Mushroom Classifier Machine Learning Project

Documentation

Version: 1.0
Date: March 27, 2025
Author: Vatsal Kathiriya (vatsalkathiriya2@gmail.com)

Table of Contents

---

1. Introduction

The Mushroom Classifier is a comprehensive machine learning project that predicts whether mushrooms are edible or poisonous based on their physical characteristics. This documentation provides detailed insights into the project's components, methodologies, and implementation.

Mushroom identification is a critical task as misidentification can lead to serious health risks. This project applies machine learning techniques to automate this process, making it accessible through an intuitive web interface. The system uses a Logistic Regression model chosen for its interpretability, efficiency, and accuracy in binary classification tasks.

---

2. Project Overview

2.1 Objectives

The primary objectives of this project are:

- To develop an accurate classification model for distinguishing edible from poisonous mushrooms
- To create an intuitive web interface for users to input mushroom characteristics and receive predictions
- To provide visualizations to understand patterns in the dataset and model behavior
- To demonstrate the application of machine learning in a real-world scenario

2.2 Project Structure

The project is structured into two main phases:

1. **Phase 1: Data Analysis and Model Training**
    - Data loading and preprocessing
    - Exploratory data analysis
    - Initial model training and evaluation
2. **Phase 2: Model Refinement and Web Deployment**
    - Feature importance analysis
    - Model refinement
    - Web application development
    - Visualization dashboard integration

2.3 Technology Stack

The project utilizes the following technologies:

- **Python**: Core programming language
- **Scikit-learn**: Machine learning library for model development
- **Pandas & NumPy**: Data manipulation and processing
- **Matplotlib & Seaborn**: Data visualization
- **Flask**: Web application framework
- **HTML/CSS/JavaScript**: Frontend web development
- **Bootstrap**: Responsive web design framework

---

3. Dataset Description

3.1 Dataset Overview

The project uses a comprehensive mushroom dataset with the following characteristics:

- **Filename**: secondary_data.csv
- **Size**: 61,069 samples
- **Source**: Based on data from Patrick Hardin's "Mushrooms & Toadstools" (1999)
- **Origin**: Simulated dataset based on 173 mushroom species with approximately 353 examples per species

3.2 Features Description

The dataset contains 20 features:

- 17 categorical features (e.g., cap shape, gill color, habitat)
- 3 numerical features (e.g., cap diameter)

Key features utilized in the simplified web interface include:

- cap-shape: Bell, Conical, Convex, Flat, Sunken, Spherical
- cap-color: Brown, Buff, Gray, Green, Pink, Purple, Red, White, Yellow
- does-bruise-or-bleed: Yes/No
- gill-color: Brown, Buff, Gray, Pink, Purple, Red, White, Yellow
- habitat: Grasses, Leaves, Meadows, Paths, Heaths, Urban, Waste, Woods
- season: Spring, Summer, Autumn, Winter

3.3 Target Variable

The target variable is class with two possible values:

- e: Edible mushrooms
- p: Poisonous mushrooms (includes mushrooms of unknown edibility, considered unsafe)

4. Project Architecture

4.1 File Structure

```
mushroom-classifier/
 ├── data/
 │   ├── secondary_data.csv      # Main dataset
 │   ├── secondary_data_meta.txt  # Dataset metadata
 │   └── primary_data_meta.txt    # Additional data information
 │
 ├── models/
 │   ├── logistic_regression_model.pkl  # Initial trained model
 │   ├── refined_mushroom_model.pkl     # Refined model for web app
 │   ├── preprocessor.pkl           # Data preprocessor
 │   └── feature_map.pkl            # Feature mapping for UI
 │
 ├── src/
 │   ├── data_preprocessing.py    # Data loading and preprocessing functions
 │   ├── model_training.py       # Model training functions
 │   ├── evaluation.py         # Model evaluation functions
 │   ├── feature_analysis.py    # Feature importance analysis
 │   └── visualization.py       # Data visualization functions
 │
 ├── static/
 │   ├── css/
 │   │   └── sweet.css         # Custom styling
 │   ├── js/
 │   │   └── sweet.js          # Form validation and animations
 │   └── img/               # Application images
 │
 ├── templates/
```

```
│   ├── index.html          # Main classification form
│   ├── result.html         # Classification results
│   ├── about.html           # Project information
│   └── visualizations.html     # Data visualizations dashboard
│
├── app.py                 # Flask web application
├── main.py                 # Main script for training and launching
└── setup.py               # Package installation script
```

