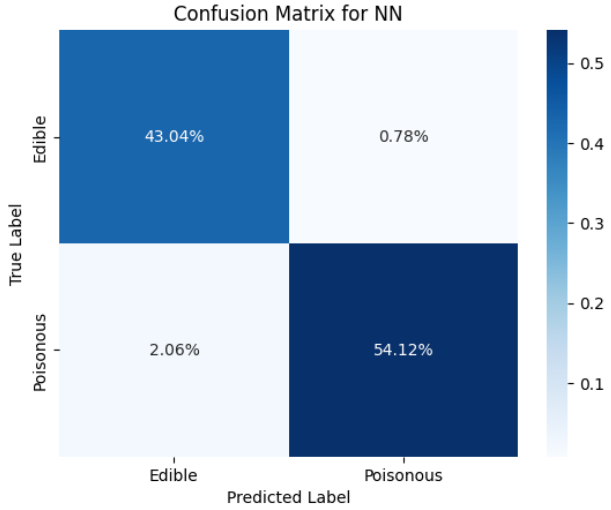


Figure 8: Confusion Matrices for Neural Network

### 3.2.5 Non-probabilistic Methods

In order not to make this section meaningless, a new classifier is introduced, the decision tree classifier.

Decision trees recursively split the input space based on feature values, creating a tree-like structure of decision nodes and leaf nodes. Each leaf node represents a class label.

The result is excellent, especially when it takes only 0.8 seconds to execute the cell!

```
Accuracy:  0.9893018940014191
F1-score:  0.9905314009661836
```

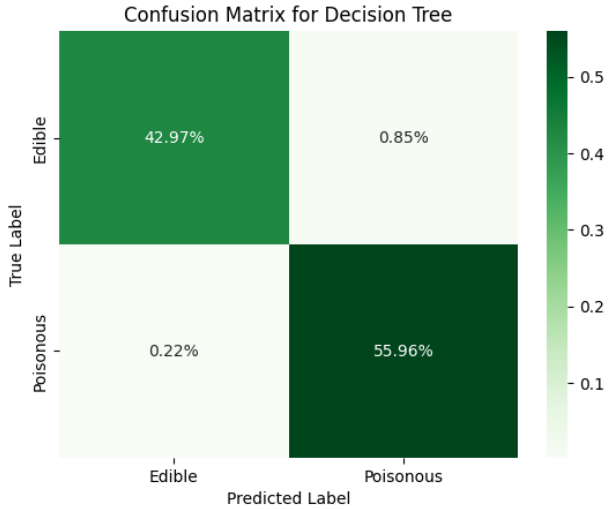


Figure 9: Confusion Matrices for Decision Tree

## 3.3 Feature Engineering

### 3.3.1 Overview

This approach is actually distinct from both of the two above. Simply put, it totally abandons categorical features, and replaces them with processed statistics, with `df.groupby()` playing an important part. This could divide the data into several groups by certain categorical features, and the min/max/value of a certain numerical one within each group could be added as three new numerical features to the set.

In summary, this route is in the opposite way of the above two.

The next problem is, what categorical features are to be added for group division? The Pearson correlation is studied on the label encoded data set, as below:

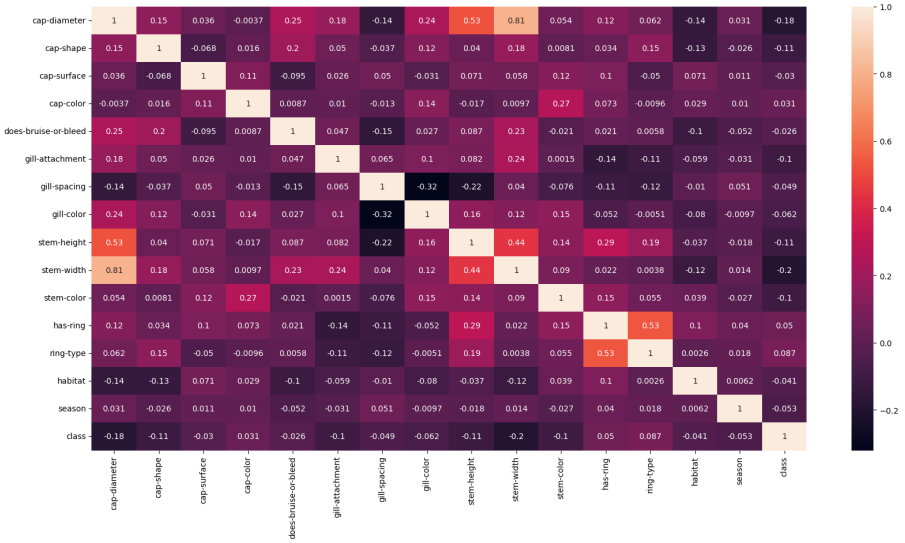


Figure 10: Pearson Correlation Matrix

Pearson correlation evaluates the linear relationship between two continuous variables, while the Spearman correlation concentrates on the monotonic relationship. Considering the features are categorical in essence, the Spearman coefficient is also studied:

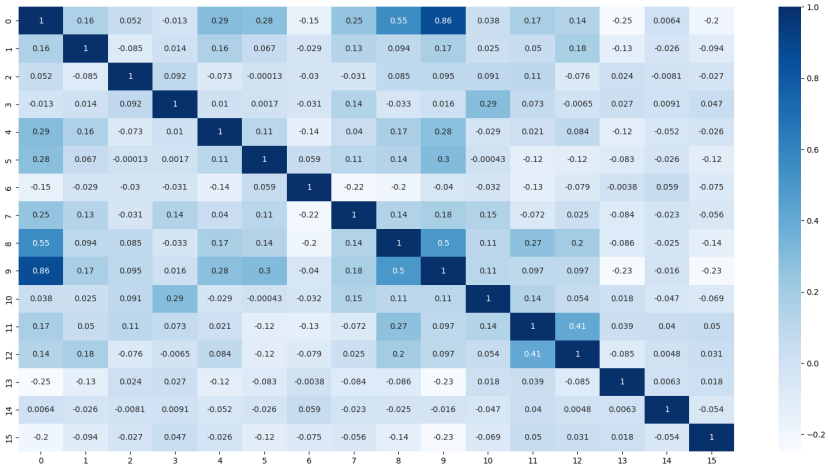


Figure 11: Spearman Correlation Matrix

Based on the results, three of the features `stem-color, gill-attachment, ring-type` are chosen as the final set in this part.

As an example,

```
grouped_color = X_merge.groupby('stem-color')
cap_diameter_min_scolor = grouped_color['cap-diameter'].min()
X_merge['cap-diameter-min-stem-color'] = X_merge['stem-color'].map(cap_diameter_min_scolor)
```

This code snippet adds the minimum value of cap diameter grouped by various stem colors to the dataset.

### 3.3.2 SVM

The result of SVM is decent, but still a little far from our expectation.

`Accuracy:` `0.8069974346378472`
`F1-score:` `0.8273774653388011`

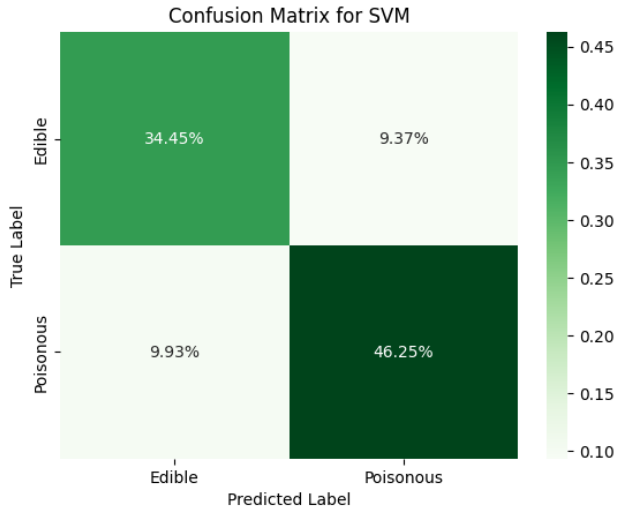


Figure 12: Confusion Matrix for SVM

### 3.3.3 k-NN

This measure of feature engineering seems quite effective when it comes to k-NN classifier. A 97+% result is the best proof.

`Accuracy:` `0.9773483980132089`
`F1-score:` `0.9798514346749526`

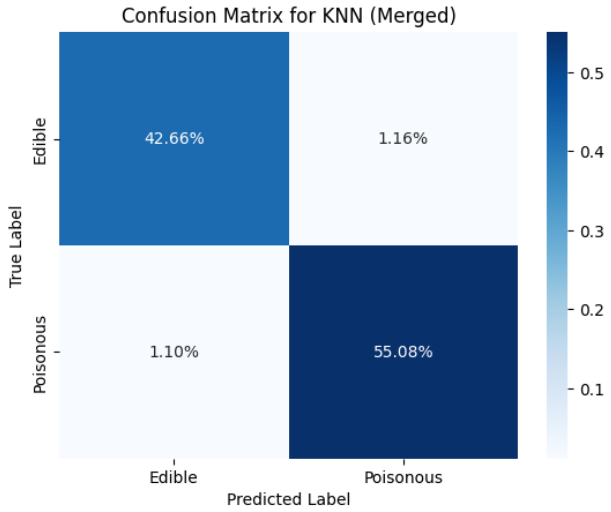


Figure 13: Confusion Matrix for KNN(Merged)

