An Implementation of a Decision Tree to Classify Poisonous and Edible Mushrooms

Jon David* and Jarrett Decker*

***Abstract*—Mushrooms are delectable ingredients that are difficult for amateurs to forage and cultivate. A common problem amateur mushroom foragers face is identifying edible and poisonous mushrooms. Incorrectly classifying a mushroom as edible could lead to mushroom poisoning, with symptoms ranging between gastrointestinal discomfort to death. In this paper we discuss a machine learning approach to classify these mushrooms. We implement and train decision trees and evaluate their performance over several parameters such as attribute-selection criteria, and confidence level. The experiments show that after training, all of the generated decisions trees perform very well. They all produce 99% accuracy on the validation set.**

I. Introduction

Machine learning is an approach to problem solving that involves training a model on a large set of examples to extract generalized knowledge and to make predictions on new examples. It requires a well-specified task, a measure of performance, and a training examples [1].

The classification of poisonous and edible mushrooms is a well-specified task. It's measure of performance is the accuracy of its predictions. And a large dataset of 8124 total mushroom instances is available, 4062 of which is allocated as training examples. This means that a machine learning approach is a viable option to solving the task of classifying poisonous and edible mushrooms. Decision trees in particular are suitable for this task [1] because data is represented as attribute-value pairs and, the target function's range is a set of categorical values.

One of the main advantages of using decision trees is their reasoning mechanism is easy to understand. When a decision tree classifies an instance its reasoning mechanism can be traced back from the leaf node to the root.

Another advantage decision trees have is their structure. Their tree structure allows for classification in $\mathcal{O}(log_v n)$ time, where v is the maximum number of valid values for any given attribute.

II. Design and Implementation

As per the project instructions our Decision Tree is modeled as the ID3 algorithm. In an effort to maintain good software engineering practices, we also chose to use object oriented programming methodologies when coding our Decision Tree. Our first challenge was getting the data in a structure that we could manage and manipulate. We then started defining our main classes for the tree. We implemented both entropy and misclassification error for our information gain metrics. After we had information gain we could then construct the tree. Pruning via chi square calculations was the next mechanic needed. We then were able to create our classification functionality.

e-mail: jdavid@cs.unm.edu *authors contributed equally
e-mail: jdeck069@unm.edu *authors contributed equally

A. Data Structures

The ShroomDefs class abstracts the definition of a mushroom instance. It defines the set of valid class values, the set of valid attributes, and the set of valid symbols for each attribute. This approach would also allow us to possibly define new types of data sets at runtime, or to define a new set of attribute and valid values simply by defining a new file.

The ShroomRecord and ShroomDatabase classes abstract the dataset and provide functions to access data without having to understand its underlying implementation. For example, the database class provides functions to access an attribute columns which make it easier to calculate entropy because the task of extracting those values has already been taken care of.

The ID3Tree is composed of ID3Nodes and ID3Edges. The connectivity between nodes and edges is defined by an adjacency matrix. The ID3Nodes have subclasses ID3DecisionNode and ID3LeafNode. ID3LeafNodes are a simple class that keep track of only their classification class. ID3DecisionNodes keep track of which attribute they split on, how much information they gain, and their chi-squared value. ID3Edges are composed of a source and destination node, and their branch attribute and branch value.

The ID3Tree class, and the ID3Nodes and ID3Edges classes it contains, abstract the implementation of adding nodes and trees to existing trees, making it easier for the ID3 algorithm to build a tree. The ID3Tree also abstracts the mechanism of classifying instances.

B. ID3 Algorithm

The algorithm generates an ID3Tree. It first checks for homogeneity among the dataset's class values. If the dataset's class values are pure, (i.e., homogeneous), ID3 returns a new subtree with a single ID3LeafNode that classifies that dataset.

If the dataset is not homogeneous but the set of attributes is empty, that means there are no more attributes to split off of. ID3 returns a new subtree with a single ID3LeafNode that classifies the majority of the classes in the dataset.

