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

Jon David* and Jarrett Decker*

***Abstract*—The problem we have been asked to solve is to classify mushrooms as either edible or poisonous. Our data contains 22 different non-numeric attributes to help us in this endeavor. This is a perfect candidate for the use of a Decision Tree and will allow us to hopefully come up with a few rules to help us identify poisonous mushrooms in the real world.**

I. Background on Decision Trees

We should cite [1] here. Typically, a machine learning algorithm works by taking a known set of data and devising a function that generalizes it. Computers allow us to define multidimensional data sets and find these functions almost effortlessly. One of the drawbacks of these approaches, however, is that these functions are usually incomprehensible to humans. Most machine learning algorithms also only deal with numerical data. Decision Trees are a method which overcomes both of these shortcomings. Decision Trees create a system of rules that a human could read and understand, and when the rules are applied to new data, allows for fast classification.

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.

A. Data Structures

Data sets lie at the heart of all machine learning methods so being able to work with the data is very important. The approach we took was to create two classes, a database class, and a datadef class. The database class would essentially be a list of records created from the raw data set we were given. A record is a set of dictionaries where the dictionary key is a string representing an attribute and the value is the value of that attribute, for every attribute of a single row in our data set. The datadef class reads in a definition file that lists every attribute and every possible valid symbol for that attribute. This was useful for iterating through all possible symbols of an attribute. This approach would also allow us to possibly define new types of data sets at runtime, instead of having to hard code attribute definitions.

We also wrote a few helper functions to grab a single column of a database. For instance, grabbing a vector of just the labels of each record in a database was an easy way to put together an output file after classification of that database.

B. Tree Building

Our tree is made up of a few classes. The ID3Tree class is the highest level class which is composed of ID3Nodes and ID3Edges. The ID3Node class is inherited from by the ID3DecisionNode and ID3LeafNode classes. Leaf nodes only keep track of the label they represent while decision nodes know what attribute they split on, how much information gain they represent, and their chi squared value. Edges have a source node and a destination node, the attribute that caused the split, and what value specifically this edge represents. This infrastructure allowed us to see exactly how the tree was architected and comprehend what was going on at any point in the tree. The ID3Tree class implemented most of the functions needed to construct the tree as well as interact with it through classification.

C. ID3 Algorithm

With our infrastructure, information gain functions, and our chi square check, we could now create a decision tree with the ID3 algorithm. The algorithm works by taking in a database object, an information gain object, attribute definitions, and a chi square table with a confidence interval. A node is created for the database and then the database is checked for class impurities. If the database only has a single label then it is deemed pure and the node is returned as a leaf node. If the node isn't pure then the database is analyzed by the information gain class to find a candidate attribute to split on.

Stuff with chi square happens too.

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 chi square 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 is defined in Algorithm 1.

D. Attribute Selection

The two methods of information gain we needed to use were entropy and misclassification error. We created two classes

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

Algorithm 1 id3(C, D, T, A, α)

if D's class is homogeneous **then**
  $label \leftarrow$ mode(D['class'])
  return ID3Tree(ID3LeafNode($label$))
end if
if A = $\emptyset$ **then**
  $label \leftarrow$ mode(D[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, D
  $dof \leftarrow$ (number of valid values in A) - 1
  if ShouldPrune(χ^2, dof, α) **then**
    $label \leftarrow$ mode($D[T]$)
    return ID3Tree(ID3Leaf($label$))
  end if
  $edge \leftarrow$ new ID3Edge(R, v)
  Let $D_v \subseteq D$ such that $D[R]$ = v
  if D_v = $\emptyset$ **then**
    $label \leftarrow$ mode(D['class'])
    $leafnode \leftarrow$ ID3LeafNode[$label$]
    $tree$.add($decisionnode$, $edge$, $leafnode$)
  else
    $subtree \leftarrow$ id3(C, D_v, T, A-R, α)
    $tree$.addtree($decisionnode$, $edge$, $subtree$)
  end if
  return $tree$
end for

to deal with this, InformationGainCriteria and ClassificationErrorCriteria. The purpose of these classes were to be able to read through a database and then produce the best attribute candidate for splitting the database at the given node. Both of these methods work in similar ways. Entropy is calculated as -P(x)log(P(x)) and InformationGainCriteria would take the entropy at the parent node and subtract the weighted entropy of each of the child nodes after a simulated split for each possible attribute.

1) Selection via information gain: Algorithm for selecting the best attribute using information gain.

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

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

Equation for information gain:

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

We select the attribute that offers the greatest information gain.

2) Selection via misclassification error: Algorithm for selecting the best attribute using misclassification error.

Equation for misclassification error.

$$M_E = 1 - max(p_1, p_2, ..., p_n) \tag{3}$$

We select the attribute that offers the minimal misclassification error.

E. 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,

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

F. Classification

Stuff about classification. Now here's the classify algorithm.

Algorithm 2 tree.classify(node, record)

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 $edge.branchvalue$ = $record[attribute]$ **then**
    $nextnode \leftarrow edge.destinationnode$
    break
  end if
end for
return classify($nextnode$, $record$)

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

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.