Classification of Mushrooms under Various Algorithms

Kartik Alle
ksalle@ncsu.edu

Vasudev Menon
vmenon2@ncsu.edu

Abstract

The classification of mushrooms into edible and poisonous categories has been a topic of interest due to the potentially severe health risks associated with consuming wild mushrooms. In the real world, most methods rely on human expertise for identification tasks; however, this methodology may not always be reliable, especially when given new mushroom species that were previously undiscovered. In this paper, we propose an approach to classify mushrooms using machine learning algorithms given data about their physical features. We explore and compare the performances of three approaches: logistic regression, support vector machines (SVM), and a simple neural network. We test the aforementioned approaches to a comprehensive dataset containing information about various mushroom features and compute metrics that gauge the efficacy of each algorithm. Our results demonstrate the success of neural networks in accurately distinguishing between edible and poisonous mushrooms, outperforming the other two models.

1 Introduction

The topic of whether a mushroom is safe to eat has been one that has been debated thoroughly throughout history. Certain mushrooms have various benefits if consumed while others have detrimental effects. It is well known that we cannot determine if a mushroom is safe or harmful to eat without actually consuming or identifying it. However, using certain data and specifications about different mushrooms, we can make an educated guess. This is what we plan to do.

The existing way to classify mushrooms is to determine all of a mushroom's features and then identify them using a database of known mushrooms. This method works with all currently discovered mushrooms, but will not work with newly discovered mushrooms. In the past, humans have discovered which mushrooms are poisonous through trial and error. Nowadays, we have the technology to identify known toxins in mushrooms. However, a new toxin in a newly discovered species of mushroom might not be identified. This is where a classification algorithm to predict if a mushroom is poisonous or not can help.

As stated above, it cannot exactly be determined if a mushroom is safe to eat or not by just examining its features. However, there are features of mushrooms that could point us in a certain direction. Identifying multiple features of a mushroom can help us identify the mushroom itself and help us further classify it. Features such as a mushroom's cap, its stalk, whether it has gills, pores, or teeth underneath, its ring, its volva, and its spores come together to help identify a mushroom and determine whether it is safe to eat or not [3]. No one feature can provide a tell-tale sign about if a mushroom is safe.

The purpose of this research is to try to find a correlation between the features of a mushroom and whether it is safe to eat. Finding a correlation between sets of features and if it is safe to eat can speed up the mushroom-identifying process and possibly show which features make a mushroom more likely to be poisonous. Trying to classify mushrooms under various algorithms will rigorously test whether we can find a correlation.

2 Dataset Description

2.1 Dataset Overview

The dataset we chose contains information about mushrooms from 173 species and 23 mushroom families. The dataset simulates 353 hypothetical mushroom entries per species based on real-world mushrooms from the mushroom identification book from 2013 *Mushrooms and Toadstools*. This leads to a total of 61,069 entries. The data has 1 binary field (which represents either an edible or poisonous mushroom), 17 qualitative fields, and 3 quantitative fields. The dataset also exhibits good balance. Around 45% of the values are edible and the remaining 55% are poisonous. The data is also not linearly separable, making it hard to find a simple correlation between certain sets of features and how safe a mushroom is. Information about each variable can be found at https://mushroom.mathematik.uni-marburg.de/files/SecondaryData/secondary_data_meta.txt [5].

2.2 One Hot Encoding

The data being used has many categorical data fields, such as cap color, gill color, stalk shape, etc. To use them in classification models, such as a Linear SVM, we need to turn the data into numerical data. One hot encoding takes each categorical field in the data and turns each category into a new field. For example, if one of our fields is gill spacing (the amount of space between the gills under a mushroom's cap), we can use one-hot encoding to break up close and wide gill spacing into different categories.

Class	Gill Spacing
Safe	Wide
Unsafe	Close

Table 1: Original Table.

Class	Gill Spacing Wide	Gill Spacing Close
Safe	1	0
Unsafe	0	1

Table 2: Table after One Hot Encoding.

