CS183FZ Critical Skills Project

Predication of Poisonous Mushroom

LINZE CHEN
24126390
QIYE CHEN
24127311
TIANXIA GAO
23125241
YIXUAN HUANG
24125822

Project – **2025**

Degree Major - **Computer Science and Software Engineering**

Maynooth International Engineering College
福州大学梅努斯国际工程学院

BSc in Computer Science and Software Engineering

Supervisor: **Dr.Emad Iranmanesh**

Declaration

We hereby certify that this material, which we now submit for assessment on the program of study as part of **(Maynooth international engineering college, Fuzhou University)** qualification, is *entirely* our own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of our work.

We hereby acknowledge and accept that this thesis may be distributed to future first year students, as an example of the standard expected first year projects.

Signed: Date: 2025/6/7

Team Roles:

LINZE CHEN 24126390: Team leader and write the reflective report

QIYE CHEN 24127311: Deputy team leader, finish all the models, frontend and backend of our project

TIANXIA GAO 233125241: Write the thesis and help team leader do the report

YIXUAN HUANG 24125822: Time reminder

Catalog

Declaration 2

Abstract 4

Introduction 4

Topic: 4

Problem: 4

Approach: 4

Metrics: 5

Project Achievements: 6

Background 6

Focus on the topic: 6

Relating to the Technical: 6

Problem 7

Difficult feature positioning: 7

Data preprocessing: 7

The risk of overfitting: 7

Solution&Evaluation: 8

Data Preprocessing: 8

Feature Selection: 8

Model Training: 8

Detailed Model Comparison: 8

Front-End Progress(code in appendix 3): 9

Back-End progress(code in appendix 1): 10

Model Optimization(decline the overfitting): 10

Evaluation： 10

Conclusions 10

Future Work: 10

References 11

Appendices 11

Abstract

This study aims to establish a reliable model for predicting the toxicity of mushrooms based on their physical characteristics. Accurate classification of mushrooms as either toxic or edible is of great significance for public safety and ecological research. Using Python and machine learning algorithms, we processed and analyzed the dataset provided by our supervisor, achieving a prediction accuracy of over 98% in multiple test models. Our approach includes data preprocessing, feature selection, model training and evaluation. We also conducted correlation analysis and established a linear regression model, while using a random forest model to improve the accuracy of the data.

Initially, a decision tree model was applied to predict mushroom toxicity. However, decision trees are prone to overfitting, especially when handling complex datasets. To overcome this limitation, we transitioned to random forests, an ensemble method that combines multiple decision trees. This shift significantly improved prediction accuracy, achieving over 98% accuracy in multiple test scenarios.

Keywords—Analyze the dataset, Machine learning, Model training and evaluation.

Introduction

Topic:

The identification of poisonous mushrooms is a critical task with significant implications for both ecological studies and public safety. Accurate prediction models can help in the rapid classification of mushroom species, preventing potential poisoning incidents and aiding in ecological research. Mushrooms are a diverse group of fungi, with some species being edible while others are highly toxic. The ability to distinguish between these species is essential for foragers and researchers. Traditional methods of identification often rely on expert knowledge and field guides, which can be time-consuming and prone to human error. With the advent of machine learning and data science, there is an opportunity to develop more efficient and accurate methods for predicting mushroom toxicity.

Problem:

The primary challenge in predicting poisonous mushrooms stems from the high diversity and variability across mushroom species, which makes it difficult to establish a universal prediction model. The dataset provided includes numerous features, such as cap shape, gill color, and spore print color, which can influence toxicity. Understanding the relationships between these features and toxicity is crucial for developing an effective prediction model. Misidentification of poisonous mushrooms can lead to severe poisoning incidents, while incorrect classification of edible mushrooms as poisonous can result in unnecessary waste. Therefore, developing a reliable and efficient prediction model is of great importance.

Approach:

We approached this problem using a combination of data preprocessing, feature selection, model training, and evaluation. We approached this problem using a combination of data preprocessing, feature selection, model training, and evaluation. We utilized Python and several machine learning algorithms, including logistic regression, decision trees, and random forests, to process the dataset and develop a reliable prediction model. Among these models, random forests were selected for their ability to reduce overfitting and improve accuracy by combining multiple decision trees. Our evaluation included metrics such as accuracy, precision, recall, and F1-score to ensure the model's reliability and effectiveness. The ability to distinguish between edible and poisonous mushrooms has long been a challenge due to the vast diversity of mushroom species and the subtle differences in their physical characteristics. Traditional methods of identification often rely on expert knowledge and field guides, which can be time-consuming and prone to human error. With the advent of machine learning and data science, there is an opportunity to develop more efficient and accurate methods for predicting mushroom toxicity. This project explores the application of these advanced techniques to address this important problem.

Metrics:

Here are the metrics we use and the results they got:

Accuracy: The proportion of correctly classified instances. Result: over 98%. This metric was selected to measure the overall performance of our model, as it provides a clear indication of how well the model classifies both toxic and non-toxic mushrooms. (code in appendix2-1)

Decision tree: A tree-shaped model that divides data layer by layer based on features, used for classification or regression. Result: depth 15(code in appendix2-3)

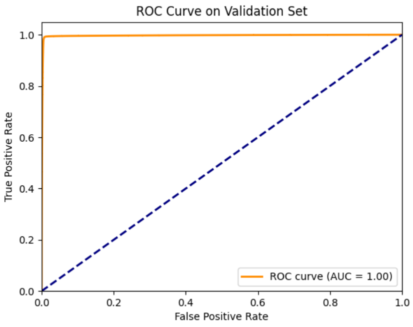
Random forest: Composed of multiple decision trees it is a model constructed by randomly sampling and feature selection to build trees and then integrating the results. Result: best max depth 22(code in appendix2-4)

Precision: The proportion of true positives among all positive predictions. Result:99% (code in appendix2-7)

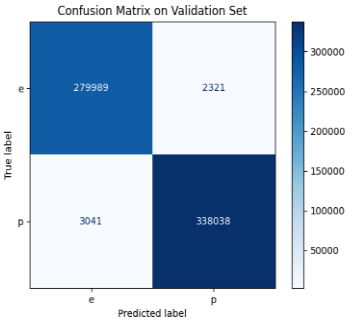
Recall: The proportion of true positives among all actual positives. Result:99.1%(code in appendix2-8)

F1-score: The harmonic mean of precision and recall. Result:99.1%(code in appendix2-2)

ROC curve: A graphical representation method used to evaluate the performance of classification models. (code in appendix2-9)



Confusion Matrix: A tabular tool used to evaluate the performance of classification model. (code in appendix2-5)



Project Achievements:

1.Achieved over 98% accuracy in predicting mushroom toxicity.

