DECISION TREE CLASSIFIER FOR MUSROOMS DATASET

Laura Soto
University of Miami
lms338@miami.edu

ABSTRACT

This project aims to detect if a mushroom is poisonous or not using a decision tree classifier model built from the "mushroom.cvs" dataset. The project proposes two different approaches to build the model: the first one uses a ratio split of 70/30, and the second one uses 5-fold cross-validation. In order to build both classification models, the high-level language used was Python and its libraries. The decision tree classifier models obtained are successful, with an accuracy of 100% and 99%. The accuracy of the model is evaluated through the model accuracy scores, a classification report, and the confusion matrix.

1. INTRODUCTION

Mushrooms are an important type of funguses, which contains numerous qualities, but on the other hand, some can be toxic, and eating them can be dangerous. The "mushroom.cvs" dataset contains a description of hypothetical samples corresponding to twenty-three species of gilled mushrooms. Each mushroom species is either classify as edible or poisonous. By using classification learning, we can create a model that would be trained with the given data and would be able to predict the class label of a given mushroom specie. The classification learning model that is going to be used in order to achieve this goal, is a decision tree classifier, which displays the resulting model as a step by step tree. The models would be tested by its accuracy score when classifying unlabeled instances. Finally, if the results are successful when testing the model, we would be able to conclude that the model works, and we were able to create a model that differentiates the edible from the poisonous mushrooms.

2. DATA

The data used to build the models is the "mushroom.cvs" dataset which was initially donated to the UCI Machine Learning repository in 1987. The data set contains 8124 instances, 22 attributes, and one class label. All the attributes are qualitative. From these 22 attributes, the ones that are also discrete and nominal because their values are fixed and don't have any ordering are; *Cap-Shape, Cap-Surface, Cap-Color, Odor, Grill-Attachment, Grill-Spacing, Grill-Color, Stalk-Root, Stalk-Color-Above-Ring, Stalk-Color-Below-Ring, Veil-Color, Ring-Type, Spore-Print-Color*, and *Habitat.* The attributes that are discrete and ordinal because they have some order are; *Stalk-Surface-Above-Ring, Stalk-Surface-Below-Ring, Ring-Number,* and *Population*. Not all the attributes are discrete, we have one attribute that is binary and ordinal: *Grill-Size*, and three attributes that are binary and nominal: *Bruises, Stalk-Shape*, and *Veil-Type*. Since the *Class label* can only have two values, edible or poisonous, it is binary and nominal. Below is a graphic representation pie of how the class values are distributed among the 8124 instances.

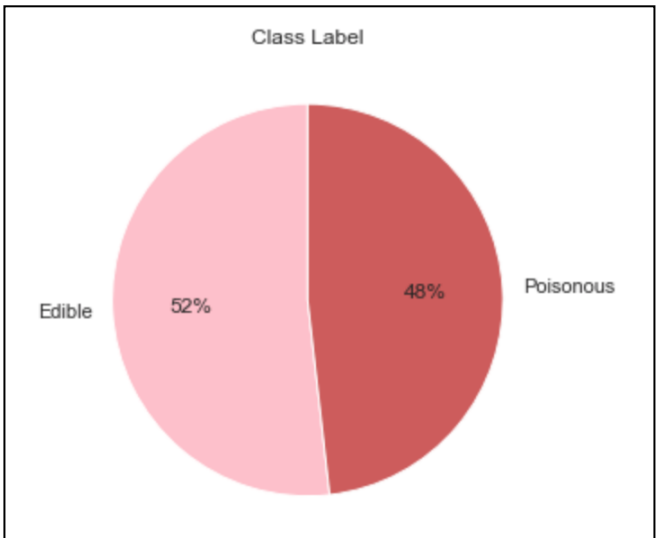

Fig. 1. Visualization of the class label data distribution in "mushroom.cvs" dataset.

3. DATA PRE-PROCESSING

In order to build a Data Mining model, the first step in the process is to do some data preprocessing. Data preprocessing is the step where data gets transformed or encoded. Moreover, represents the transformation of raw data into an understandable format. We first need to check for missing values, and then make sure that all the data can be used for training and testing the model. Due to the fact, that all our attributes are qualitative, we need to encode the

data in order to transform nominal values into numeral values.

3.1 Missing values

On this particular dataset, we do not have any value equal to null. On the other hand, we do have a value equal to '?' for the *Stalk-Root* attribute. This value represents 30% of the total instances for the attribute, equivalent to 2480 examples. This value could be interpreted as a missing value, and if missing values are not handled properly the model could end up not being accurate. In order to manage missing values for nominal data, we can consider many different techniques. The first technique that we considered is to replace the missing values with the most common value for this attribute. By counting the number of times that each value appears on this attribute, we found that the most common value for the Stalk-Root example is 'b' with a total number of 3776 instances. If we replace the missing instances '?' with 'b' we would obtain a total count for the instance 'b' equal to 6256 instances out of 8124, which equivalates to 77%. Even though this is a valid approach, because of the high number of instances that would have the same value, this approach would not be implemented for this project. The second approach that was considered, is to include the missing value as a valid value. Due to the fact that this would create a more normally distributed data, the '?' value would be considered as a valid value for the Stalk-Root attribute.

