Fatal Fungi: Enchancing Mushrrom Safety through Backprogated Multi-Layer Perceptron Neural Network Identification

CSS 581 - Final Project ML Implmentation

Yasmine Subbagh

Importing the Data

The dataset used to train and test this neural netowrk is a mixture of two free open-source Kaggle datasets. This method was chosen in order to maximize the training dataset size to create a more accurate classifier.

In [1]:
```
import kagglehub
import numpy as np # linear algebra
np.random.seed(10)
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings

# Required magic to display matplotlib plots in notebooks
%matplotlib inline

from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classificatior
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packa
ges/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_instal
l.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
# download from kaggle
path1 = kagglehub.dataset_download("vishalpnaik/mushroom-classification-edib
```

```
path2 = kagglehub.dataset_download("ulrikthygepedersen/mushroom-attributes")

files1 = os.listdir(path1)
files2 = os.listdir(path2)

file1 = os.path.join(path1, "mushroom.csv")
file2 = os.path.join(path2, "mushroom.csv")

# Read the file into a list and clean up the byte-like characters
with open(file2, "r") as file:
    lines = file.readlines()

# Clean the lines
cleaned_lines = [line.replace("b'", "").replace("'", "").strip() for line in

# Create a DataFrame
header = cleaned_lines[0].split(",")
data = [row.split(",") for row in cleaned_lines[1:]]

# Load the datasets
df1 = pd.read_csv(file1)
df2 = pd.DataFrame(data, columns=header)

# Print the size of the datasets
print(f"Dataset 1 Size: {df1.shape[0]} rows, {df1.shape[1]} columns")
print(f"Dataset 2 Size: {df2.shape[0]} rows, {df2.shape[1]} columns")
```

```
Dataset 1 Size: 61069 rows, 21 columns
Dataset 2 Size: 8124 rows, 23 columns
```

Next, the data needs to be cleaned up and combined into one dataset.

In [3]:
```
# Display the DataFrame
print(df2.head())
```

```
  cap-shape cap-surface cap-color bruises%3F odor gill-attachment  \
0         x           s         n          t    p               f
1         x           s         y          t    a               f
2                     s         w          t    l               f
3         x           y         w          t    p               f
4         x           s         g          f    n               f

  gill-spacing gill-size gill-color stalk-shape stalk-root  \
0            c         n          k           e          e
1            c                    k           e          c
2            c                    n           e          c
3            c         n          n           e          e
4            w                    k           t          e

  stalk-surface-above-ring stalk-surface-below-ring stalk-color-above-ring
\
0                        s                        s                       w
1                        s                        s                       w
2                        s                        s                       w
3                        s                        s                       w
4                        s                        s                       w

  stalk-color-below-ring veil-type veil-color ring-number ring-type  \
0                      w         p          w           o         p
1                      w         p          w           o         p
2                      w         p          w           o         p
3                      w         p          w           o         p
4                      w         p          w           o         e

  spore-print-color population habitat class
0                 k          s       u     p
1                 n          n       g     e
2                 n          n       m     e
3                 k          s       u     p
4                 n          a       g     e
```

In [4]:
```
# Display the DataFrame
print(df1.head())
```

```
  class  cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed
\
0     p         15.26         x           g         o                    f
1     p         16.60         x           g         o                    f
2     p         14.07         x           g         o                    f
3     p         14.17         f           h         e                    f
4     p         14.64         x           h         o                    f

  gill-attachment gill-spacing gill-color  stem-height  stem-width stem-root
\
0               e          NaN          w        16.95       17.09         s
1               e          NaN          w        17.99       18.19         s
2               e          NaN          w        17.80       17.74         s
3               e          NaN          w        15.77       15.98         s
4               e          NaN          w        16.53       17.20         s

  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0            y          w         u          w        t         g
1            y          w         u          w        t         g
2            y          w         u          w        t         g
3            y          w         u          w        t         p
4            y          w         u          w        t         p

  spore-print-color habitat season
0               NaN       d      w
1               NaN       d      u
2               NaN       d      w
3               NaN       d      w
4               NaN       d      w
```

After exporloration of the dataset size and attributes, the executive decision to not to continue to use dataset 2 moving forward. This is because it is signifaclty smaller and the attributes do not align closely, lots of atributes would need to be cut to combine the dataset.

Next, we need to identify duplicate rows and delete them. As well as replace missing catergoicals data with a "unkown" value to avoid deleting too many data rows. But rows with missing numerical data can be deleted.

In [5]:
```
# drop duplicates
print("Number of duplicate rows:", df1.duplicated().sum())
df1 = df1.drop_duplicates()

# drop rows with missing numerical values
numerical_columns = df1.select_dtypes(include=['int64', 'float64']).columns
print(f"Missing values in numerical columns:\n{df1[numerical_columns].isnull
df1 = df1.dropna(subset=numerical_columns)
```

```
# replace missing categorical values with 'Unknown'
categorical_columns = df1.select_dtypes(include=['object']).columns
print(f"Missing values in categorical columns:\n{df1[categorical_columns].is
df1[categorical_columns] = df1[categorical_columns].fillna('Unknown')
```

```
Number of duplicate rows: 146
Missing values in numerical columns:
cap-diameter    0
stem-height     0
stem-width      0
dtype: int64
Missing values in categorical columns:
class                       0
cap-shape                   0
cap-surface             14120
cap-color                   0
does-bruise-or-bleed        0
gill-attachment          9855
gill-spacing            25062
gill-color                  0
stem-root               51536
stem-surface            38122
stem-color                  0
veil-type               57746
veil-color              53510
has-ring                    0
ring-type                2471
spore-print-color       54597
habitat                     0
season                      0
dtype: int64
```

We will not be dropping any columns as it is important for the model to be able to take into account as many attributes to be able to most accuretly classify the species, every detail is important and small nuacnces can indicate differenet species (which could change class type). Additionally, feature creating is not possible for this type of dataset.