### 3.3.4 Neural Network

In this section the same network is applied again as a comparison.
The result surprisingly decreases compared to 3.2.4.

```
Accuracy:  0.9509852082309918
F1-score:  0.9554740182467274
```

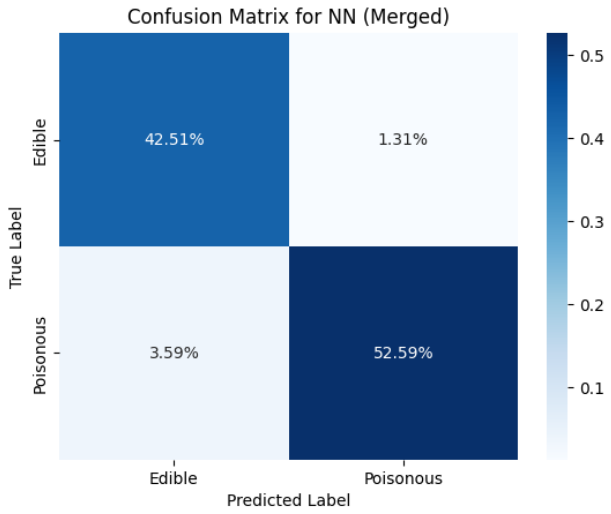


Figure 14: Confusion Matrix for NN(Merged)

### 3.3.5 Non-probabilistic

In this approach the same decision tree is used, but not as good as before. The possible reason will be discussed later.

```
Accuracy:  0.889525680912614
F1-score:  0.9039027632703446
```

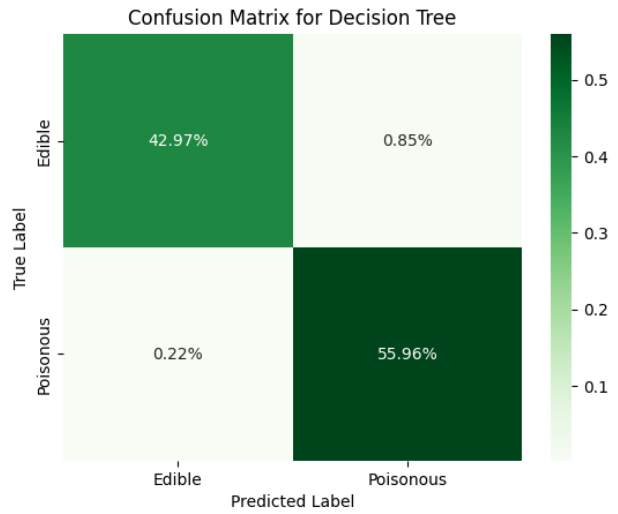


Figure 15: Confusion Matrix for Decision Tree

### 3.3.6 Feature Reduction

In this section, key feature selections will be carried out through a chi-squared test, rather than manual selection based on correlation or linear classifiers.

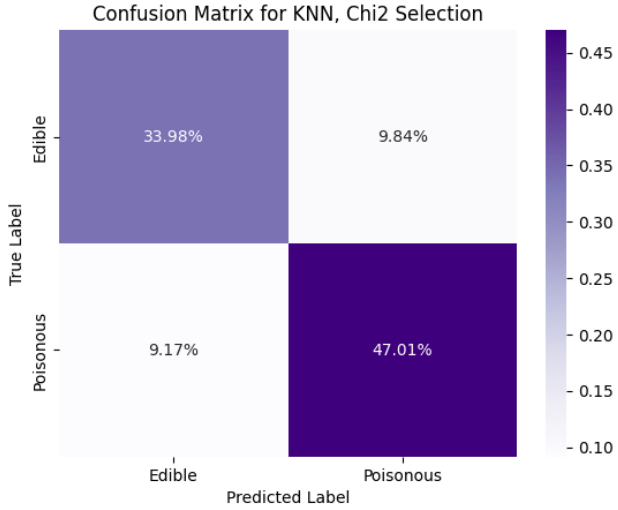


Figure 16: Confusion Matrix for KNN, Chi-squared Selection

To implement, `SelectKBest` and `chi2` from `sklearn.feature_selection` are imported. 8 features are chosen, with the final results listed below:

# 4 Analysis: Comparison of Results, Interpretation

We will analyze each of the methods one by one.

## 4.1 Non-probabilistic Methods

Linear models or anything learnt before midterm seem not so capable of dealing with this complex data set accurately. The dimension is too high for a SGD Classifier, so it is not strange that the performance is bad.

