Deadly mushrooms

Machine Learning methods for poisonous mushrooms identification

Aprenentatge Automàtic 1
Marc Vernet Sancho
Jordi Puig Rabat
UPC, 2020

Index

Introduction

The chosen dataset for this project contains mushroom records drawn from *The Audubon Society Field Guide to North American Mushrooms (1981)*. This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family, described with 22 categorical attributes about shape, color and other mushroom information plus one target variable determining the edibility of that mushroom.

The dataset documentation describes that originally the mushrooms where divided in three classes: definitely edible, definitely poisonous, or of unknown edibility and not recommended. Although there were 3 classes in the beginning, the available dataset in the website[1] the mushrooms of unknown edibility class were combined with the poisonous one.

Our objective is to experiment with different methods to design a classifier that separates between poisonous and edible mushrooms, and test its performance with proper techniques.

As we were explaining, the observations of the dataset were hypothetical, this means this are not real observations but how the observation of a mushroom belonging to a certain specie should be. There are 22 variables related to shape, color, odor and other physical attributes of the mushrooms plus the target variable indicating whether is poisonous or not.

In order to design a classifier we had to transform the data; moreover we have performed a little exploration in order to support the hypothesis that it is possible to classify the data without much error. This will be explained on the next part. Then, once we have the data ready to train models, we are going to explain what methods we have used and the results obtained with this methods.

[1] UCI Machine Learning Repository: Mushroom Data Set

Preprocessing

Before starting to use the data for predictions, it's necessary some preprocessing in order to assure the quality and the tractability of it. The download data is written in a dataframe, where with a first analisis we can know more about it. As a first step, we can check in how many classes each categorical variable is classified (Fig. 2). One outstanding characteristic is that the variable *veil.type* only has one class. This variable can be ignored as we want to use the data for predictions.

Something that can be checked on the data set is whether the classes of the variable to predict are balanced. The classes are balanced if there is a similar number of occurrences of edible and poisonous mushrooms. Of the total number of mushrooms, 52.8% are classified as edible and 48.2% as poisonous, thus, we consider the classes are clearly balanced.

If we check for missing data or NA, the only thing that we can find is the character "?" as a class for the *stalk.root* variable. There are 2840 occurrences out of a total of the 8124 instances. In the dataset documentation it's described as "missing stalk root", so it will be considered as another class of stalk root.

As a way to orientate further analysis, it could be useful to plot the histograms of each variable separated by the class we want to predict. Just looking it can be seen that some of the variables with more differences between edible and poisonous classes are: *gill.size, odor, gill.color, spore.print.color, habitat, population, ring.type*. Some of them are plotted to see the differences with more detail (*Fig. 1*).

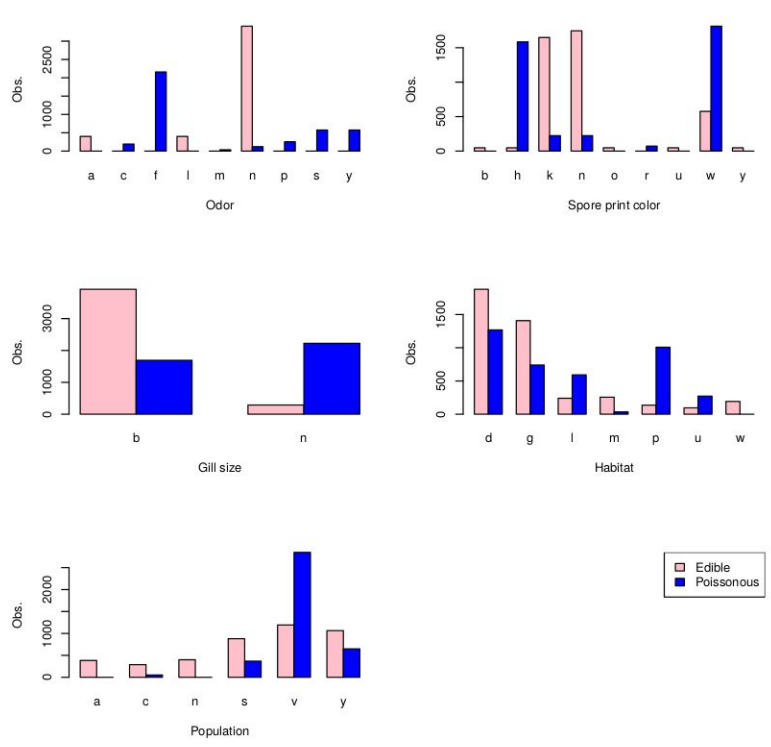

Figure 1: Histograms of different variables.