2.Conducted a comprehensive correlation analysis to understand feature relationships.

3.Established a linear regression model to quantify these relationships.

4.Implemented and evaluated multiple machine learning models, including logistic regression, decision trees, and random forests.

5.Provided a detailed analysis of the dataset and the factors influencing mushroom toxicity.

6.Develop a simple frontend interface to test and demonstrate the prediction functionality through the API

Background

Focus on the topic:

Mushrooms are a diverse group of fungi, with some species being edible while others are highly toxic. The ability to distinguish between these species is essential for foragers and this section, our team mainly explored the methods for identifying poisonous mushrooms, which involve judging based on certain characteristics of the mushrooms, such as their smell, the shape of the cap, and the colors at various locations[1]. Previous studies have utilized various machine learning techniques to classify mushrooms based on their physical and chemical properties[2]. This provided our team with the direction and ideas for project development. These studies have shown promising results, but there is still room for improvement . The study of mushroom toxicity has a long history, with early efforts focusing on the classification of species based on morphological characteristics. However, these methods often proved to be insufficient due to the high variability within and between species. In recent years, the availability of large datasets and advances in machine learning have opened up new possibilities for more accurate and reliable prediction models. Several studies have explored the use of decision trees, random forests, and other algorithms to classify mushrooms based on a variety of features, including cap shape, gill colour, and spore print colour[3]. There still are different ways to achieve this model, but we choose the machine learning lastly[4].

Relating to the Technical:

Techniques such as logistic regression, decision trees, and random forests have been widely used for their ability to handle large datasets and identify patterns that are not immediately apparent[5]. In this project, we leveraged these algorithms to develop a reliable prediction model for mushroom toxicity.

Among them, the decision tree model is an important model for our team to improve accuracy, and we have identified its advantages. The random forest algorithm has the following features: randomly and repeatedly draws the training decision tree dataset; and randomly selects the node splitting attributes in the decision tree growth strategy. These two excellent features of the algorithm make the prediction less prone to overfitting[6]. Decision trees offer a straightforward and interpretable approach to classification, while random forests enhance the robustness and accuracy by combining multiple decision trees. These algorithms were chosen for their balance of simplicity, interpretability, and performance, making them suitable for our task of predicting mushroom toxicity[6]. Our approach involved preprocessing the data, selecting relevant features, training the models, and evaluating their performance using standard metrics. Machine learning algorithms have proven to be effective in handling complex datasets and identifying patterns that may not be immediately apparent to human analysts. In this project, we utilized several well-established machine learning techniques to develop our prediction model. Logistic regression is a fundamental algorithm for binary classification tasks, providing a baseline for comparison with more complex models[7].

Problem

Difficult feature positioning:

The primary challenge in predicting poisonous mushrooms lies in the complexity and variability of mushroom species. The dataset provided for this project contains a wide range of features that describe the physical characteristics of mushrooms, including cap shape, cap surface, gill colour, gill spacing, spore print colour, and others. Each of these features can potentially influence the toxicity of a mushroom, making the task of prediction both challenging and interesting. Additionally, some features may have redundant information or may not contribute significantly to the prediction task. Therefore, feature selection is a crucial step in developing an effective model. Understanding the relationships between these features and toxicity is crucial for developing an effective prediction model. The dataset provided for this project contains a wide range of features that describe the physical characteristics of mushrooms. These features include cap shape, cap surface, gill color, gill spacing, spore print color, and many others. Each of these features can potentially influence the toxicity of a mushroom, making the task of prediction both challenging and interesting. It's hard for us to confirm which feature we really wants.

Data preprocessing:

To better understand the problem, we conducted an initial analysis of the dataset. The dataset contains a lot of instances; each is described by more than 15 features. These features include both categorical and numerical attributes, such as cap shape, cap surface, gill color, gill spacing, spore print color, and more. We performed exploratory data analysis (EDA) to gain insights into the dataset. This involved visualizing the distribution of features, identifying correlations between features, and understanding the relationship between features and the target variable. We used various visualization techniques, such as histograms, box plots, and heatmaps, to explore the data. For example, we found that certain gill colors, such as white and pink, were more frequently associated with edible mushrooms, while brown and black gills were more common in poisonous mushrooms. Similarly, cap shape and spore print color also showed distinct patterns that could be useful for classification.

The risk of overfitting:

Another challenge is the potential for overfitting, especially given the relatively small size of the dataset. Overfitting occurs when a model learns the training data too well, including its noise and outliers, resulting in poor generalization to new data. To mitigate this risk, we employed techniques such as cross-validation and regularization during model training.

Solution&Evaluation

Data Preprocessing:

We began by cleaning the data, handling missing values, and encoding categorical variables using one-hot encoding. This process ensured that the data was in a suitable format for machine learning algorithms. Data preprocessing is a crucial step in any machine learning project, as the quality of the data directly affects the performance of the model. The dataset we received contained a mix of numerical and categorical features, with some missing values. To handle these issues, we first performed a thorough exploratory data analysis to understand the structure and characteristics of the data. We then cleaned the data by removing any duplicate entries and handling missing values. For categorical variables, we used one-hot encoding to convert them into numerical format.This process involved creating binary columns for each category, ensuring that the data was properly formatted for further analysis.