Otherwise, ID3 uses a attribute-selection criteria (either information-gain or misclassification-error) to determine the next attribute to split on. When an attribute is found the ID3 algorithm recursively calls itself to create nodes out of the split portions of the database and finishes either when each node

hits a pure set or when the χ^2 test halts splitting. This function returns a tree object which can then be used for classification.

Our implementation is based on the ID3 algorithm described in [2] and with our own version defined in Algorithm 1.

Algorithm 1 id3(C, S, T, A, α)

▷ % Input: attribute-selection criteria C, %
▷ % Input: set of examples S, target attribute T %
▷ % Input: set of attributes A, 1-confidence α %
if S's class is homogeneous **then**
  $label \leftarrow$ mode(S['class'])
  return ID3Tree(ID3LeafNode($label$))
end if
if A = $\emptyset$ **then**
  $label \leftarrow$ mode(S[T])
  return ID3Tree(ID3LeafNode($label$))
end if
Let R be C's recommended attribute
$decisionnode \leftarrow$ ID3DecisionNode(R)
$tree \leftarrow$ ID3Tree($decisionnode$)
for $v \in$ val(R) **do**
  $\chi^2 \leftarrow$ calculate χ^2 using R, S
  $dof \leftarrow$ (number of valid values in A) - 1
  if not ShouldPrune(χ^2, dof, α) **then**
    $label \leftarrow$ mode($S[T]$)
    return ID3Tree(ID3Leaf($label$))
  end if
  $edge \leftarrow$ new ID3Edge(R, v)
  Let $S_v \subseteq S$ such that $S[R]$ = v
  if S_v = $\emptyset$ **then**
    $label \leftarrow$ mode(S['class'])
    $leafnode \leftarrow$ ID3LeafNode[$label$]
    $tree$.add($decisionnode$, $edge$, $leafnode$)
  else
    $subtree \leftarrow$ id3(C, S_v, T, A-{R}, α)
    $tree$.addtree($decisionnode$, $edge$, $subtree$)
  end if
  return $tree$
end for

C. Attribute Selection

Attribute selection is the process of selecting the attribute which best partitions the dataset such that the subset datasets become more homogeneous and easier to classify.

Two methods implement attribute selection. The first method seeks to maximize information gain and the second seeks to minimize misclassification error.

1) Attribute selection via information gain: Attribute selection via information gain makes use of an idea borrowed from information theory.

We use the definition of entropy as defined in [1]:

$$Entropy(S) = \sum_{v \in val(A)} -p_i log_2(p_i) \tag{1}$$

Where S is the dataset at the current node, A is the current attribute of interest, and p_i is the proportion of samples $\in S$ whose attribute value equals v.

The entropy function is used when calculating the information gain on each attribute. We use the definition of information gain as defined in [1]:

$$Gain(S, A) = Entropy(S) - \sum_{v \in val(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2}$$

The best attribute is the one with the greatest information gain. The information gain algorithm is defined in algorithm 2.

Algorithm 2 calculate information gain(A, S)

▷ %Input: attribute A, dataset S%
$entropybefore \leftarrow$ Entropy(S)
$entropyafter \leftarrow$ 0.0
for $v \in$ vals(A) **do**
  Let $S_v \subseteq S$, such that $S[A]$ = v
  $localentropy \leftarrow \frac{|S_v|}{|S|} \times$ Entropy(S_v)
  $entropyafter \leftarrow entropyafter$ + $localentropy$
end for
$gain \leftarrow entropybefore$ - $entropyafter$
return $gain$

2) Attribute selection via misclassification error: Attribute selection via misclassification error criteria is a more naïve method. But it's easier to calculate. We define the misclassification error as follows:

$$M_E(S, A) = 1 - max(p_a, p_b, ..., p_k) \tag{3}$$

Where, S is the dataset available at the current node, A is the attribute we are calculating the misclassification error for, and p_x is the proportion of samples in S such that $S[A]$ = x.

After calculating the misclassification error for each attribute, the best attribute is the one that offers the minimal misclassification error.