Next, we will rename and reformat the columns and data to be hot coded or scaled to be able to be used in the perceptron layers with accuracy.

In [6]:
```
# Modify 'class' column to 'edible' with one-hot encoding (0 for 'p' and 1 f
df1['edible'] = df1['class'].map({'p': 0, 'e': 1})

# One-hot encode the 'does-bruise-or-bleed' column
df1['does_bruise_or_bleed'] = df1['does-bruise-or-bleed'].map({'t': 1, 'f':

# One-hot encode the 'has-ring' column
df1['has-ring'] = df1['has-ring'].map({'t': 1, 'f': 0})
```

```
# scale the cap diamter, stem height, and stem width to be within 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
df1['cap-diameter'] = scaler.fit_transform(df1[['cap-diameter']])
df1['stem-height'] = scaler.fit_transform(df1[['stem-height']])
df1['stem-width'] = scaler.fit_transform(df1[['stem-width']])

# Use pd.get_dummies() to one-hot encode the 'gill-attachment' column
df_encoded_gillAttachment = pd.get_dummies(df1['gill-attachment'], prefix='g
df_encoded_gillAttachment = df_encoded_gillAttachment * 1
df1 = pd.concat([df1, df_encoded_gillAttachment], axis=1)

# Use pd.get_dummies() to one-hot encode the 'cap-shape' column
df_encoded_cap_shape = pd.get_dummies(df1['cap-shape'], prefix='cap_shape',
df1 = pd.concat([df1, df_encoded_cap_shape], axis=1)

# Use pd.get_dummies() to one-hot encode the 'cap-surface' column
df_encoded_cap_surface = pd.get_dummies(df1['cap-surface'], prefix='cap_surf
df1 = pd.concat([df1, df_encoded_cap_surface], axis=1)

# Use pd.get_dummies() to one-hot encode the 'cap-color' column
df_encoded_cap_color = pd.get_dummies(df1['cap-color'], prefix='cap_color',
df1 = pd.concat([df1, df_encoded_cap_color], axis=1)

# Use pd.get_dummies() to one-hot encode the 'gill-spacing' column
df_encoded_gill_spacing = pd.get_dummies(df1['gill-spacing'], prefix='gill_s
df1 = pd.concat([df1, df_encoded_gill_spacing], axis=1)

# Use pd.get_dummies() to one-hot encode the 'gill-color' column
df_encoded_gill_color = pd.get_dummies(df1['gill-color'], prefix='gill_color
df1 = pd.concat([df1, df_encoded_gill_color], axis=1)

# Use pd.get_dummies() to one-hot encode the 'stem-root' column
df_encoded_stem_root = pd.get_dummies(df1['stem-root'], prefix='stem_root',
df1 = pd.concat([df1, df_encoded_stem_root], axis=1)

# Use pd.get_dummies() to one-hot encode the 'stem-surface' column
df_encoded_stem_surface = pd.get_dummies(df1['stem-surface'], prefix='stem_s
df1 = pd.concat([df1, df_encoded_stem_surface], axis=1)

# Use pd.get_dummies() to one-hot encode the 'stem-color' column
df_encoded_stem_color = pd.get_dummies(df1['stem-color'], prefix='stem_color
df1 = pd.concat([df1, df_encoded_stem_color], axis=1)

# Use pd.get_dummies() to one-hot encode the 'veil-color' column
df_encoded_veil_color = pd.get_dummies(df1['veil-color'], prefix='veil_color
df1 = pd.concat([df1, df_encoded_veil_color], axis=1)

# Use pd.get_dummies() to one-hot encode the 'ring-type' column
df_encoded_ring_type = pd.get_dummies(df1['ring-type'], prefix='ring-type',
df1 = pd.concat([df1, df_encoded_ring_type], axis=1)
```

```
# Use pd.get_dummies() to one-hot encode the 'spore-print-color' column
df_encoded_spore_print_color = pd.get_dummies(df1['spore-print-color'], pref
df1 = pd.concat([df1, df_encoded_spore_print_color], axis=1)

# Use pd.get_dummies() to one-hot encode the 'season' column
df_encoded_season = pd.get_dummies(df1['season'], prefix='season', drop_firs
df1 = pd.concat([df1, df_encoded_season], axis=1)

# Use pd.get_dummies() to one-hot encode the 'habitat' column
df_encoded_habitat = pd.get_dummies(df1['habitat'], prefix='habitat', drop_f
df1 = pd.concat([df1, df_encoded_habitat], axis=1)

#only one value for veil-type, drop column
df1 = df1.drop(columns=['veil-type'])

# Drop the old columns
df1 = df1.drop(columns=['habitat', 'season', 'spore-print-color', 'spore_pri
print(df1.head())
```

```
   cap-diameter  stem-height  stem-width  has-ring  edible  \
0      0.240155     0.499705    0.164469         1       0
1      0.261782     0.530366    0.175055         1       0
2      0.220949     0.524764    0.170725         1       0
3      0.222563     0.464917    0.153787         1       0
4      0.230148     0.487323    0.165528         1       0

   does_bruise_or_bleed  gill_attachment_a  gill_attachment_d  \
0                     0                  0                  0
1                     0                  0                  0
2                     0                  0                  0
3                     0                  0                  0
4                     0                  0                  0

   gill_attachment_e  gill_attachment_f  gill_attachment_p  gill_attachment_
s  \
0                  1                  0                  0
0
1                  1                  0                  0
0
2                  1                  0                  0
0
3                  1                  0                  0
0
4                  1                  0                  0
0

   gill_attachment_x  cap_shape_b  cap_shape_c  cap_shape_f  cap_shape_o  \
0                  0            0            0            0            0
1                  0            0            0            0            0
```

```
2                  0            0            0            0            0
3                  0            0            0            1            0
4                  0            0            0            0            0

   cap_shape_p  cap_shape_s  cap_shape_x  cap_surface_d  cap_surface_e  \
0            0            0            1              0              0
1            0            0            1              0              0
2            0            0            1              0              0
3            0            0            0              0              0
4            0            0            1              0              0

   cap_surface_g  cap_surface_h  cap_surface_i  cap_surface_k  cap_surface_l
\
0              1              0              0              0              0
1              1              0              0              0              0
2              1              0              0              0              0
3              0              1              0              0              0
4              0              1              0              0              0

   cap_surface_s  cap_surface_t  cap_surface_w  cap_surface_y  cap_color_b
\
0              0              0              0              0            0
1              0              0              0              0            0
2              0              0              0              0            0
3              0              0              0              0            0
4              0              0              0              0            0

   cap_color_e  cap_color_g  cap_color_k  cap_color_l  cap_color_n  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            1            0            0            0            0
4            0            0            0            0            0

   cap_color_o  cap_color_p  cap_color_r  cap_color_u  cap_color_w  \
0            1            0            0            0            0
1            1            0            0            0            0
2            1            0            0            0            0
3            0            0            0            0            0
4            1            0            0            0            0

   cap_color_y  gill_spacing_c  gill_spacing_d  gill_spacing_f  gill_color_b
\
0            0               0               0               0            0
1            0               0               0               0            0
2            0               0               0               0            0
3            0               0               0               0            0
4            0               0               0               0            0

   gill_color_e  gill_color_f  gill_color_g  gill_color_k  gill_color_n  \
```