Feature Selection:

We performed exploratory data analysis to identify the most relevant features for predicting mushroom toxicity. Using techniques such as correlation analysis, we selected features that had a significant impact on the target variable. This step helped in reducing the dimensionality of the dataset and improving model performance. Feature selection is an important step in building an effective machine learning model. With a large number of features, it was challenging to determine which ones are most relevant to the prediction task. To address this issue, we conducted an exploratory data analysis to understand the relationships between different features and the target variable. We used correlation analysis to identify features that had a strong relationship with mushroom toxicity. By selecting only the most relevant features, we were able to reduce the dimensionality of the dataset, which not only improved the performance of the model but also reduced the computational cost. This step was crucial in ensuring that our model was both efficient and accurate.

Model Training:

Initially, a decision tree model was implemented due to its simplicity and interpretability. However, the decision tree showed poor generalization and was highly prone to overfitting, especially with a larger and more complex dataset. To overcome this limitation, we transitioned to another machine learning models, specifically random forests. The random forest model improved performance by combining multiple decision trees, reducing overfitting and enhancing model accuracy. With random forests, we achieved over 98% accuracy and significantly better performance on unseen data. We implemented several machine learning models, including logistic regression, decision trees, and random forests, to predict mushroom toxicity. Each model was trained using the preprocessed dataset, and hyperparameters were tuned to optimize performance. Model training is a key part of the machine learning process, where the chosen algorithm learns from the data to make accurate predictions. In this project, we experimented with a variety of machine learning models to find the one that performed best for our task. We started with logistic regression, a simple yet powerful algorithm for binary classification tasks. We then moved on to more complex models, such as decision trees and random forests, which can capture more intricate patterns in the data. Each model was trained using the preprocessed dataset, and we carefully tuned the hyperparameters to optimize its performance. This involved experimenting with different settings and configurations to find the best combination that resulted in the highest accuracy and reliability.

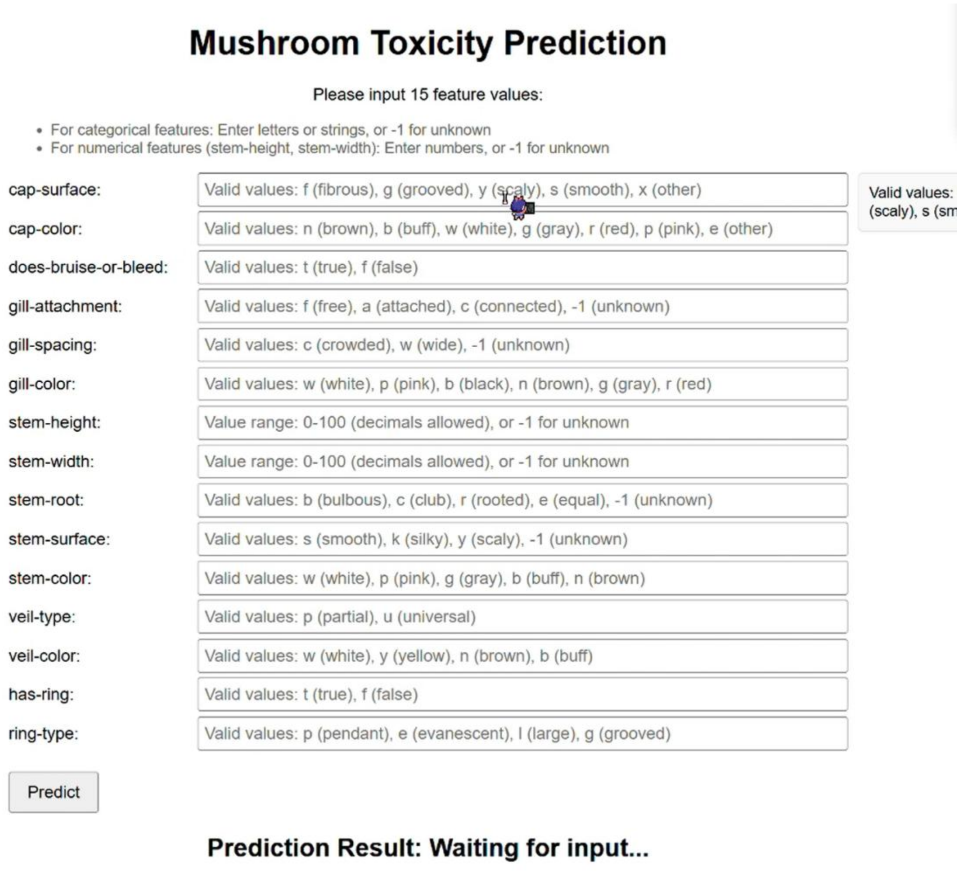
Detailed Model Comparison:

We compared the performance of several machine learning models, including logistic regression, decision trees, and random forests. Each model has its strengths and weaknesses, and the choice of model can significantly impact the prediction accuracy. Logistic regression is a simple and interpretable model that works well for linearly separable data. However, it may not capture complex patterns in the data. Decision trees are more flexible and can handle non-linear relationships, but they are prone to overfitting. Random forests, which are an ensemble of decision trees, offer a good balance between flexibility and robustness, making them suitable for our task. Our experiments showed that the random forest model achieved the highest accuracy, outperforming both logistic regression and decision trees. The comparison illustrates that random forests, as an ensemble method, significantly outperform individual decision trees in handling complex mushroom toxicity data, as evidenced by improved accuracy.

Front-End Progress(code in appendix 3):

The front-end interface in index.html has been completely redesigned to offer a better user experience. Input fields are now dynamically generated based on a list of 15 mushroom features. Each input field is accompanied by a tooltip that provides clear instructions on valid input values, making it easier for users to understand what is required. Input validation has been significantly improved. A validateInput function was added to check the validity of each input. For numerical features such as "stem height" and "stem width," it ensures the values are within the range of 0 to 100 or -1 for unknown values. For categorical features, it verifies that the input matches a predefined list of valid values. If an invalid input is detected, the system alerts the user with a detailed error message.

The form submission process has also been optimized. When the user submits the form, the system first validates all inputs. If all inputs are valid, it sends a POST request to the back-end API at http://localhost:8000/predict. During the prediction process, a "Predicting..." message is displayed to keep the user informed. Once the prediction result is received, it is shown on the page, indicating whether the mushroom is "Edible" or "Poisonous."