Out of the results above, We decide to introduce the decision tree classifier. This algorithm inherently handles categorical data well, as it constructs a tree structure by iteratively splitting the data based on the feature that provides the best separation of the target variable. To be honest, we are still not familiar enough with this one, but it does present a great result when it comes to split data based on categories. After the categories are replaced with certain numbers, its performance drops as expected.

Another thing worth reminding is that, this classifier is especially fast, much faster than neural networks. We personally nominate it as one of the best classifiers for this data set, though it is outside the scope of EE559.

## 4.2 SVM

The method of support vector machines is not suitable for a categorical data set. However, one-hot encoding saves all of this. it makes all of the new features easy to understand and tell apart. Other than that, it might not be so helpful.

## 4.3 Probabilistic Methods

### 4.3.1 k-NN

In the one-hot encoding part, k-NN works pretty well, which is out of our expectation. We assume there might be some overfitting, but sanity checks tell us that is not the case.

We perform sanity checks as below:

```
# Create a pipeline
pca_knn_pipeline = make_pipeline(pca, knn_man_new)
# Perform 4-fold cross-validation
cv_scores = cross_val_score(pca_knn_pipeline, X, y_train, cv=4)
mean_cv_score = np.mean(cv_scores)
std_cv_score = np.std(cv_scores)
```

The result is:

```
Mean CV score:  0.999274819874614
Std Deviation:  0.00016705877300795597
```

k-NN is not so capable of splitting categorical features either. For categorical labels, the distance cannot be expressed clearly. After all, the "distance" from label 6 to 1 should be the same as that from 6 to 7. The euclidean distance has more error than using Manhattan distance, and therefore has a worse performance.

Nevertheless, when the features are all replaced as numbers, the "distance" could have its realistic meaning in the feature space, and makes the model work well. We only choose three of the features to do feature engineering, according the abs value of their correlation between the class parameter (p or e). We could choose more of them to see if a higher accuracy could be reached if we have time.

Actually "cap-shape" should also be among these chosen features, but we find out it might not as a positive factor, but would instead decrease accuracy.

#### 4.3.2 Naive Bayes

Speaking of the correlations, the two tables are the reasons we want to try `categoricalNB()`, since all the correlations as to the class parameter are around 0.1 or -0.1. It is reasonable to assume their independence.

If this assumption does not hold true for the data set, the classifier's performance could be adversely affected. The result might disprove this assumption.

### 4.4 Neural Network

Neural Network is nearly the cure-all in problems of this size. As long as time permits, a decent result should always be available. In our project, however, we face the curse of dimensionality. After feature engineering that expands the dimension, the performance deteriorates. More of the data points in a higher-dimensional space get far away from each other, making the feature space more sparse, and more difficult to find a decision boundary.

We will keep the result here since 95% is also good enough, also as a notice of this curse.

### 4.5 Feature Reduction

The issue in this data set, in our opinion, is more than solely choosing key features. Feature reduction might not be a good idea. We tried existing methods to select features, and the result is as expected.

## 5 Libraries used and what you coded yourself

Here are the libraries used:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import *
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier

from scipy.stats import spearmanr

import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
```

Most of the classifiers have existing methods. We process features by ourselves, as is mentioned in Part 3.3.1. The accuracy, F1-score, and plot of confusion matrices all come from ourselves.

Special thanks to **Github Copilot**. This VS code extension really saves us a lot of time.

# 6 Contributions of each team member

**Yuan Chang**: Overall contribution 50%. Completed the one-hot encoding route and certain parts of label encoding part.

**Fangqi Yuan**: Overall contribution 50%. Completed the rest of the label encoding part and feature engineering.

The report is written together on `https://overleaf.com`.

# 7 Summary and conclusions

In this project, various preprocessing approaches have been employed to handle the mixed data set containing both categorical and numerical features. Following these preprocessing steps, classification is performed using 4 classifiers: k-nearest neighbors, decision trees, neural networks, and support vector machines.

The k-nearest neighbor classifier exhibits impressive performance in certain preprocessing scenarios. On the other hand, decision trees demonstrate their versatility by adapting well to different feature representations, making them a reliable choice in categorical sets.

Neural networks stand out as a robust and highly adaptive classifier, capable of handling the challenges posed by the curse of dimensionality, which may arise due to one-hot encoding or other high-dimensional feature representations.

In summary, this project explores multiple preprocessing approaches and classifiers, through which it is possible to identify the most effective strategy for accurately classifying instances in the mushroom data set, ultimately leading to a deeper understanding of the underlying patterns and relationships within the data.

# References


[1] *Mushroom Classification*, available at `https://www.kaggle.com/code/riddhiparakh/mushroom-classification`

[2] *Mushroom Classification using Neural Network*, available at `https://www.kaggle.com/code/wenzhengding/mushroom-classification-using-neural-network/notebook`