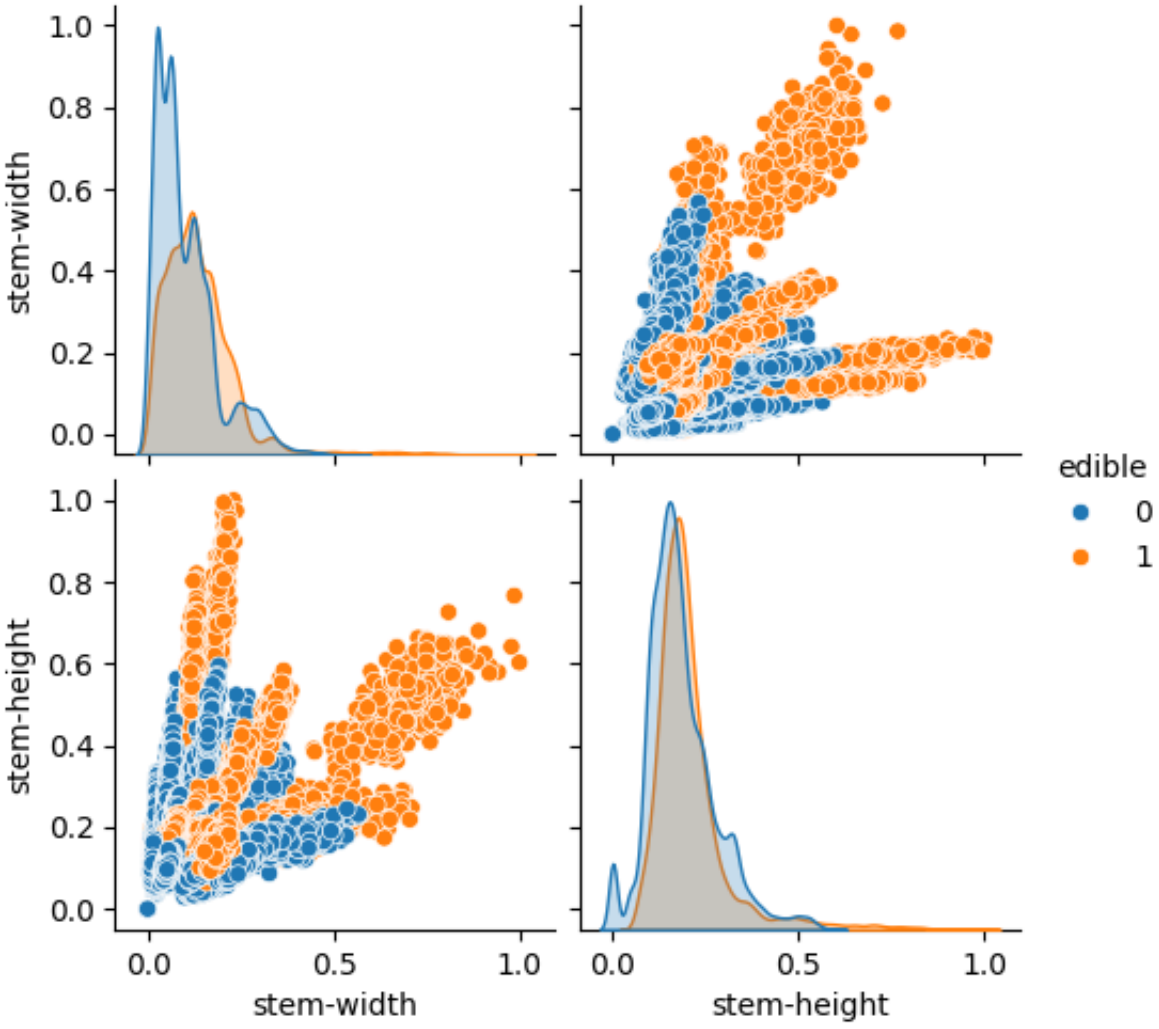
```
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             0             0             0
3             0             0             0             0             0
4             0             0             0             0             0

   gill_color_o  gill_color_p  gill_color_r  gill_color_u  gill_color_w  \
0             0             0             0             0             1
1             0             0             0             0             1
2             0             0             0             0             1
3             0             0             0             0             1
4             0             0             0             0             1

   gill_color_y  stem_root_b  stem_root_c  stem_root_f  stem_root_r  \
0             0            0            0            0            0
1             0            0            0            0            0
2             0            0            0            0            0
3             0            0            0            0            0
4             0            0            0            0            0

   stem_root_s  stem_surface_f  stem_surface_g  stem_surface_h  \
0            1               0               0               0
1            1               0               0               0
2            1               0               0               0
3            1               0               0               0
4            1               0               0               0

   stem_surface_i  stem_surface_k  stem_surface_s  stem_surface_t  \
0               0               0               0               0
1               0               0               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

   stem_surface_y  stem_color_b  stem_color_e  stem_color_f  stem_color_g  \
0               1             0             0             0             0
1               1             0             0             0             0
2               1             0             0             0             0
3               1             0             0             0             0
4               1             0             0             0             0

   stem_color_k  stem_color_l  stem_color_n  stem_color_o  stem_color_p  \
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             0             0             0
3             0             0             0             0             0
4             0             0             0             0             0

   stem_color_r  stem_color_u  stem_color_w  stem_color_y  veil_color_e  \
0             0             0             1             0             0
```

```
1             0             0             1             0             0
2             0             0             1             0             0
3             0             0             1             0             0
4             0             0             1             0             0

   veil_color_k  veil_color_n  veil_color_u  veil_color_w  veil_color_y  \
0             0             0             0             1             0
1             0             0             0             1             0
2             0             0             0             1             0
3             0             0             0             1             0
4             0             0             0             1             0

   ring-type_e  ring-type_f  ring-type_g  ring-type_l  ring-type_m  \
0            0            0            1            0            0
1            0            0            1            0            0
2            0            0            1            0            0
3            0            0            0            0            0
4            0            0            0            0            0

   ring-type_p  ring-type_r  ring-type_z  spore_print_color_g  \
0            0            0            0                    0
1            0            0            0                    0
2            0            0            0                    0
3            1            0            0                    0
4            1            0            0                    0

   spore_print_color_k  spore_print_color_n  spore_print_color_p  \
0                    0                    0                    0
1                    0                    0                    0
2                    0                    0                    0
3                    0                    0                    0
4                    0                    0                    0

   spore_print_color_r  spore_print_color_u  spore_print_color_w  season_a
\
0                    0                    0                    0         0
1                    0                    0                    0         0
2                    0                    0                    0         0
3                    0                    0                    0         0
4                    0                    0                    0         0

   season_s  season_u  season_w  habitat_d  habitat_g  habitat_h  habitat_l
\
0         0         0         1          1          0          0          0
1         0         1         0          1          0          0          0
2         0         0         1          1          0          0          0
3         0         0         1          1          0          0          0
4         0         0         1          1          0          0          0

   habitat_m  habitat_p  habitat_u  habitat_w
```