Mushroom Toxicity Prediction

Please input 15 feature values:

- For categorical features: Enter letters or strings, or -1 for unknown
- For numerical features (stem-height, stem-width): Enter numbers, or -1 for unknown

cap-surface:	Valid values: f (fibrous), g (grooved), y (scaly), s (smooth), x (other)
cap-color:	Valid values: n (brown), b (buff), w (white), g (gray), r (red), p (pink), e (other)
does-bruise-or-bleed:	Valid values: t (true), f (false)
gill-attachment:	Valid values: f (free), a (attached), c (connected), -1 (unknown)
gill-spacing:	Valid values: c (crowded), w (wide), -1 (unknown)
gill-color:	Valid values: w (white), p (pink), b (black), n (brown), g (gray), r (red)
stem-height:	Value range: 0-100 (decimals allowed), or -1 for unknown
stem-width:	Value range: 0-100 (decimals allowed), or -1 for unknown
stem-root:	Valid values: b (bulbous), c (club), r (rooted), e (equal), -1 (unknown)
stem-surface:	Valid values: s (smooth), k (silky), y (scaly), -1 (unknown)
stem-color:	Valid values: w (white), p (pink), g (gray), b (buff), n (brown)
veil-type:	Valid values: p (partial), u (universal)
veil-color:	Valid values: w (white), y (yellow), n (brown), b (buff)
has-ring:	Valid values: t (true), f (false)
ring-type:	Valid values: p (pendant), e (evanescent), l (large), g (grooved)

Valid values: (scaly), s (sm

Prediction Result: Waiting for input...

Back-End progress(code in appendix 1):

The back-end API, implemented using the FastAPI framework, was developed from scratch to handle the data sent from the front-end and perform the toxicity prediction using the pre-trained model. A Mushroom Features data model was defined using Pydantic to validate the input data, ensuring that all required features are present and that their values are in the correct format.

Model Optimization(decline the overfitting):

To further improve the performance of our models, we employed techniques such as hyperparameter tuning and cross-validation. Hyperparameter tuning involves adjusting the parameters of the model to find the optimal combination that results in the best performance. We used grid search and random search techniques to explore different hyperparameter settings and select the best ones. Cross-validation is a technique used to assess the generalization performance of a model by splitting the data into multiple folds and training the model on different subsets of the data. This helps in reducing overfitting and ensures that the model performs well on unseen data. Our experiments showed that using cross-validation and hyperparameter tuning significantly improved the accuracy and robustness of our models.

Evaluation：

Model evaluation is essential to assess the effectiveness of the trained models and to determine how well they perform on unseen data. We used a variety of metrics to evaluate our models, including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of the model's performance, highlighting its strengths and weaknesses. Our best-performing model achieved an impressive accuracy of over 99%, demonstrating its high level of reliability. To further understand the relationships between different features and mushroom toxicity, we conducted a correlation analysis. This analysis helped us identify which features were most strongly related to the target variable, providing valuable insights into the factors that influence mushroom toxicity. Additionally, we established a linear regression model to quantify these relationships, allowing us to make more informed predictions and better understand the underlying patterns in the data[8][9].

Conclusions

Our project successfully developed a reliable prediction model for identifying poisonous mushrooms. The use of machine learning techniques allowed us to achieve accurate results. Future work will focus on incorporating additional features and exploring more advanced machine learning algorithms to further enhance prediction accuracy. In conclusion, By leveraging advanced data analysis methods and carefully selecting and tuning machine learning models, we were able to achieve a high level of accuracy in our predictions. This work not only provides a valuable tool for identifying poisonous mushrooms but also lays the foundation for future research in this area.

Future Work:

1.Incorporate Additional Features: Involve incorporating additional features, such as chemical composition, environmental factors, and geographical data, to further improve the accuracy and robustness of the prediction models.

2.Advanced Machine Learning Algorithms: Exploring more advanced machine learning algorithms, such as deep learning techniques (e.g., neural networks), could provide new insights and improve prediction accuracy.

3.Real-World Testing: Conducting real-world testing and validation of the prediction model in field conditions could help assess its practical applicability and reliability.

References

[1] 高帆,谭廷鸿,孙琰妮,等.毒蘑菇鉴别技术研究进展[J].食品安全质量检测学报,2024,15(09):51-61.DOI:10.19812/j.cnki.jfsq11-5956/ts.20240124004.

[2] 张超群.基于机器学习的毒蘑菇识别与研究[D].山西农业大学,2019.DOI:10.27285/d.cnki.gsxnu.2019.000146.

[3] 王鹏倞.对毒蘑菇毒素的分类与识别探讨[J].科技与创新,2018,(11):61-62.DOI:10.15913/j.cnki.kjycx.2018.11.061.

[4] 刘斌,张振东,张婷婷.基于贝叶斯分类的毒蘑菇识别[J].软件导刊,2015,14(11):60-62.

[5] A.Bachhotia, S. Dagar and A. Chaudhary, "Shrooming Prediction and Analysis using Machine Learning Algorithms," 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 1337-1340, doi: 10.23919/INDIACom61295.2024.10498367.
URL:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10498367&isnumber=10498148

[6] W.Cai, X. He, Y. Wu, J. Zhang, Y. Ran and L. Li, "Research on energy consumption prediction model based on big data platform and random forest algorithm," 2022 International Conference on Data Analytics, Computing and Artificial Intelligence (ICDACAI), Zakopane, Poland, 2022, pp. 384-388, doi: 10.1109/ICDACAI57211.2022.00082.

URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9973839&isnumber=9973442

[7] K.Kousalya, B. Krishnakumar, S. Boomika, N. Dharati and N. Hemavathy, "Edible Mushroom Identification Using Machine Learning," 2022 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2022, pp. 1-7, doi: 10.1109/ICCCI54379.2022.9741040.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9741040&isnumber=9740653

[8] A.Saryoko, E. P. Saputra, S. Nurajizah, M. Maulidah and N. Hidayati, "Product Prediction of Mushroom Agricultural Plants Using Machine Learning Techniques," 2022 International Conference on Information Technology Research and Innovation (ICITRI), Jakarta, Indonesia, 2022, pp. 184-187, doi:10.1109/ICITRI56423.2022.9970233.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9970233&isnumber=9970202

[9] S.Verma, O. Pandey, T. Choudhury, A. Sar, K. Kotecha and T. Choudhury, "A Comprehensive Study on the Classification of the Edibility of Mushrooms," 2023 12th International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2023, pp. 7-13, doi:10.1109/SMART59791.2023.10428619.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10428619&isnumber=10428128

Appendices

Appendix 1:The source code

```
from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel, Field, validator

from typing import Dict, Union
```

```
import joblib
import numpy as np
import pandas as pd
import os

class MushroomFeatures(BaseModel):
    features: Dict[str, str] = Field(
        ...,
        description="Mushroom feature dictionary, keys are feature names, values are feature values (all values should be strings)",
        example={
            "cap-surface": "x",
            "cap-color": "n",
            "does-bruise-or-bleed": "t",
            "gill-attachment": "-1",
            "gill-spacing": "-1",
            "gill-color": "w",
            "stem-height": "11",
            "stem-width": "17",
            "stem-root": "b",
            "stem-surface": "-1",
            "stem-color": "w",
            "veil-type": "u",
            "veil-color": "w",
            "has-ring": "t",
            "ring-type": "g"
        }
    )

    @validator('features')
    def validate_features(cls, v):
        required_features = [
            "cap-surface", "cap-color", "does-bruise-or-bleed",
            "gill-attachment", "gill-spacing", "gill-color",
            "stem-height", "stem-width", "stem-root",
            "stem-surface", "stem-color", "veil-type",
            "veil-color", "has-ring", "ring-type"
        ]

        # Check if all required features are present
        for feature in required_features:
            if feature not in v:
                raise ValueError(f"Missing required feature: {feature}")
            if not isinstance(v[feature], str):
                v[feature] = str(v[feature])

        return v

    class Config:
        json_schema_extra = {
            "example": {
                "features": {
                    "cap-surface": "x",
                    "cap-color": "n",
                    "does-bruise-or-bleed": "t",
                    "gill-attachment": "-1",
                    "gill-spacing": "-1",
                    "gill-color": "w",
                    "stem-height": "11",
                    "stem-width": "17",
                    "stem-root": "b",
                    "stem-surface": "-1",
                    "stem-color": "w",
                    "veil-type": "u",
                    "veil-color": "w",
                    "has-ring": "t",
                    "ring-type": "g"
                }
            }
        }

app = FastAPI(
    title="Mushroom Toxicity Prediction API",
    description="FastAPI service using pre-trained Random Forest model and ordinal encoder",
    version="1.0"
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],    # Allow all origins in development environment
    allow_methods=["*"],
    allow_headers=["*"],
)

BASE_DIR = os.path.dirname(__file__)
MODEL_PATH = os.path.join(BASE_DIR, "models", "mushroomModel.joblib")
ENCODER_PATH = os.path.join(BASE_DIR, "models", "mushroomEncoder.joblib")

try:
    clf = joblib.load(MODEL_PATH)
    encoder = joblib.load(ENCODER_PATH)
    print("Loaded feature list:", encoder.feature_names_in_)
    print("Encoder type:", type(encoder))
    print("Encoder parameters:", {
        'handle_unknown': encoder.handle_unknown,
        'unknown_value': encoder.unknown_value,
        'encoded_missing_value': encoder.encoded_missing_value,
        'dtype': encoder.dtype
    })
except Exception as e:
    raise RuntimeError(f"Failed to load model or encoder: {e}")

# Features that need to be converted to numeric values, adjust based on your training data
NUMERIC_FEATURES = ['stem-height', 'stem-width']

@app.get("/health")
def health_check():
    return {"status": "healthy"}

@app.post("/predict")
async def predict(data: MushroomFeatures):
    try:
        features_dict = data.features
        print("Received raw data:", features_dict)

        # Get encoder feature names
        expected_features = list(encoder.feature_names_in_)
        print("Expected feature order:", expected_features)

        # Preprocess data
        processed_dict = {}
        for feature in expected_features:
            if feature not in features_dict:
                raise HTTPException(
                    status_code=400,
                    detail=f"Missing feature: {feature}"
                )
            value = features_dict[feature]

            if value is None or (isinstance(value, str) and not value.strip()):

                processed_dict[feature] = '-1'
                continue

            # Handle numeric features
            if feature in NUMERIC_FEATURES:
                if str(value) == '-1':
                    processed_dict[feature] = '-1'
                else:
                    try:
                        if isinstance(value, (int, float)):
                            num = float(value)
                        else:
                            num = float(str(value).strip())
                        # Ensure value is between 0-100
                        if num < 0 or num > 100:
                            raise HTTPException(
                                status_code=400,
                                detail=f"Value for feature {feature} must be between 0 and 100, received: {value}"
                            )
                        processed_dict[feature] = str(int(num))
                    except ValueError:
                        raise HTTPException(
                            status_code=400,
                            detail=f"Feature {feature} requires numeric value, received: {value}"
                        )
            else:
                # Handle categorical features
                processed_dict[feature] = str(value).lower().strip()


            print(f"Processing feature {feature}: Original value = {value}, Processed =
{processed_dict[feature]}")


        print("Processed data:", processed_dict)


        try:
            # Create DataFrame to maintain feature names
            features_df = pd.DataFrame([processed_dict], columns=expected_features)
            print("DataFrame:", features_df)


            try:
                # Transform features using encoder
                x_enc = encoder.transform(features_df)
                print("Encoded shape:", x_enc.shape)
            except Exception as e:
                print("Encoding error:", str(e))
                print("Encoder feature names:", encoder.feature_names_in_)
                print("Input data feature names:", features_df.columns.tolist())
                raise HTTPException(
                    status_code=400,
                    detail=f"Feature encoding error: {str(e)}\nInput data: {processed_dict}"
                )


            try:
                # Make prediction
                pred_label = clf.predict(x_enc)[0]
                print("Prediction result:", pred_label)
                return {"prediction": pred_label, "status": "success"}
            except Exception as e:
                print("Prediction error:", str(e))
                raise HTTPException(
                    status_code=500,
                    detail=f"Error during prediction: {str(e)}"
                )

        except HTTPException:
            raise
        except Exception as e:
            print("Unexpected error:", str(e))
            raise HTTPException(
                status_code=500,
                detail=f"Error during processing: {str(e)}"
            )

    except HTTPException:
        raise
    except Exception as e:
        print("Unexpected error:", str(e))
        raise HTTPException(
            status_code=500,
            detail=f"Error during processing: {str(e)}"
        )

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```

Appendix 2:Code for training model.

1.Accuracy