3.2 Nominal to numeric

In order to build the classification learning model, the Python Skirt-learn library is going to be used. On the other hand, the decision trees implemented using this library uses only continuous numerical values. Therefore, the transformation of our data from nominal values to numerical values is needed. In order to achieve this transformation, we are going to apply the Hot Encoding technique, which created a vector size with numerical values for each nominal value. By encoding our dataset each attribute values would be transformed into numeric, and we would obtain an equivalent distributed numerical dataset. For our class label, we do not need to modify the values into numerical values, so they would keep their original type. Finally, after this process is completed the data is ready to be used for creating the model.

4. DECISION TREE CLASSIFICATION

Decision tree learning is a predictive model used in data mining. This approach uses a decision tree to go from the observations/conditions to the conclusions. A decision tree classifier has the goal of predicting the class label, which is represented in the leaves, and the branches represent the conjunctions of conditions that lead to that conclusion.

In order to build a decision tree classifier, we need to use classification learning rules. Classification learning is a supervised machine learning approach, that creates the classification rules by following three basic steps; first, the dataset has to be split into testing and training, second, the model has to be trained using the training set, and third, the model has to be tested on the testing set. In this project two different approaches are going to be used for training and building the model, the first approach uses a data split ratio of 70/30, and the second approach uses 5-fold cross-validation.

4.1 Decision Tree Classifier Model: Ratio Split

The first decision tree classifier model is built based on a training data set composed of 70% of the "mushroom.cvs" dataset. The data is randomly distributed, and the classification decision three is created by fitting this training data into the model. It is important to know that this is a supervised learning, so when creating the model, this has to be fitted with both the dataset instances and the class label. This model uses a rule-based approach, which tries to identify the relationship between the conditions, and the conclusions present on the given data. That is why the conclusion of this process, is a decision tree model that has the rules/relationships founded from the training data as branches and the class label as the leaves. Moreover, the performance obtained from this first model, when predicting the class label for the training set, can be evaluated as:

```
Model Results:

Accuracy Score: 100%

Classification Report:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00      2974
           p       1.00      1.00      1.00      2712

    accuracy                           1.00      5686
   macro avg       1.00      1.00      1.00      5686
weighted avg       1.00      1.00      1.00      5686


Confusion Matrix:
[[2974    0]
 [   0 2712]]
```

Fig. 2. Results obtained from the decision tree classifier using a 70/30 data split, and testing it on the training data

Now that we have built a model using the training set, the next step is to test the model. For this purpose, we use the remaining 30% of the data. During this step, the model, created by evaluating the training set, is used to predict the class label of the testing set. In order to evaluate the performance of the classification predictions, we compare the classified class labels obtained thought the tree, with the real class labels, and we can evaluate the performance as:

```
Model Results:

Accuracy Score: 100%

Classification Report:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00      1234
           p       1.00      1.00      1.00      1204

    accuracy                           1.00      2438
   macro avg       1.00      1.00      1.00      2438
weighted avg       1.00      1.00      1.00      2438


Confusion Matrix:
[[1234    0]
 [   0 1204]]
```

Fig. 3. Results obtained from the decision tree classifier using a 70/30 data split, and testing it on the training data

Visually, we can represent the decision making of the model created on the following decision tree as:

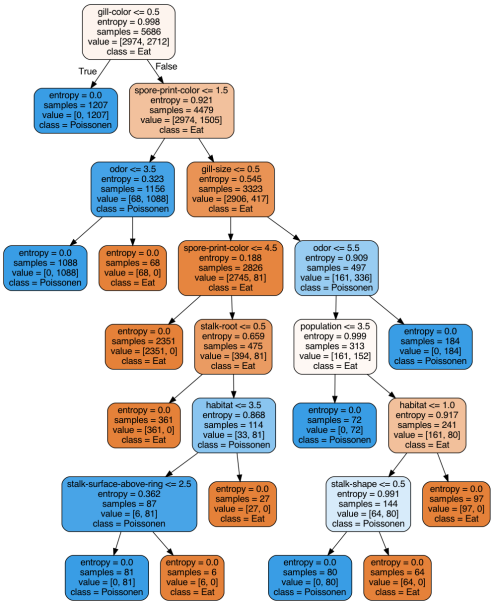

Fig. 4. Visual representation of the decision tree classification created with the 70/30 ratio split.