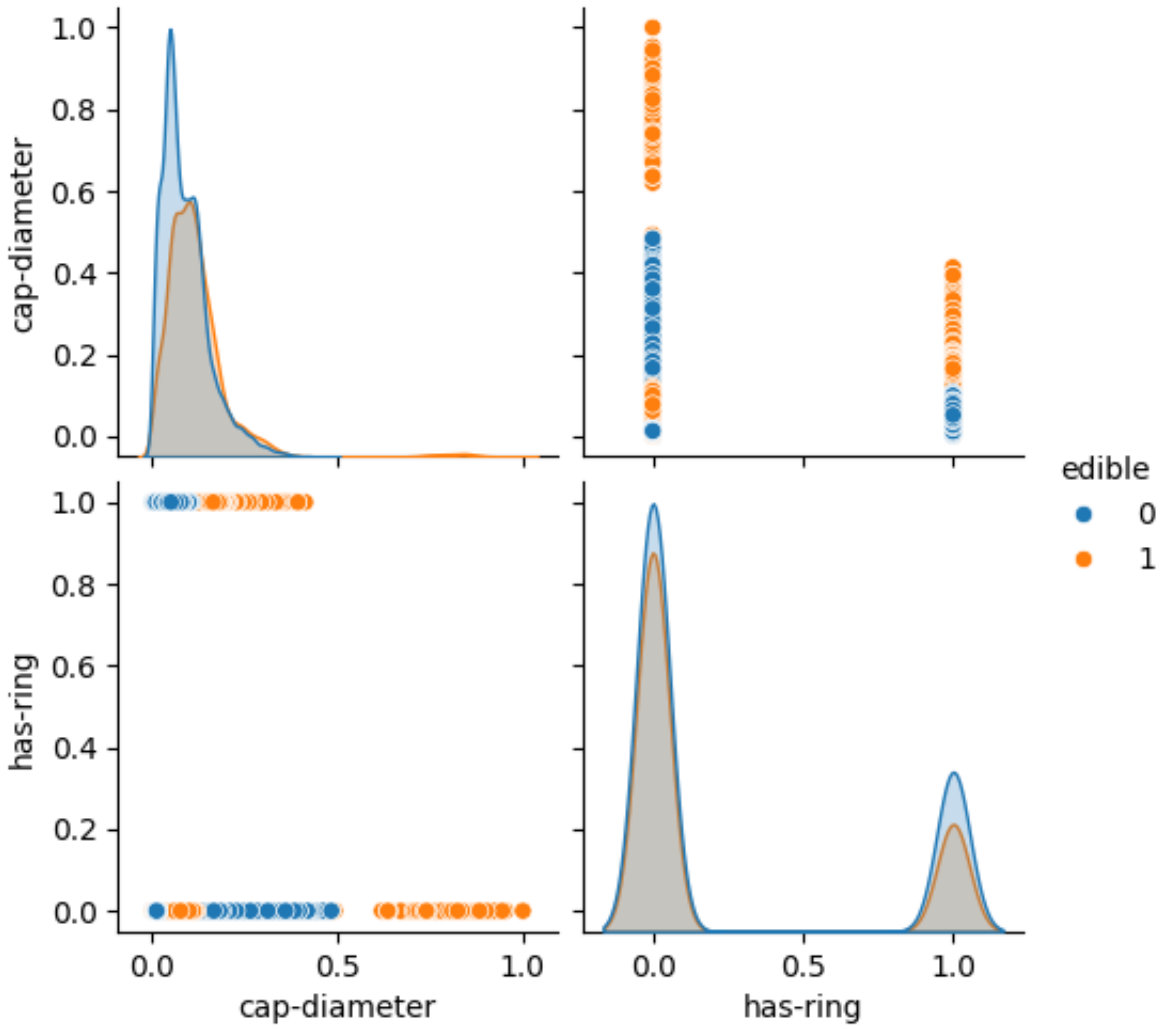
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Now that we have finished up with the data pre-processing, we can do some quick data exploration.