```
import pandas as pd
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

mushroom = pd.read_csv("X:\\critical_skill\\train.csv")

label_encoder = LabelEncoder()

x_ori = mushroom.iloc[:, 2:]
remove_cols = ['season', 'habitat', 'spore-print-color', 'cap-diameter', 'cap-shape']
x = x_ori.drop(remove_cols, axis=1)

y = mushroom.iloc[:, 1]

for column in x.select_dtypes(include=['object']).columns:
    x[column] = label_encoder.fit_transform(x[column])



y = label_encoder.fit_transform(y)

Range = [0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95]
for i in Range:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=i, random_state=42)

    clf = DecisionTreeClassifier(max_depth=19)
    clf.fit(x_train, y_train)

    y_pred = clf.predict(x_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"The accuracy of {i} model: {accuracy * 100:.2f}%")
```

2.F1-score

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score


def main():
    mushroom = pd.read_csv("X:\\critical_skill\\train.csv")


    label_encoder = LabelEncoder()


    x_ori = mushroom.iloc[:, 2:]
    remove_cols = ['season', 'habitat', 'spore-print-color', 'cap-diameter', 'cap-shape']
    x = x_ori.drop(remove_cols, axis=1)


    y = mushroom.iloc[:, 1]


    for column in x.select_dtypes(include=['object']).columns:
        x[column] = label_encoder.fit_transform(x[column])


    y = label_encoder.fit_transform(y)


    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)


    clf = RandomForestClassifier(n_estimators=200, max_depth=19, random_state=42)
    clf.fit(x_train, y_train)


    y_pred = clf.predict(x_test)


    f1 = f1_score(y_test, y_pred, average='macro')
    print(f"F1 Score: {f1 * 100:.2f}%")


if __name__ == "__main__":
    main()
```

3.Grid search (decision tree)

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

mushroom = pd.read_csv("X:\\critical_skill\\train.csv")
X = mushroom.iloc[:, 2:]
y = mushroom.iloc[:, 1]

label_encoder = LabelEncoder()
for col in X.select_dtypes(include=['object']).columns:
    X[col] = label_encoder.fit_transform(X[col])
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

param_grid = {'max_depth': [3, 5, 7, 9, 11, 13, 15]}

clf = DecisionTreeClassifier()
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)

print("The greatest depth:", grid_search.best_params_['max_depth'])
```

4.Grid search2(random forest)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

mushroom = pd.read_csv("X:\\critical_skill\\train.csv")

remove_cols = ['cap-diameter', 'cap-shape','spore-print-color', 'habitat','season']
X_ori = mushroom.iloc[:, 2:]
X = X_ori.drop(remove_cols, axis=1)
y = mushroom.iloc[:, 1]

label_encoder = LabelEncoder()
for col in X.columns:
    X[col] = label_encoder.fit_transform(X[col])
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depths = range(1, 31)
accuracies = []

for depth in max_depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Max depth {depth}: Accuracy {accuracy:.4f}")

best_depth = max_depths[np.argmax(accuracies)]
```

```
print(f"Best max depth: {best_depth}")
```

5.Confusion Matrix

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


def main():
    train_data = pd.read_csv("X:\\critical_skill\\train.csv")

    to_remove = ['cap-diameter', 'cap-shape', 'spore-print-color', 'habitat', 'season']
    selected_features = [col for col in train_data.columns if col not in ['id', 'class'] + to_remove]

    X = train_data[selected_features]
    y = train_data['class']

    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y)

    ordinal_encoder = OrdinalEncoder(
        handle_unknown='use_encoded_value',
        unknown_value=-1,
        encoded_missing_value=-1,
        dtype=np.int32
    )

    X_train_encoded = ordinal_encoder.fit_transform(X_train)
    X_val_encoded = ordinal_encoder.transform(X_val)
    clf = RandomForestClassifier(n_estimators=2000, max_depth=19, max_features='sqrt', random_state=42)
    clf.fit(X_train_encoded, y_train)

    y_val_pred = clf.predict(X_val_encoded)

    cm = confusion_matrix(y_val, y_val_pred, labels=clf.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix on Validation Set")
    plt.show()

if __name__ == "__main__":
    main()
```

6.ML_train

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder
import numpy as np
import   joblib

train_data = pd.read_csv("X:\\critical_skill\\train.csv")
test_data = pd.read_csv("X:\\critical_skill\\test.csv")

to_remove = ['cap-diameter', 'cap-shape', 'spore-print-color', 'habitat', 'season']
selected_features = [col for col in train_data.columns if col not in ['id', 'class'] + to_remove]

X_train = train_data[selected_features]
y_train = train_data['class']
X_test = test_data[selected_features]
```

```
ordinal_encoder = OrdinalEncoder(
        handle_unknown='use_encoded_value',
        unknown_value=-1,
        encoded_missing_value=-1,
        dtype=np.int32                                            )

X_train_encoded = ordinal_encoder.fit_transform(X_train)
X_test_encoded = ordinal_encoder.transform(X_test)


clf = RandomForestClassifier(n_estimators=2000,max_depth=19, max_features='sqrt',
random_state=42)
clf.fit(X_train_encoded, y_train)


y_pred = clf.predict(X_test_encoded)


joblib.dump(clf, "mushroomModel.joblib")
joblib.dump(ordinal_encoder, "mushroomEncoder.joblib")


pd.DataFrame(y_pred, columns=["Predicted Class"]).to_csv("predictions1.csv", index=False)
```