	Classes
class	2
cap.shape	6
cap.surface	4
cap.color	10
bruises	2
odor	9
gill.attachment	2
gill.spacing	2
gill.size	2
gill.color	12
stalk.shape	2
stalk.root	5
stalk.surface.above.ring	4
stalk.surface.below.ring	4
stalk.color.above.ring	9
stalk.color.below.ring	9
veil.type	1
veil.color	4
ring.number	3
ring.type	5
spore.print.color	9
population	6
habitat	7

Figure 2: The number of classes for each variable

Data exploration

In order to elaborate a plot of the data, we have considered using the Multidimensional Scaling tools we have learned in Data Analysis. One way to measure the distance between two observation is computing the number of different values for each variable. This gives us with a distance with range [0, 22] as there are 22 variables. We have computed the full distance matrix with a simple program written in *c++*[2]. We have decided to use this lenguage because the size of the resulting matrix is 8124x8124, which means the program must be fast; *c++* gives us more speed than R in this kind of simple calculations. The MDS object with 2 dimensions has been calculated in R (Fig. 3) with the built-in function *cmdscale*.

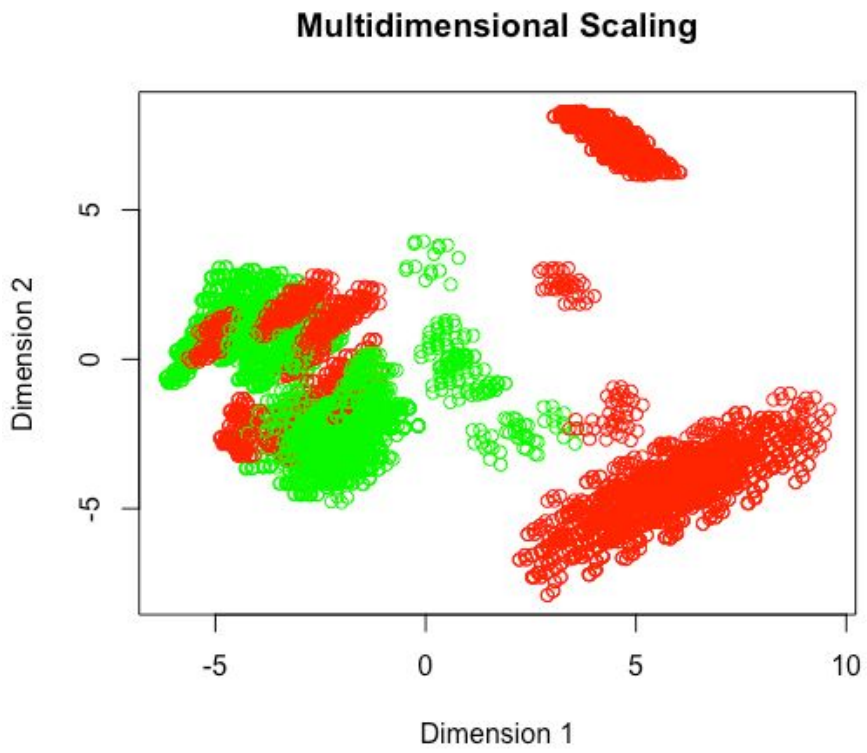

Figure 3: The MDS plot over 2 dimensions. Geen points represent edible mushrooms and red points are the poisonous ones. We can distinguish three main regions, two of them are clearly red and the third one is a mix of red and green. Here we presume the red points in the third cloud are the ones representing some of the mushrooms whose edibility was uncertain and were merged in the non-edible class.

Although we had good results with less observations, we have decided to plot all the observations in order to make sure nothing escapes to us. This result is worth the additional time spent in computing and programming.

[2] The code can be founded in the annex.

Dataset Encoding

The original data is encoded in categorical variables, it could be useful to have it encoded as numeric variables in order to have more methods to study the data. One way to do this is simply assign a different numeric value to every class of every variable. Although this is a simple procedure, it is erroneous because it could be considered that a class categorized as 3 has more importance than a class categorized as 1 in correlation or distance calculations. This method has been discarded and we have used the binary encoding method.

Binary Encoding:

The correct way of categorical to numeric dataset encoding is One-Hot encoding, where all data is binarized. A column for each class of each variable is created with 0 or 1 as values representing the class with a 1 and the other columns –the other classes of the same variable– with a 0. This encoding solves the important problematic that happens in the numerical encoding. The disadvantage is that the data frame is now much bigger (from having 23 columns to having 118). This procedure is done automatically by R when linear models are used with categorical variables. Moreover, it is possible to create a data frame with this data to feed other models proposed in the next section.

Prediction Models

Here we are going to discuss the models and methods considered to be useful to accomplish our main goal.

Generalized Linear Models

The first method of prediction investigated is linear models. Different GLM models have been proposed and tested using the preprocessed categorical dataset.
The original dataset has been divided in three subsets: train, test and validate with approximately the same size. The models will be trained with the **train** dataset and tested with the **validation** dataset to obtain fast approximated accuracy results in order to see which models are more promising. The models with best results will be tested using a 10-fold cross validation with the combined **train** and **validation** dataset. Finally, to obtain a final test accuracy, the best models will be tested using the **test** dataset

Some preliminary models that can be checked are:

- *mod.colors*: using only color related variables stored in ***colors***
- *mod.stalk:* using only stalk related variables stored in ***stalk***

Here we are using the *dataTrain* –the training data as it names indicates– to train the model and the validation accuracy is computed with the **validation** partition.