In [7]:
```
# Pairplot of a subset of features (you can select a few important ones to m
subset = df1[['stem-width', 'stem-height', 'edible']]
sns.pairplot(subset, hue='edible')  # Color by 'edible' class
plt.show()
```



In [8]:
```
# Pairplot of a subset of features (you can select a few important ones to m
subset = df1[['cap-diameter', 'has-ring', 'edible']]
sns.pairplot(subset, hue='edible')  # Color by 'edible' class
plt.show()
```

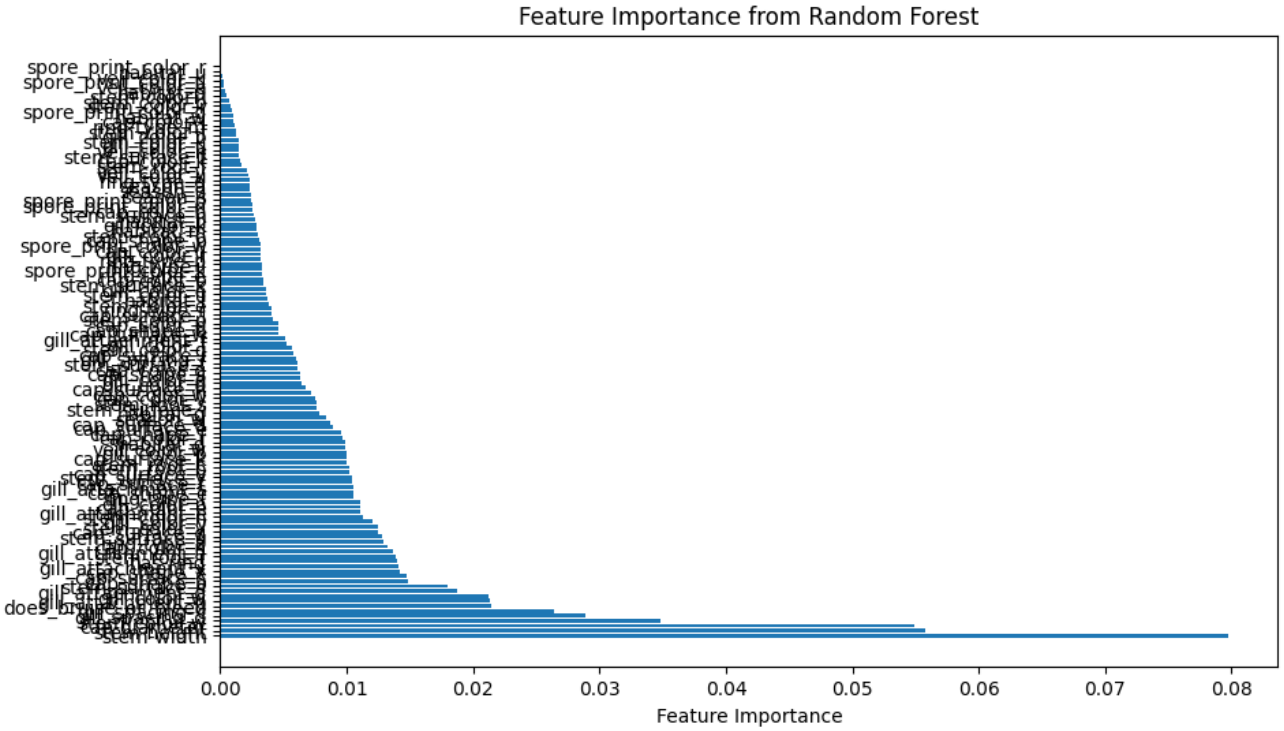


As we can see from the pairplots above, there seams to be a strong grouping features to the edible (or not) classification. This indidcates that are model should be able to classify mushrooms given the data