7. Precision

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score


def main():
        mushroom = pd.read_csv("X:\\critical_skill\\train.csv")


        label_encoder = LabelEncoder()


        x_ori = mushroom.iloc[:, 2:]
```

```
    remove_cols = ['season', 'habitat', 'spore-print-color', 'cap-diameter', 'cap-shape']
    x = x_ori.drop(remove_cols, axis=1)

    y = mushroom.iloc[:, 1]

    for column in x.select_dtypes(include=['object']).columns:
        x[column] = label_encoder.fit_transform(x[column])

    y = label_encoder.fit_transform(y)

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

    clf = RandomForestClassifier(n_estimators=200, max_depth=19, random_state=42)
    clf.fit(x_train, y_train)

    y_pred = clf.predict(x_test)

    precision = precision_score(y_test, y_pred, average='macro')  # 多分类用 macro 平均
    print(f"Precision: {precision * 100:.2f}%")

if __name__ == "__main__":
    main()
```

8.Recall

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score


def main():
    mushroom = pd.read_csv("X:\\critical_skill\\train.csv")
    label_encoder = LabelEncoder()

    x_ori = mushroom.iloc[:, 2:]
    remove_cols = ['season', 'habitat', 'spore-print-color', 'cap-diameter', 'cap-shape']
    x = x_ori.drop(remove_cols, axis=1)

    y = mushroom.iloc[:, 1]

    for column in x.select_dtypes(include=['object']).columns:
        x[column] = label_encoder.fit_transform(x[column])

    y = label_encoder.fit_transform(y)

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

    clf = RandomForestClassifier(n_estimators=200, max_depth=19, random_state=42)
    clf.fit(x_train, y_train)

    y_pred = clf.predict(x_test)

    recall = recall_score(y_test, y_pred, average='macro')
    print(f"Recall: {recall * 100:.2f}%")

if __name__ == "__main__":
    main()
```

9.ROC curve’s code

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


def main():
    train_data = pd.read_csv("X:\\critical_skill\\train.csv")


    to_remove = ['cap-diameter', 'cap-shape', 'spore-print-color', 'habitat', 'season']
    selected_features = [col for col in train_data.columns if col not in ['id', 'class'] + to_remove]


    X = train_data[selected_features]
    y = train_data['class']


    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y)


    ordinal_encoder = OrdinalEncoder(
        handle_unknown='use_encoded_value',
        unknown_value=-1,
        encoded_missing_value=-1,
        dtype=np.int32
    )


    X_train_encoded = ordinal_encoder.fit_transform(X_train)
    X_val_encoded = ordinal_encoder.transform(X_val)


    clf = RandomForestClassifier(n_estimators=2000, max_depth=19, max_features='sqrt',
random_state=42)
    clf.fit(X_train_encoded, y_train)


    y_val_prob = clf.predict_proba(X_val_encoded)[:, 1]
    classes = clf.classes_
    fpr, tpr, thresholds = roc_curve(y_val, y_val_prob, pos_label=classes[1])
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve on Validation Set')
    plt.legend(loc="lower right")
    plt.show()