4.2 Component Architecture

The project follows a modular architecture with clear separation of concerns:

1. **Data Layer**: Handles data loading, preprocessing, and transformation
2. **Model Layer**: Manages model training, evaluation, and prediction
3. **Application Layer**: Delivers the web interface and visualization dashboard
4. **Utility Layer**: Provides support functions for visualization and analysis

4.3 Data Flow

```
┌──────────────┐   ┌──────────────┐   ┌──────────────┐
│ Data Loading │──►│Preprocessing │──►│   Training   │
└──────────────┘   └──────────────┘   └──────────────┘
                                             │
                                             ▼
┌──────────────┐   ┌──────────────┐   ┌──────────────┐
│  Prediction  │◄──│  Model Save  │◄──│  Evaluation  │
└──────────────┘   └──────────────┘   └──────────────┘
      │
      ▼
┌──────────────┐
│  Web Display │
└──────────────┘
```

5. Data Preprocessing

5.1 Data Loading

The data loading process is handled by the load_data() function in data_preprocessing.py:

```
def load_data(file_path, delimiter=';'):
  """
  Load the mushroom dataset
  """
  try:
    df = pd.read_csv(file_path, delimiter=delimiter)
    print(f"Successfully loaded data with shape: {df.shape}")
    return df
  except Exception as e:
    print(f"Error loading data: {e}")
    return None
```

5.2 Data Cleaning

The following preprocessing steps are applied:

1. **Missing Value Handling**:
    - Empty strings are replaced with NaN
    - Categorical missing values are imputed with the mode (most frequent value)
2. **Target Encoding**:
    - The target variable 'class' is mapped from categorical ('e', 'p') to numerical (0, 1)

5.3 Feature Engineering

The preprocess_data() function handles:

1. **Feature Separation**:
    - Numerical and categorical features are identified automatically
2. **Feature Transformation**:
    - Numerical features: Standardization using StandardScaler
    - Categorical features: One-hot encoding using OneHotEncoder
3. **Train-Test Split**:

    - Data is split into training (80%) and testing (20%) sets with a fixed random seed for reproducibility

5.4 Preprocessor Pipeline

A scikit-learn ColumnTransformer pipeline is created to standardize preprocessing:

```
preprocessor = ColumnTransformer(
  transformers=[
    ('num', StandardScaler(), numerical_cols),
    ('cat', OneHotEncoder(drop='first'), categorical_cols)
  ])
```

This preprocessor is saved for later use in the web application to ensure consistent transformations.

---

6. Model Development

6.1 Model Selection

Logistic Regression was chosen as the classification algorithm for several reasons:

1. **Interpretability**: Coefficients directly indicate feature importance and direction
2. **Efficiency**: Fast training and prediction with low computational overhead
3. **Performance**: Strong performance on linearly separable data
4. **Simplicity**: Easy to deploy and maintain

6.2 Model Training

The model is trained using the train_logistic_regression() function:

```
def train_logistic_regression(X_train, y_train, random_state=42, max_iter=1000):
  """
  Train a logistic regression model
  """
  model = LogisticRegression(
    random_state=random_state,
    max_iter=max_iter,
    C=1.0,
    solver='liblinear'  # Works well for small datasets
  )
```

```
)

    model.fit(X_train, y_train)

    return model
```

Key hyperparameters include:

- C=1.0: Regularization strength (inverse of regularization parameter)
- solver='liblinear': Efficient algorithm for small datasets
- max_iter=1000: Maximum number of iterations for convergence

6.3 Model Evaluation

The model is evaluated using multiple metrics and visualizations:

1. **Classification Metrics**:
   - Accuracy
   - Precision
   - Recall
   - F1 Score
2. **Visualizations**:
   - Confusion Matrix
   - ROC Curve
   - Coefficient Plot