In [9]:
```
# Train a RandomForest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(df1.drop(columns=['edible']), df1['edible'])

# Get feature importances
importances = model.feature_importances_

feature_importance_df = pd.DataFrame({
    'Feature': df1.drop(columns=['edible']).columns,
    'Importance': importances
})

feature_importance_df = feature_importance_df.sort_values(by='Importance', a

# List the top 5 most important features
```

```
top_5_features = feature_importance_df.head(5)
print(top_5_features)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance
plt.xlabel('Feature Importance')
plt.title('Feature Importance from Random Forest')
plt.show()
```

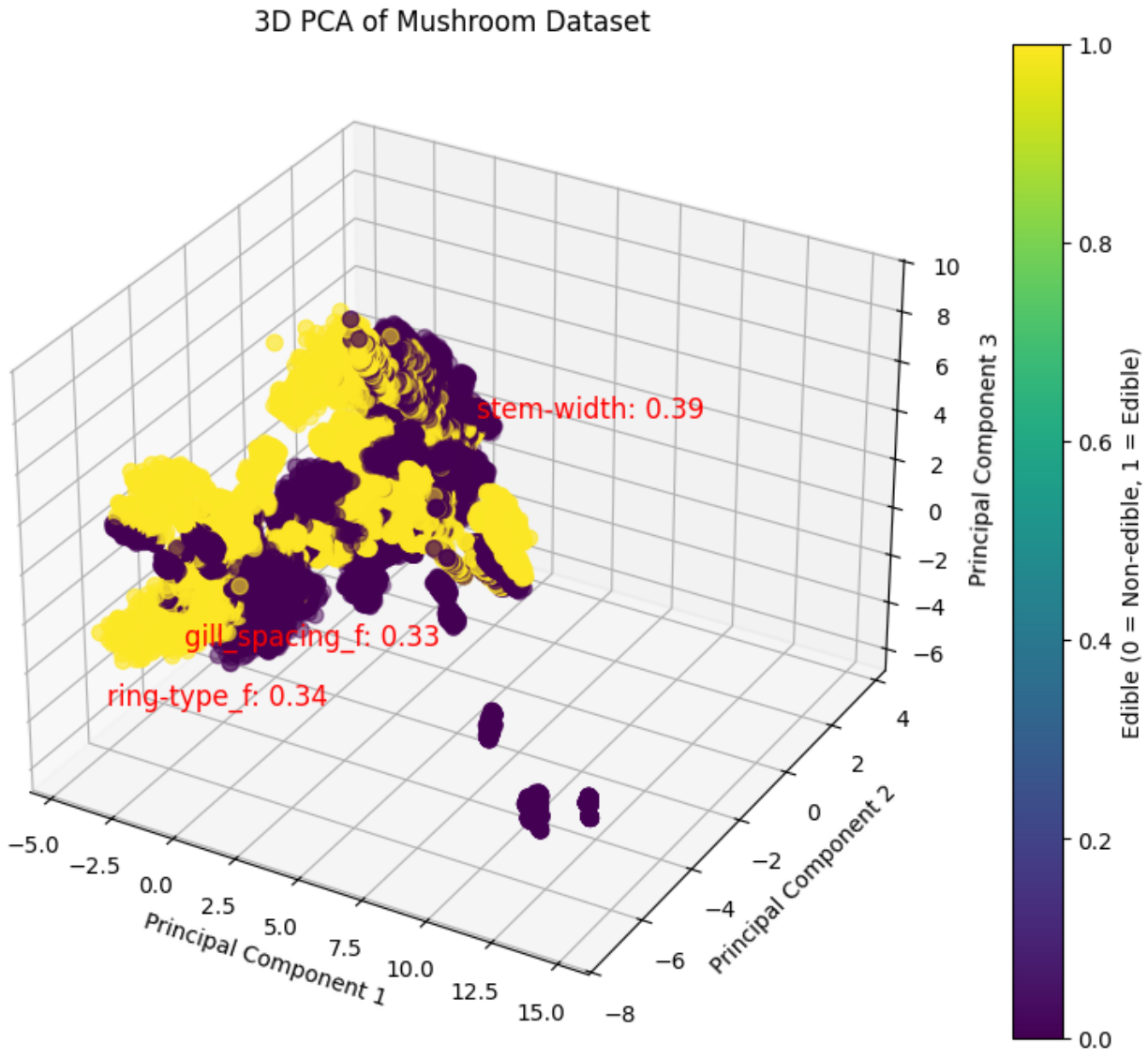
```
          Feature  Importance
2       stem-width    0.079685
1      stem-height    0.055777
0     cap-diameter    0.054871
81    stem_color_w    0.034807
43  gill_spacing_d    0.028843
```



As we can see from the feature importance graph above, the stem-width has strong feature importance compared to the rest of the features. However, since the data is hot coded, the other features could not be truly represnted.

In [10]:
```
# Separate features and target
X = df1.drop(columns=['edible'])
y = df1['edible']

# Perform PCA and reduce to 3 components
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=3)
```

```
X_pca = pca.fit_transform(X_scaled)

pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3'])
pca_df['edible'] = y
components_df = pd.DataFrame(pca.components_, columns=X.columns, index=['PC1

# Plot the 3D PCA plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('3D PCA of Mushroom Dataset')

# Color points based on the 'edible' target (0 for non-edible, 1 for edible)
scatter = ax.scatter(pca_df['PC1'], pca_df['PC2'], pca_df['PC3'], c=pca_df['

fig.colorbar(scatter, label='Edible (0 = Non-edible, 1 = Edible)')
for i in range(3):  # Loop through the 3 components
    ax.text(
        pca_df.iloc[0, i], pca_df.iloc[1, i], pca_df.iloc[2, i],
        f'{components_df.iloc[i].idxmax()}: {components_df.iloc[i].max():.2f
        color='red', fontsize=12
    )

plt.show()
```



The 3D PCA shown above shows some grouping thats are clear in their inedibility due to stem_width, we saw this earlier in the feature importance, so this is supporting that the model will be able to classify with accuracy. While there are still groupings that can lead to easy classification by the 3 components, they are not as clear and seperated.