As shown in the tables above, one hot encoding converts the categorical data into numerical data by increasing the number of fields of data. This increases the number of dimensions of our input vectors. In general, this takes our model longer to train the data. Also, since one hot encoding creates a different field for each category of the data, it makes the data very sparse if there are a lot of categories.

3 Methodology

We want to use various algorithms to classify the mushrooms to give us a broader scope of why some algorithms might classify the mushrooms more accurately than others. This will also help us discover the best algorithm to classify these mushrooms. We plan to use the following algorithms:

- Logistic Regression to Implement Binary Classification
- Support Vector Machines
- Simple Fully Connected Neural Network

3.1 Logistic Regression

One of the models we are going to train is a simple logistic regression model. A logistic regression model maps the input vectors to a value between 0 and 1. The logistic function can be written as

$$P(\vec{x}) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}}{1+e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}} \text{ where } \vec{x} \in \mathbb{R}^n \text{ and } \vec{\beta} \in \mathbb{R}^{n+1} \tag{1}$$

The input vector is $\vec{x}$. This consists of different data fields about a specific mushroom that we want to classify. Our regression coefficients are the components of the vector $\vec{\beta}$. To train the model is to determine the beta vector that gives us the most accurate output. To determine the best $\vec{\beta}$, you must use an optimization technique. We used the Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (LMBFGS), which is SciKit Learn's default optimization algorithm for logistic regression [2]. The output $P(\vec{x})$ is a value between 0 and 1. Choose a value q where $0 < q < 1$. If $P(\vec{x}) < q$, then we can classify the input mushroom into category 0 or safe. If $P(\vec{x}) \geq q$, then we can classify the input mushroom into category 1 or unsafe. In our implementation $q = 0.5$, that is, $P(\vec{x}) \geq 0.5$ denotes a poisonous mushroom, and $P(\vec{x}) < 0.5$ denotes an edible mushroom.

3.2 Support Vector Machines

The second model that we are going to train is a Linear SVM. A Linear SVM takes the data points and attempts to draw a hyperplane through the middle of them. This splits the data into 2 categories where one can be safe mushrooms and the other can be unsafe mushrooms. The equation of a hyperplane can be defined as the following:

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n = 0 \text{ where } \vec{x} \in \mathbb{R}^n \text{ and } \vec{\beta} \in \mathbb{R}^{n+1} \tag{2}$$

If a data point $\vec{x}$ satisfies the equation above, then it lies on the hyperplane. If $\vec{x}$ lies on either side of the plane, we can classify it into one of our classes. We can classify an observation as

$$y_i = \begin{cases} -1 & \beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in} < 0 \\ 1 & \beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in} > 0 \end{cases} \text{ where } \vec{x_i} \in \mathbb{R}^n \text{ and } i = 1, 2, \ldots, m \tag{3}$$

The value m is the number of data points we are training the model on. One thing to note is that we are less confident in our model if there are many data points close to the hyperplane. The maximal margin hyperplane is the hyperplane that is farthest away from all the observations. This will give us the best model. To obtain the maximal margin hyperplane, we must train the model through optimization. We used SciKit Learn's default optimization function for the Linear SVM [2].

3.3 Simple Neural Network

The use of a simple neural network can go a long way. In our model, we used a neural network with $L = 2$ hidden layers. Our input vector is $\vec{x}^{(0)}$. Let $l = 0, 1, \ldots, L+1$. Then we can define H_l as the width of layer l. Then, the output of the l-th layer is

$$\vec{x}^{(l)} = \sigma(W^{(l)}\vec{x}^{(l-1)} + \vec{b}^{(l)}) \tag{4}$$

where $W^{(l)} \in \mathbb{R}^{H_l \times H_{l-1}}$ is the weight matrix, $\vec{b}^{(l)} \in \mathbb{R}^{H_l}$ is the bias vector, and σ is the activation function applied component wise. In our model, our input vector has 83 components. For our first hidden layer, we have 64 neurons. For our second hidden layer, we have 32 neurons. Our output layer has 1 neuron which represents either a safe or unsafe mushroom. We also chose a ReLU activation function for every layer except the last. For the last layer, we chose a sigmoid activation function. Since $\vec{x}^{(0)} \in \mathbb{R}^{83}$, our model will look like this:

$$\vec{x}^{(1)} = \sigma_1(W^{(1)}\vec{x}^{(0)} + \vec{b}^{(0)}) \in \mathbb{R}^{64} \text{ where } W^{(1)} \in \mathbb{R}^{64\times 83} \text{ and } \vec{b}^{(1)} \in \mathbb{R}^{64}$$

$$\vec{x}^{(2)} = \sigma_1(W^{(2)}\vec{x}^{(1)} + \vec{b}^{(2)}) \in \mathbb{R}^{32} \text{ where } W^{(2)} \in \mathbb{R}^{32\times 64} \text{ and } \vec{b}^{(2)} \in \mathbb{R}^{32}$$
$$\vec{x}^{(3)} = \sigma_2(W^{(3)}\vec{x}^{(2)} + \vec{b}^{(3)}) \in \mathbb{R} \text{ where } W^{(3)} \in \mathbb{R}^{1\times 32} \text{ and } \vec{b}^{(3)} \in \mathbb{R}$$

where σ_1 is the ReLU function and σ_2 is the sigmoid function such that $\sigma_1(x) = \frac{x+|x|}{2}$ and $\sigma_2(x) = \frac{e^x}{1+e^x}$ for all $x \in \mathbb{R}$. All in one equation, our model would look like this:

$$f_{NN}(\vec{x}) = \sigma_2(W^{(3)}\sigma_1(W^{(2)}\sigma_1(W^{(1)}\vec{x} + \vec{b}^{(1)}) + \vec{b}^{(2)}) + \vec{b}^{(3)}) \tag{5}$$

In practice, the architecture of our model is as follows:

Fully Connected Network		
Layer (type)	**Output Shape**	**Param #**
input_layer (InputLayer)	(None, 83)	0
dense (Dense)	(None, 64)	5376
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 1)	33
Total params: 7489 (29.25 KB) Trainable params: 7489 (29.25 KB) Non-trainable params: 0 (0.00 Byte)		

Table 3: Model summary for our Neural Network.

The idea of a neural network is to optimize the weights and bias vectors to minimize a loss function. In our implementation, we elected to utilize the binary cross-entropy loss function.

Mathematically, the binary cross-entropy loss function is defined as:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{6}$$

where N is the number of samples, y_i is the true label of the i-th sample, $\hat{y}_i$ is the predicted probability of the positive class for the i-th sample.

To minimize the above loss and to optimize the weights and biases in the network, we elected to use TensorFlow's default optimizer function, Adam [4], with the default hyperparameters.

In our implementation, we performed training for 20 epochs with a batch size of 32. The training process was executed on the TensorFlow and Keras platforms via Google Colab, leveraging a T4 GPU for accelerated computation [1].

4 Experimental Results

We split our data into a training set and a testing set. We used 70% of our data points to train our models and the remaining 30% to test them. After the training process, we tested each of the three models on the test split. The three metrics we used to evaluate how good our models were are accuracy, precision, and recall. The accuracy is the percentage of the test set that our model predicted correctly. The precision is the number of true positives over all the actual positives. The recall is the number of true positives predicted over all the positives predicted. The precision and recall can be calculated from a confusion matrix shown in the tables below. To ensure that we obtain reliable

statistics, we trained the model 10 times, with each iteration having a different randomized train-test split, and computed the aforementioned metrics by averaging their values across these runs.

An actual positive represents a value in the test set that is poisonous. An actual negative represents a value in the test set that is edible. A predicted positive value is a value in the test set that our model predicted to be poisonous. A predicted negative value is a value in the test set that our model predicted to be edible.