6.4 Feature Importance Analysis

Feature importance is analyzed using logistic regression coefficients:

```
def plot_logistic_coefficients(model, feature_names, top_n=20):
    """
    Plot the coefficients of a logistic regression model
    """
    # Create DataFrame of feature names and coefficients
    coefs = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': model.coef_[0]
    })

    # Sort and visualize
    coefs['Abs_Coefficient'] = np.abs(coefs['Coefficient'])
    top_coeffs = coefs.sort_values('Abs_Coefficient', ascending=False).head(top_n)

    # Plot with color coding (green for edible, red for poisonous)
    plt.figure(figsize=(10, 8))
    colors = ['#4CAF50' if c < 0 else '#F44336' for c in top_coeffs['Coefficient']]
    plt.barh(top_coeffs['Feature'], top_coeffs['Coefficient'], color=colors)
    # ...visualization code...
```

Permutation importance is also calculated to provide a robust measure of feature importance.

---

7. Web Application

7.1 Flask Application Structure

The web application is implemented using Flask in app.py:

```
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg'}
app.secret_key = 'mushroom-classifier-secret-key'
# Load the model and preprocessor
try:
    model = joblib.load('models/refined_mushroom_model.pkl')
    preprocessor = joblib.load('models/preprocessor.pkl')
    feature_map = joblib.load('models/feature_map.pkl')
    print("Model loaded successfully!")
except FileNotFoundError:
    print("Warning: Model files not found. Run Phase 2 first.")
    model = None
    preprocessor = None
    feature_map = None
```

7.2 Key Routes

The application exposes several routes:

- ***/***: Main page with the classification form

- **/predict**: Handles form submission and returns prediction
- **/about**: Information about the project
- **/visualizations**: Data visualization dashboard

7.3 User Interface

The user interface is built with HTML, CSS (Bootstrap), and custom styling:

1. **Main Classification Form**:
    - Simplified form with key mushroom features
    - Dropdown selectors for categorical variables
    - Warning information about mushroom safety
2. **Results Page**:
    - Clear visual indicator of prediction (green for edible, red for poisonous)
    - Confidence score
    - Summary of input features
    - Prominent safety warning
3. **About Page**:
    - Project description
    - Methodology explanation
    - Key statistics

7.4 Styling

Custom styling is defined in sweet.css with a mushroom-themed color palette:

```
:root {
  --primary: #8B4513;      /* Brown */
  --primary-light: #A67B5B; /* Light brown */
  --primary-dark: #5D3A14;  /* Dark brown */
  --background: #F8F5F2;    /* Cream background */
  --edible: #4CAF50;       /* Green for edible */
  --poisonous: #F44336;     /* Red for poisonous */
  --warning: #FF9800;      /* Orange for warnings */
}
```

The design is fully responsive, adapting to different screen sizes and devices.

8. Data Visualization Dashboard

8.1 Dashboard Structure

The visualization dashboard (/visualizations route) presents multiple data insights:

```
@app.route('/visualizations')
def visualizations():
    """Data visualization dashboard page"""
    # Create visualizations for the dashboard
    viz_data = create_visualizations()
    return render_template('visualizations.html', viz_data=viz_data)
```

8.2 Visualizations Included

The dashboard includes the following visualizations:

1. **Class Distribution**:
   - Bar chart showing the balance between edible and poisonous mushrooms
   - Color-coded with green for edible and red for poisonous
2. **Categorical Feature Analysis**:
   - Multiple charts showing the distribution of key categorical features
   - Split by class to show feature value distribution differences
3. **Feature Correlations**:
   - Bar chart showing how numerical features correlate with edibility
   - Color-coded to show positive (red, poisonous) and negative (green, edible) correlations
4. **Logistic Regression Coefficients**:
   - Horizontal bar chart showing model coefficients
   - Red bars indicate features that increase probability of poisonous classification
   - Green bars indicate features that increase probability of edible classification
5. **PCA Visualization**:
   - Scatter plot showing 2D projection of the dataset
   - Points colored by class (green for edible, red for poisonous)
   - Shows how well the classes separate in feature space

8.3 Implementation Details

The visualizations are generated using matplotlib and seaborn, then converted to base64 images for embedding in HTML:

```
def get_image_base64(plt):
    """Convert matplotlib plot to base64 encoded string for embedding in HTML"""
    img = io.BytesIO()
    plt.savefig(img, format='png', bbox_inches='tight')
    img.seek(0)
    plt.close()
    return base64.b64encode(img.getvalue()).decode()
```

Each visualization includes descriptive insights to help users interpret the data patterns.