Backpropogated Perceptron Neural Netowrk

For this model, we will be using a 3 layer backpropogated percpetron neural nework. The activation function will be sigmoid to as ReLu could drop smaller features from contributing to the classification.

In [11]:
```
# seperate features and target
X = df1.drop('edible', axis=1)
Y = df1['edible']
# spit data into training and testing sets 80:20
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rar

#reshape to vertical vector
Y_train = Y_train.values.reshape(-1, 1)
Y_test = Y_test.values.reshape(-1, 1)

# print splits
print("Training feature set shape:", X_train.shape)
print("Testing feature set shape:", X_test.shape)
print("Training labels shape:", Y_train.shape)
print("Testing labels shape:", Y_test.shape)
```

```
Training feature set shape: (48738, 116)
Testing feature set shape: (12185, 116)
Training labels shape: (48738, 1)
Testing labels shape: (12185, 1)
```

With the data now split, we need to build and train the model.

In [18]:
```
# Arrays to store the loss values
train_losses = []
test_losses = []

# layer perceptron nueron counts
input_neurons = X.shape[1]
hidden_neurons_1 = 21  # first hidden layer
hidden_neurons_2 = 7   # second hidden layer
output_neurons = 1

# initialize random weights and biases
np.random.seed(42)
W1 = np.random.randn(input_neurons, hidden_neurons_1) * np.sqrt(2 / input_ne
b1 = np.zeros((1, hidden_neurons_1))
W2 = np.random.randn(hidden_neurons_1, hidden_neurons_2) * np.sqrt(2 / hidde
b2 = np.zeros((1, hidden_neurons_2))
W3 = np.random.randn(hidden_neurons_2, output_neurons) * np.sqrt(2 / hidden_
b3 = np.zeros((1, output_neurons))

# activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Hyperparameters
learning_rate = 0.05
epochs = 30000

# Training loop
```

```
for epoch in range(epochs):
    # Forward pass
    Z1 = np.dot(X_train, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)
    Z3 = np.dot(A2, W3) + b3
    A3 = sigmoid(Z3)

    # Compute cross-entropy loss
    epsilon = 1e-10  # To avoid log(0)
    loss = -np.mean(Y_train * np.log(A3 + epsilon) + (1 - Y_train) * np.log(
    train_losses.append(loss)

    # Backward pass
    dZ3 = A3 - Y_train
    dW3 = np.dot(A2.T, dZ3) / X_train.shape[0]
    db3 = np.sum(dZ3, axis=0, keepdims=True) / X_train.shape[0]

    dA2 = np.dot(dZ3, W3.T)
    dZ2 = dA2 * sigmoid_derivative(A2)
    dW2 = np.dot(A1.T, dZ2) / X_train.shape[0]
    db2 = np.sum(dZ2, axis=0, keepdims=True) / X_train.shape[0]

    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * sigmoid_derivative(A1)
    dW1 = np.dot(X_train.T, dZ1) / X_train.shape[0]
    db1 = np.sum(dZ1, axis=0, keepdims=True) / X_train.shape[0]

    # Update weights and biases
    W3 -= learning_rate * dW3
    b3 -= learning_rate * db3
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1

    # Forward pass (testing data)
    Z1_test = np.dot(X_test, W1) + b1
    A1_test = sigmoid(Z1_test)
    Z2_test = np.dot(A1_test, W2) + b2
    A2_test = sigmoid(Z2_test)
    Z3_test = np.dot(A2_test, W3) + b3
    A3_test = sigmoid(Z3_test)

    # Compute testing loss (cross-entropy)
    test_loss = -np.mean(Y_test * np.log(A3_test) + (1 - Y_test) * np.log(1
    test_losses.append(test_loss)

    # print testing loss
```

```
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

```
Epoch 0, Loss: 0.7026
Epoch 1000, Loss: 0.6829
Epoch 2000, Loss: 0.6649
Epoch 3000, Loss: 0.6043
Epoch 4000, Loss: 0.5227
Epoch 5000, Loss: 0.4597
Epoch 6000, Loss: 0.4170
Epoch 7000, Loss: 0.3822
Epoch 8000, Loss: 0.3495
Epoch 9000, Loss: 0.3172
Epoch 10000, Loss: 0.2845
Epoch 11000, Loss: 0.2526
Epoch 12000, Loss: 0.2226
Epoch 13000, Loss: 0.1938
Epoch 14000, Loss: 0.1655
Epoch 15000, Loss: 0.1384
Epoch 16000, Loss: 0.1139
Epoch 17000, Loss: 0.0929
Epoch 18000, Loss: 0.0757
Epoch 19000, Loss: 0.0621
Epoch 20000, Loss: 0.0513
Epoch 21000, Loss: 0.0429
Epoch 22000, Loss: 0.0363
Epoch 23000, Loss: 0.0311
Epoch 24000, Loss: 0.0270
Epoch 25000, Loss: 0.0236
Epoch 26000, Loss: 0.0208
Epoch 27000, Loss: 0.0186
Epoch 28000, Loss: 0.0167
Epoch 29000, Loss: 0.0151
```

Next we will test the accuracy of the model using the previsouly seperated testing data.

*Note: While the hyperparamets were modified in order to try and achieve a better performing model, using LOTS of epochs proved to train the model best and denote the best performing model as possible.

In [22]:
```
# Classify accuracy using test data
Z1_test = np.dot(X_test, W1) + b1
A1_test = sigmoid(Z1_test)
Z2_test = np.dot(A1_test, W2) + b2
A2_test = sigmoid(Z2_test)
Z3_test = np.dot(A2_test, W3) + b3
A3_test = sigmoid(Z3_test)