The following are the averaged results we obtained from 10 runs:

Table 4: Performance Metrics

Model	**Accuracy**	**Precision**	**Recall**
LogReg	0.7577	0.7871	0.7707
LinearSVM	0.7566	0.7886	0.7676
NN	0.9859	0.9872	0.9874

Table 5: Confusion Matrix for Logistic Regression

	Predicted Negative	**Predicted Positive**
Actual Negative	6066.9	2113.8
Actual Positive	2324.8	7815.5

Table 6: Confusion Matrix for Linear SVM

	Predicted Negative	**Predicted Positive**
Actual Negative	6051.7	2093.8
Actual Positive	2365.4	7810.1

Table 7: Confusion Matrix for Neural Network

	Predicted Negative	**Predicted Positive**
Actual Negative	8028.3	130.6
Actual Positive	127.9	10034.2

5 Discussion

Our project aims to classify mushrooms into edible and poisonous categories using logistic regression, support vector machines (SVM), and a simple neural network. Our results indicate that all three models achieved fairly high accuracy; however, the neural network approach significantly outperformed the other two in all three metrics: accuracy, precision, and recall.

The logistic regression and SVM models achieved comparable performance. The models attained an accuracy of around 0.75, precision of approximately 0.78, and recall hovering around 0.77 and 0.76 for logistic regression and SVM respectively. On the other hand, the neural network model achieved a value of over 0.98 for all three metrics, showing a notable improvement over the other two models. Additionally, the neural network approach had significantly lower false positive and false negative rates. In our task, it is much more important to minimize the false negative rates, since it would be much more harmful to classify a poisonous mushroom as safe than classifying a safe mushroom as poisonous. The neural network outperformed all other algorithms in this regard, with only around 128 observations resulting in a false negative, compared to over 2350 for the other models.

This all suggests that the neural network was able to learn the complex patterns and nuances exhibited

within the dataset that the other models could not. The results that a deep learning approach could be more effective compared to other methods, especially when dealing with classification tasks that contain high-dimensional and nonlinear data like mushroom features.

This disparity between the neural network model and the logistic regression and SVM models could be explained by the increased complexity in neural networks that allow them to capture higher-dimensional and nonlinear characteristics within a larger dataset. Because our encoded mushroom dataset contained over 80 columns, "lightweight" algorithms like SVM and logistic regression may struggle to capture the intricate relationships and non-linear patterns present in such high-dimensional data.

One potential drawback of using a neural network approach for our mushroom classification task is the added computation time and the lower interpretability of the model; however, our neural network architecture is lightweight, as it only contains around 30KB of parameters. Additionally, with the absence of tell-tale, human-interpretable indicators of mushroom safety, the "black-box" nature of our neural network does not compromise anything in this regard, thus, we have deemed the neural network approach to be the best.

6 Conclusion

In this paper, we aimed to classify mushrooms as safe to eat or poisonous. To do so, we used 3 different models: logistic regression, SVM, and a neural network. Out of the three, the neural network outperformed the other models. Our research also exposes the limitations of simpler models like logistic regression and SVM. With simpler tasks, where the input vectors have fewer components, they perform well, but they show their age with more complex tasks. A big goal of our research was to reduce the number of false negatives as they are much more harmful to a person than a false positive. The neural network significantly outperformed the other models in this metric, highlighting the difference in performance.

Going forward, improving upon our neural network approach might yield even better results. We only used a simple neural network, but a deeper network, with possibly even more complex computations such as regularization or normalization, could yield better results. Our research could be a basis for others to improve upon. Other researchers can experiment with creating deeper neural networks and other models to classify the data while using our results as a benchmark for their models.

The importance of classifying mushrooms as edible or poisonous cannot be understated. Mushrooms are an integral part of many people's lives and helping classify them could have many benefits. Creating a model that accurately classifies mushrooms can help us classify newly discovered mushrooms in the future.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[3] Ellen Crocker and Nicole Ward Gauthier. Don't eat those wild mushrooms. https://plantpathology.ca.uky.edu/files/ppfs-gen-14.pdf, 2016.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Dennis Wagner, Dominik Heider, and Georges Hattab. Mushroom data creation, curation, and simulation to support classification tasks. https://doi.org/10.1038/s41598-021-87602-3, 2021.