if __name__ == "__main__":
    main()
```

Appendix 3:the code for frontend interface

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Mushroom Toxicity Prediction</title>
  <style>
    body { max-width: 800px; margin: 2em auto; font-family: Arial, sans-serif; }
    label { display: flex; align-items: center; margin: 0.3em 0; position: relative; }
    label span { width: 180px; }
    input { flex: 1; padding: 0.3em; font-size: 1em; }
    button { padding: 0.5em 1em; font-size: 1em; margin-top: 1em; }
    h1, h2, p { text-align: center; }
```

```
        .tooltip {
            display: none;
            position: absolute;
            background: #f9f9f9;
            border: 1px solid #ddd;
            padding: 10px;
            border-radius: 4px;
            left: 100%;
            top: 0;
            margin-left: 10px;
            width: 250px;
            z-index: 1000;
            font-size: 0.9em;
        }
        label:hover .tooltip {
            display: block;
        }
        .error {
            border-color: red;
        }
        .error-message {
            color: red;
            font-size: 0.8em;
            margin-top: 0.2em;
        }
    </style>
</head>
<body>
    <h1>Mushroom Toxicity Prediction</h1>
    <p>Please input 15 feature values:</p>
    <ul style="font-size: 0.9em; color: #666;">
        <li>For categorical features: Enter letters or strings, or -1 for unknown</li>
    <li>For numerical features (stem-height, stem-width): Enter numbers, or -1 for unknown</li>
</ul>

<form id="predictForm">
    <div id="inputs"></div>
    <button type="submit">Predict</button>
</form>

<h2>Prediction Result: <span id="result">Waiting for input...</span></h2>

<script>
    const featureCount = 15;
    const featureNames = [
        "cap-surface",
        "cap-color",
        "does-bruise-or-bleed",
        "gill-attachment",
        "gill-spacing",
        "gill-color",
        "stem-height",
        "stem-width",
        "stem-root",
        "stem-surface",
        "stem-color",
        "veil-type",
        "veil-color",
        "has-ring",
        "ring-type"
    ];

    const numericFeatures = ['stem-height', 'stem-width'];
    const validValues = {
        'cap-surface': ['f', 'g', 'y', 's', 'x'],
        'cap-color': ['n', 'b', 'w', 'g', 'r', 'p', 'e'],
        'does-bruise-or-bleed': ['t', 'f'],
        'gill-attachment': ['f', 'a', 'c', '-1'],
        'gill-spacing': ['c', 'w', '-1'],
        'gill-color': ['w', 'p', 'b', 'n', 'g', 'r'],
        'stem-root': ['b', 'c', 'r', 'e', '-1'],
        'stem-surface': ['s', 'k', 'y', '-1'],
        'stem-color': ['w', 'p', 'g', 'b', 'n'],
        'veil-type': ['p', 'u'],
        'veil-color': ['w', 'y', 'n', 'b'],
        'has-ring': ['t', 'f'],
        'ring-type': ['p', 'e', 'l', 'g']
    };

    const tooltips = {
        'cap-surface': 'Valid values: f (fibrous), g (grooved), y (scaly), s (smooth), x (other)',
        'cap-color': 'Valid values: n (brown), b (buff), w (white), g (gray), r (red), p (pink), e (other)',
        'does-bruise-or-bleed': 'Valid values: t (true), f (false)',
        'gill-attachment': 'Valid values: f (free), a (attached), c (connected), -1 (unknown)',
        'gill-spacing': 'Valid values: c (crowded), w (wide), -1 (unknown)',
        'gill-color': 'Valid values: w (white), p (pink), b (black), n (brown), g (gray), r (red)',
        'stem-height': 'Value range: 0-100 (decimals allowed), or -1 for unknown',
        'stem-width': 'Value range: 0-100 (decimals allowed), or -1 for unknown',
        'stem-root': 'Valid values: b (bulbous), c (club), r (rooted), e (equal), -1 (unknown)',
        'stem-surface': 'Valid values: s (smooth), k (silky), y (scaly), -1 (unknown)',
        'stem-color': 'Valid values: w (white), p (pink), g (gray), b (buff), n (brown)',
        'veil-type': 'Valid values: p (partial), u (universal)',
        'veil-color': 'Valid values: w (white), y (yellow), n (brown), b (buff)',
        'has-ring': 'Valid values: t (true), f (false)',
        'ring-type': 'Valid values: p (pendant), e (evanescent), l (large), g (grooved)'
    };

    const featureDescriptions = tooltips;    // Using the same descriptions

    const inputsDiv = document.getElementById("inputs");
    const resultSpan = document.getElementById("result");

    function validateInput(feature, value) {
      if (value === '') return false;

      if (numericFeatures.includes(feature)) {
        if (value === '-1') return true;
        const num = parseFloat(value);
        return !isNaN(num) && num >= 0 && num <= 100;
      } else {
        const validFeatureValues = validValues[feature];
        if (!validFeatureValues) return true;
        return validFeatureValues.includes(value.toLowerCase());
      }
    }

    // Add input tooltips
    function addInputTooltips() {
      for (let i = 0; i < featureCount; i++) {
        const feature = featureNames[i];
        const input = document.getElementById(`feature${i}`);
        const tooltip = document.createElement('div');
        tooltip.className = 'tooltip';
        tooltip.textContent = tooltips[feature] || 'Please enter a valid value';
        input.parentElement.appendChild(tooltip);
      }
    }
```

```
// Generate input fields
for (let i = 0; i < featureCount; i++) {
    const feature = featureNames[i];
    const label = document.createElement('label');
    const span = document.createElement('span');
    span.textContent = feature + ":";

    const tooltip = document.createElement('div');
    tooltip.className = 'tooltip';
    tooltip.textContent = tooltips[feature] || 'Please enter a valid value';

    const input = document.createElement('input');
    input.type = numericFeatures.includes(feature) ? 'number' : 'text';
    input.placeholder = tooltips[feature];
    input.required = true;
    input.id = `feature${i}`;

    if (numericFeatures.includes(feature)) {
        input.setAttribute('step', '0.01');
        input.setAttribute('min', '-1');
        input.setAttribute('max', '100');
    }

    label.appendChild(span);
    label.appendChild(input);
    label.appendChild(tooltip);
    inputsDiv.appendChild(label);
}

// Add input tooltips when page loads
addInputTooltips();
```

```
// Form submit event
document.getElementById('predictForm').addEventListener('submit', async (e) => {
    e.preventDefault();
    const resultSpan = document.getElementById('result');
    const submitButton = document.querySelector('button[type="submit"]');

    try {
        // Disable submit button to prevent duplicate submissions
        submitButton.disabled = true;
        resultSpan.textContent = 'Predicting...';

        const featuresRaw = {};
        let hasError = false;

        // Collect form data
        for (let i = 0; i < featureCount; i++) {
            const feature = featureNames[i];
            const input = document.getElementById(`feature${i}`);
            let val = input.value.trim().toLowerCase();

            if (!val) {
                alert(`Please enter a value for ${feature}`);
                resultSpan.textContent = 'Waiting for input...';
                hasError = true;
                break;
            }

            if (!validateInput(feature, val)) {
                const tooltip = tooltips[feature] || 'Please enter a valid value';
                alert(`${feature} input is invalid.\n${tooltip}`);
                input.focus();
                resultSpan.textContent = 'Waiting for input...';
                hasError = true;
                break;
            }

            if (numericFeatures.includes(feature)) {
                if (val === '-1') {
                    featuresRaw[feature] = '-1';
                } else {
                    const num = parseFloat(val);
                    if (isNaN(num) || num < 0 || num > 100) {
                        alert(`${feature} must be between 0 and 100, or enter -1 for unknown`);
                        input.focus();
                        hasError = true;
                        break;
                    }
                    featuresRaw[feature] = Math.round(num).toString();
                }
            } else {
                featuresRaw[feature] = val;
            }
        }

        if (hasError) {
            submitButton.disabled = false;
            return;
        }

        // Send request
        console.log('Sending data:', {features: featuresRaw});

        const response = await fetch('http://localhost:8000/predict', {
```

```
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                },
                body: JSON.stringify({features: featuresRaw})
            });

            let data;
            try {
                if (!response.ok) {
                    const errorData = await response.json();
                    console.error('Server error response:', errorData);
                    throw new Error(errorData.detail || `Server returned status code ${response.status}`);
                }

                data = await response.json();
                console.log('Server response:', data);
            } catch (error) {
                console.error('Failed to parse response:', error);
                throw new Error(error.message || 'Server response format error');
            }

            if (!data || !data.prediction) {
                throw new Error('Server returned incorrect data format');
            }

            // Display prediction result
            resultSpan.textContent = data.prediction === 'e' ? 'Edible' : 'Poisonous';
            console.log('Prediction completed:', data.prediction);

        } catch (error) {
                console.error('Error handling:', error);
                alert(error.message || 'Error occurred during prediction');
                resultSpan.textContent = 'Prediction failed';
            } finally {
                // Restore submit button
                submitButton.disabled = false;
            }
        });
    </script>
</body>
</html>
```