# Convert predictions to binary
```

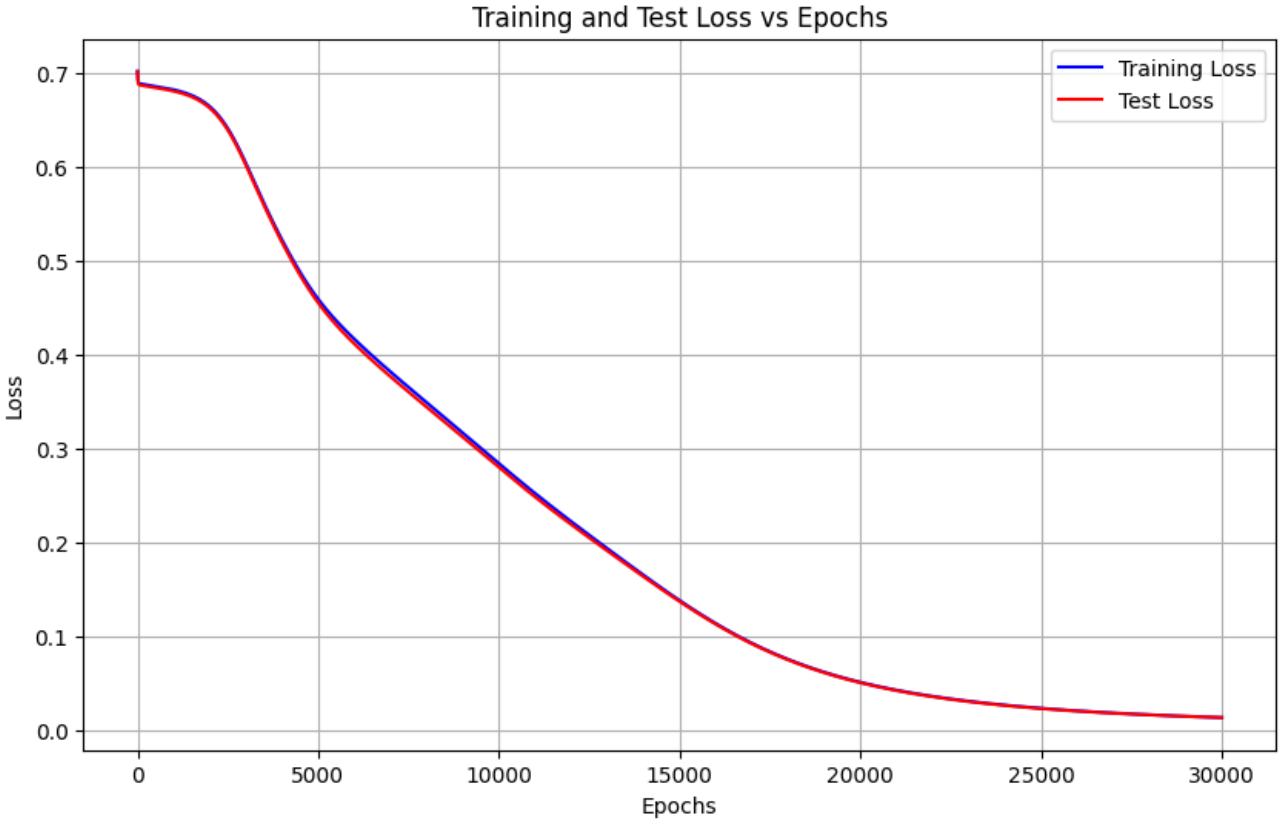
```
Y_pred = (A3_test > 0.5).astype(int)

# Print accuracy and classification report
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(Y_test, Y_pred))
```

```
Test Accuracy: 0.9990

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6820
           1       1.00      1.00      1.00      5365

    accuracy                           1.00     12185
   macro avg       1.00      1.00      1.00     12185
weighted avg       1.00      1.00      1.00     12185
```

From the accuracy and classification report, we can see that the model is performing very well as classifiying if a fungi is edible or posinous.

The precision for both the outcomes are 100%, meaning the model is able to always correctly identify if a mushroom is edible or poisnous based on its physical characteristics.

In [23]:
```
# Plotting the losses over epochs to check for overfitting
plt.figure(figsize=(10, 6))
plt.plot(range(epochs), train_losses, label='Training Loss', color='blue')
plt.plot(range(epochs), test_losses, label='Test Loss', color='red')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Test Loss vs Epochs')
plt.legend()
plt.grid(True)
plt.show()
```



As we can see from the graph above, the training loss and test data loss consistnely gets smaller. If at some the training loss were to continue to reduce bu the test loss were to begin increasing that would denote overfitting. While I was worried that the number of epochs that were used to get such a high classification score would lead to over fitting, it appears to not have and is just a well trained model.

The test below is again to help us determine if the model is overfitting by checking if the test data loss is ever more than the training loss. Since the test loss is never more than the training loss, it show us that there is no overfitting and the model is able to correctly classify new data.

In [24]:
```
# Check if the test loss is ever greater than the training loss
epochs_with_higher_test_loss = [epoch for epoch, (train_loss, test_loss) in

if epochs_with_higher_test_loss:
    print(f"The test loss exceeds the training loss at the following epochs:
else:
    print("The test loss is never greater than the training loss.")
```

```
The test loss is never greater than the training loss.
```