To better understand the visual representation of this model, it is important to understand what information is given on each node. The features that are the most relevant to classify unclassified instances are near the root, and the attribute that is considered to be the most relevant is the root. This is because the decision tree classifier chooses the highest information gain to split the tree. As we can notice, not all 22 attributes are represented on this tree, and this means that they do not affect the decision, so they are not considered when determine the class. The entropy measures how unpredictable a dataset may be.

4.2 Decision Tree Classifier Model: 5-Fold Cross Validation

The second decision tree classifier model uses 5-fold cross validation to train and create the model. Cross validation is a technique that is used to avoid overfitting, which refers to the creating of a model that has a very good performance on the training data, but not on the testing, or other data. Cross validation refers to the randomization of splitting the data in order to make sure that the model takes into account all the variations of the data and to avoid bias. Through k-fold cross validation, the original data set would be splitting in k subsets. One of the subsets would be used for testing and the others for training. This splitting process would be iterated k times until every subset has being used once for testing. Our model is created using k=5, so the data is split it into 5-folds. We recorded the accuracy obtained when predicting the class for the training set, and it can be represented as followed:

```
Model Results on Training Set:
Accuracy of Each Iteration: [100.0, 100.0, 100.0, 100.0, 100.0]
Average Accuracy: 100.0%
Standard Deviation: 0.00
```

Fig. 5. Accuracy obtained on each iteration when testing the training set. As well as the average accuracy and the standard deviation

The 5-fold cross validation model would consider to hsave and accuracy equal to the average accuracy. Furthermore, we tested the model on the testing set, and we calculated the accuracy of the predicting set as followed:

```
Model Results on Training Set:
Accuracy of Each Iteration: [100.0, 100.0, 100.0, 99.38461538461539, 96.55172413793103]
Average Accuracy: 99.19%
Standard Deviation: 1.34
```

Fig. 6. Accuracy obtained on each iteration when testing the testing set. As well as the average accuracy, and the standard deviation.

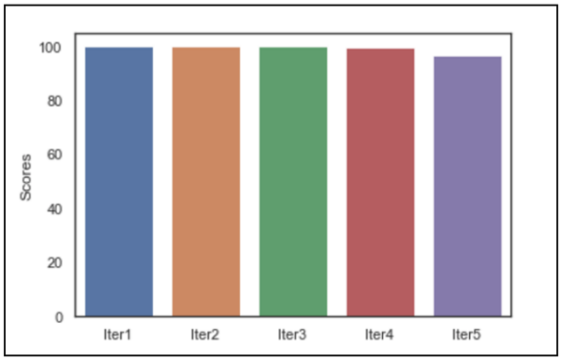

Fig. 7. Visual representation of the accuracy obtained on each iteration for the testing set

The model created using the 5-fold cross validation can be visually represented as the following tree:

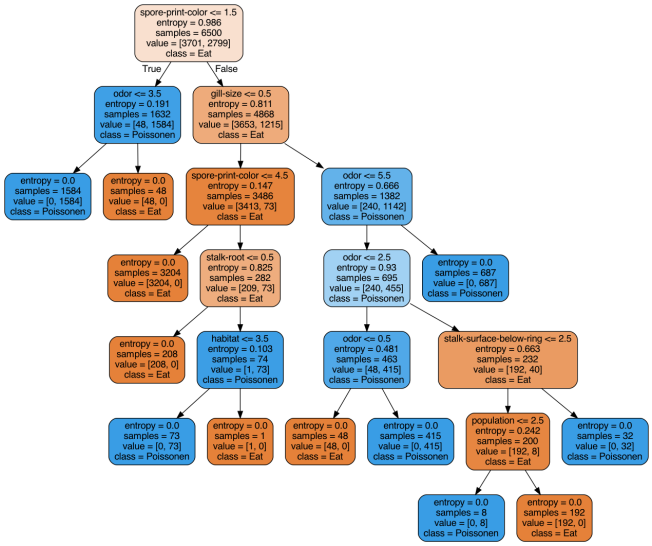

Fig. 8. Visual representation of the decision tree classification created with 5-fold cross validation

4.3 Comparing the Models

In order to compare the performance of both models, we evaluate the accuracy obtain when labeling the entire mushroom dataset. Both sets have a very high accuracy, but as we can notice the first method performs better by less than 1%.

```
Accuracy when predicting the entire dataset

First Model: 100.00%
Second Model: 99.90%
```

Fig. 9. Accuracy obtained by both models

5. CONCLUSION

In conclusion, the results for the Mushroom Database decision tree classifier models are successful. The high level of accuracy obtained on both approached represents that the models could successfully predict if a mushroom specie is poisonous or not. By comparing the models obtained by the different approaches, we can conclude that the original model did not struggle with overfitting. On the other hand, it is a better approach to perform cross validation, because the results are still very positives, and any bias can be removed. What both performances show, is that determining if a mushroom specie is poisonous or not, can be easily determined by its attributes, in particular by the score-print-color, the gill-size, and the odor.