Algorithm 3 calculate misclassification error(A, S)

▷ %Input: attribute A, dataset S %
Let T be a frequency table
Let P be a table of proportions
for each $v \in$ val(A) **do**
  for each $example \in S$ **do**
    $T[v] \leftarrow T[v]$ + 1, if $example[A]$=v
  end for
  $P[v] \leftarrow T[v]$ / $|S|$
end for
return 1-max($\{p \mid p \in P\}$)

D. Overfitting and Branch Pruning

A tendency for decision tree models is to overfit the training examples [1]. Overfitting means that the model performs well when classifying examples it has seen before, but performs poorly when attempting to classify new, unseen, examples.

One method to overcome overfitting is through pruning. [2] proposes the use of the chi-square test for stochastic independence to test whether or not there is indeed an advantage gained by splitting.

$$\chi^2 = \sum_{v \in val(A)} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(e_i - e'_i)^2}{e'_i} \tag{4}$$

Where,

$$p'_i = p \times \frac{p_i + e_i}{p + e} \tag{5}$$

When the calculated χ^2 value is greater than the value provided by the χ^2 distribution table, continue splitting the tree. Otherwise, stop splitting. This method is called split-stopping and is one method of preventing a decision tree from overfitting.

E. Classification

Classification is the process of using a trained model to make predictions on new examples. Specifically the decision tree predicts whether a mushroom is poisonous or edible.

Because this model is a tree, it is simple to define a recursive definition to classify an example. If the current node is a leaf, return that leaf's classification; otherwise, it is a decision node. Ask each of this decision node's children to classify themselves.

Algorithm 4 classify($node$, $example$)

▷ % Input: ID3Node $node$, $example$ %
if $node$ is $None$ **then**
  return most common class in this tree
end if
if $node$ is a ID3Leaf **then**
  return $node.classification$
end if
$nextnode \leftarrow None$
for each $edge$ emerging from $node$ **do**
  $attribute \leftarrow edge.branchattribute$
  if $edge.branchvalue$ = $example[attribute]$ **then**
    $nextnode \leftarrow edge.destinationnode$
    break
  end if
end for
return classify($nextnode$, $example$)

III. Experiments

A. Data

The data used to train the decision tree model can be found in the University of California, Irvine's Machine Learning Repository [3]. It is a multivariate dataset consisting of 22 categorical attributes that describe mushroom characteristics and a label. In this dataset a mushroom is labeled to be poisonous or edible. This type of dataset is appropriate for classification tasks.

The dataset in the repository contains 8124 instances. These instances are partitioned into three files: (1) a training set with 4062 instances (50%), (2) a testing set with 2031 instances (25%), and (3) a validation set with 2031 instances (25%).

B. Methods

Decision trees are built using the ID3 algorithm described in the implementation section. Different decision trees are built by specifying two parameters: (1) attribute-selection criteria, which determines which attribute will be used to further split the tree, and (2) confidence-interval, which specifies how much certainty is required before continuing to split the decision tree. There are two attribute-selection criteria: information-gain and misclassification-error; and four confidence intervals of interest: 99%, 95%, 50%, and 0%. The 0% confidence interval means the tree will always be fully grown, never pruned.

From the two attribute-selection criteria and four confidence intervals eight decision trees are built and evaluated.

C. Results

All models performed equally well over the test and validation sets. In addition, it was a surprise that no trees were pruned. The table below shows the confusion matrix shared by all 8 models.

TABLE I
Confusion Matrix

	Predicted num(p)	Predicted num(e)
Actual num(p)	985	0
Actual num(e)	0	1048

A confusion matrix is more descriptive of a model's performance than accuracy. A confusion matrix shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). From this accuracy and misclassification can be calculated. Accuracy is (TP+TN/p+e) and Misclassification is 1.0-Accuracy.

In contrast, when evaluating the model over a validation set the label is not given. The only information provided is accuracy. The table below shows the classification accuracy of each model over the validation set.

TABLE II
Classification Accuracy

Criteria	α=0.01	$\alpha = 0.05$	$\alpha = 0.50$	α=0.0
information-gain	99.95%	99.95%	99.95%	99.95%
misclass-error	99.95%	99.95%	99.95%	99.95%

Although the trees built using the information-gain criteria performed equally as well as the trees built using the misclassification-error criteria, the two trees are slightly different. The trees built using the information-gain criteria contained only 38 nodes with a max depth of 4, whereas the trees built using the misclassification-error criteria contained 45 nodes with a max depth of 5. Even though the two types of trees performed equally well, the simpler tree with fewer nodes and smaller depth is the preferred model.