---

9. Results and Evaluation

9.1 Model Performance

The Logistic Regression model demonstrates strong performance on the mushroom classification task:

- **Accuracy**: >85% on the test set
- **Precision**: High precision for both classes
- **Recall**: High recall for both classes
- **F1 Score**: Strong F1 scores, indicating good balance between precision and recall

9.2 Feature Importance

Analysis reveals the following key features for classification:

1. **Most indicative of edibility** (negative coefficients):
    - Certain gill colors (e.g., brown, white)
    - Specific habitats (e.g., woods)
    - Certain cap shapes (e.g., convex)
2. **Most indicative of toxicity** (positive coefficients):
    - Bruising/bleeding characteristics
    - Certain gill attachment types
    - Specific stem colors

9.3 Class Separability

The PCA visualization shows a clear separation between edible and poisonous mushrooms in feature space, indicating that the dataset is well-suited for binary classification.

9.4 Limitations

Despite strong performance, the following limitations should be noted:

1. **Real-world applicability**: The model should not be used as the sole determinant for mushroom consumption
2. **Dataset limitations**: Based on simulated data derived from 173 species
3. **Feature coverage**: Not all possible mushroom characteristics are captured

---

10. Installation and Usage

10.1 Prerequisites

- Python 3.6 or higher
- pip (Python package manager)

10.2 Installation

1. **Install dependencies**:

   ```
   pip install -r requirements.txt
   ```

   Alternatively, use setup.py:

   ```
   pip install .
   ```

10.3 Usage

10.3.1 Training the Model

To train the model and run the application:

```
python main.py --phase 2
```

This will:

1. Load and preprocess the data
2. Train the logistic regression model
3. Perform feature analysis
4. Save the model and preprocessor
5. Start the web application

10.3.2 Running the Web Application Only

To run the web application without retraining:

```
python main.py --run-app
```

This will load the pretrained model and start the Flask server.

10.3.3 Accessing the Application

Once running, access the application at:

- Main interface: http://127.0.0.1:5000/
- Visualization dashboard: http://127.0.0.1:5000/visualizations
- About page: http://127.0.0.1:5000/about

---

11. Future Enhancements

Potential improvements for future versions include:

1. **Model Enhancements**:
    - Implement ensemble methods for potentially improved accuracy
    - Add uncertainty quantification to predictions
    - Explore deep learning approaches for feature extraction
2. **User Interface Improvements**:
    - Add image upload capabilities for mushroom photos
    - Implement more interactive visualizations
    - Add multi-language support
3. **Extended Functionality**:
    - Expand to multi-class classification (specific mushroom species)
    - Implement user feedback loop to improve model over time
    - Create mobile application version
4. **Data Expansion**:
    - Incorporate additional mushroom datasets
    - Add more features for improved discrimination
    - Include seasonal and geographical context

---

12. Conclusion

The Mushroom Classifier project demonstrates the effective application of machine learning to the critical task of distinguishing edible from poisonous mushrooms. By leveraging Logistic Regression, the project achieves a balance of accuracy, interpretability, and computational efficiency.

The web application provides an intuitive interface for users to input mushroom characteristics and receive predictions, while the visualization dashboard offers insights into the dataset patterns and model behavior. The consistent color scheme (green for edible, red for poisonous) reinforces the classification results visually.

While the model performs well, it's crucial to emphasize that machine learning predictions should not be the sole determinant for mushroom consumption in the wild. The application includes prominent safety warnings to reinforce this message.

This project serves as both a practical tool and an educational resource, demonstrating how machine learning can be applied to real-world problems and deployed in user-friendly applications.

---

13. References

1. Hardin, Patrick. *Mushrooms & Toadstools*. Zondervan, 1999.
2. Schlimmer, Jeff. "Mushroom Data Set." UCI Machine Learning Repository, Apr. 1987, https://archive.ics.uci.edu/ml/datasets/Mushroom.
3. Wagner, Dennis. "Secondary mushroom data." September 5, 2020, https://mushroom.mathematik.uni-marburg.de/files/.
4. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
5. Flask Web Development, Miguel Grinberg, O'Reilly Media, 2018.

End of Documentation