IV. Discussion

A. Explanation of Results

All eight trees produced from information-gain criteria, misclassification-error criteria, α=0.01, α=0.05, α=0.50, α=0.0 perform equally well. It is surprising but this may be because the dataset is well suited for decision tree learning. Decision tree learning is best suited to problems where instances are represented as attribute-value pairs, and the target function has discrete output values [1]. These are characteristics found in the mushroom database.

B. Proposed Classification Rules

The Appendix contains the complete set of classification rules constructed from one of our decision trees (one built using the information-gain critiera and α=0.01).

A subset of those rules that identify poisonous mushrooms is listed here: (1) if the mushroom smells creosote, musty, foul, pungent, fishy, or spicy, then it is poisonous; (2) if the mushroom has no scent and its spore-print-color is green, then it is poisonous; (3) if the mushroom has no odor, has a white spore-print-color, and its stalk-root has a club shape, then it is poisonous; (4) if the mushroom has no odor, has a white spore-print-color, and its gill-size is narrow, then it is poisonous; and finally (5) if the mushroom has no odor, has a white spore-print-color, a bulbous stalk-root, and a white cap, then it is poisonous.

V. Conclusions

Given the performance of the decision trees it is tempting to suggest that the classification of poisonous and edible mushrooms is an easy task. However, the consumption of an incorrectly identified edible mushroom can be fatal. It is important to remember that models are representations of the things they are modeling and that no model is ever 100% complete.

References

[1] T. M. Mitchell *et al.*, *Machine learning. wcb*, 1997.

[2] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[3] J. Schlimmer, "Mushroom records drawn from the audubon society field guide to north american mushrooms," *GH Lincoff (Pres), New York*, 1981.

APPENDIX

These are the classification rules produced by a decision tree using information-gain criteria and α=0.01. These rules are equivalent to the rules produced for other information-gain trees with α=0.05, α=0.50, and α=0.0.

```
  [R1] IF ((odor=c)), THEN p.
  [R2] IF ((odor=m)), THEN p.
  [R3] IF ((odor=l)), THEN e.
  [R4] IF ((odor=f)), THEN p.
  [R5] IF ((odor=p)), THEN p.
  [R6] IF ((odor=a)), THEN e.
  [R7] IF ((odor=y)), THEN p.
  [R8] IF ((odor=s)), THEN p.
  [R9] IF ((odor=n) AND (spore-print-color=h)), THEN e.
  [R10] IF ((odor=n) AND (spore-print-color=o)), THEN e.
  [R11] IF ((odor=n) AND (spore-print-color=r)), THEN p.
  [R12] IF ((odor=n) AND (spore-print-color=b)), THEN e.
  [R13] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=c)), THEN p.
  [R14] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=?) AND (gill-size=b)),
THEN e.
  [R15] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=?) AND (gill-size=n)),
THEN p.
  [R16] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=r)), THEN e.
  [R17] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=z)), THEN e.
  [R18] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=c)),
THEN e.
  [R19] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=r)),
THEN e.
  [R20] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=b)),
THEN e.
  [R21] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=p)),
THEN e.
  [R22] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=w)),
THEN p.
  [R23] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=y)),
THEN e.
  [R24] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=n)),
THEN e.
  [R25] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=e)),
THEN e.
  [R26] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=u)),
THEN e.
  [R27] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=b) AND (cap-color=g)),
THEN e.
  [R28] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=u)), THEN e.
  [R29] IF ((odor=n) AND (spore-print-color=w) AND (stalk-root=e)), THEN e.
  [R30] IF ((odor=n) AND (spore-print-color=y)), THEN e.
  [R31] IF ((odor=n) AND (spore-print-color=n)), THEN e.
  [R32] IF ((odor=n) AND (spore-print-color=u)), THEN e.
  [R33] IF ((odor=n) AND (spore-print-color=k)), THEN e.
```