*mod.colors <- glm(dataTrain$class~. , data=dataTrain[,**colors**], family=binomial(link="logit"))*
Validation accuracy : **95.37%**

*mod.stalk <- glm(dataTrain$class~. , data=dataTrain[,**stalk**], family=binomial(link="logit"))*
Validation accuracy : **83.63%**

We can check color model in particular is pretty good.

We will consider a full model where all variables are used to predict the mushroom class. The accuracy is very high, reaching zero errors in prediction. Although this model is perfect it's also very complex, with lots of different variables and costly computational time.
Because some of the variables in the dataset have a large number of classes and the subsets used for training are small (⅓ of total), some classes where not represented in the train dataset and the model didn't work. As a solution, only for this model all the full dataset has been used(train and validation datasets used for training, test dataset for testing), so the results obtained in this case are only orientative.

mod.full <- glm(dataTrain$class~. , data=dataTrain, family=binomial(link="logit"))
Test accuracy : **100%**

Using the full model, with the R function *varImp* we can know which variables have more weight in the model and use them to create simpler models with similar results. The variables with more importance are *odor, gill.spacing, gill.size, gill.color, stalk.shape, bruises, cap.surface, cap.color.*

First we can try to create a model with the variables that have the most different distribution between the edible and poisonous class. This classes can be found in the preprocessing plots, they are: *gill.size, odor, gill.color, spore.print.color, habitat, population, ring.type*. Stored in ***diff*** we obtain these result

*mod.diff <- glm(dataTrain$class~. , data=dataTrain[,**diff**], family=binomial(link="logit"))*
Validation accuracy : **100%**

In this case the model it's also perfect, with 0 error. This model it's much better than the full model, it has the same performance but uses less than half of the variables.

If we come back to the important variables in the full model, we can try *mod.spc3* and *mod.spc7*, with the 3 and 7 variables respectively with most importance in the full model used for prediction. In order of importance : *odor, gill.spacing, gill.size, gill.color, stalk.shape, bruises, cap.color*.

mod.spc7 <- glm(dataTrain$class~. , data=dataTrain[,spc7], family=binomial(link="logit"))
Validation accuracy : **99.36%**
mod.spc3 <- glm(dataTrain$class~. , data=dataTrain[,spc3], family=binomial(link="logit"))
Validation accuracy : **98.8%**

Both models are really good, especially the 3 variable model. With a very simple model of only three variables we can have a nearly perfect prediction.

Predictions can also be made with a Naive Bayes classifier, that uses to work very well with multiple categorical variables.

NB = naiveBayes(dataTrain$class ~., data=dataTrain)
Validation accuracy : **94.4%**

We can see it is a good model but it will be discarded as the GLM with 7 or 3 variables (much simpler) seems to perform better.

10-fold Cross Validation
The models that had the most promising results were *mod.spc7* and *mod.diff* for having perfect and nearly perfect, but too much complexity (using 7 of 22 variables). On the other hand, we have *mod.spc3*, that seem to achieve a nearly perfect accuracy with small complexity. To validate this models we have performed a 10-fold cross validation.
The results in terms of accuracy:

Model	*mod.spc3*	*mod.spc7*	*mod.diff*
Accuracy	98.73%	99.4%	100%

It seems that the accuracy is validated enough. Between *mod.spc7* and *mod.diff* it's obvious that *mod.diff* it's a better choice, both have the same complexity and it has proved that it's capable to be a perfect classifier. And *mod.spc3* has good enough results for a model of its simplicity.
Finally, the definitive test accuracy calculated using the **test** dataset:

mod.spc3	*mod.diff*
98.116%	**100%**

Neural Networks

Another way of creating an efficient classificator is with Neural Networks. We will try different models of multi-layer perceptrons (MLP) Neural Networks. The configurable parameters of MLP are size and decay. Size is the number of hidden neurons in the hidden layer, and decay is a regularization parameter.
In this part, the original dataset has been divided in equal size train and test subsets.

As a first model to try, a neural network using all variables (like the full model) with 2 hidden neurons and 0 decay:

model.nnet <- nnet(class ~., data = data, subset=learn, size=2, maxit=200, decay=0)
Test Accuracy : **100%**

This model seems to give a perfect response. It gives the same output compared with the GLM. A simpler model with similar results would be better.

A GLM that had good results and was simpler was *mod.spc3*, modeled with the 3 variables with most importance in the full model GLM. Perhaps with a neural network better accuracy can be achieved. As first observation, a MLP with 2 hidden neurons and 0 decay is tried:

model.spc3 <- nnet(data$class~., data=data[,spc3], subset=learn, size=2, maxit=200, decay=0)
Test Accuracy : **99.08 %**

It seems promising. Different size and decay values can be tested to see how the model behaves. There are two strategies: consider a constant large size and find the best regularization parameter, or try to find the size that works best without regularization parameter.

Constant size=20 and variable decay	Constant decay=0 and variable size
Decay = 0 TestAccuracy = 99.11 % Decay = 0.1 TestAccuracy = 99.11 % Decay = 0.2 TestAccuracy = 99.11 % Decay = 0.3 TestAccuracy = 99.11 % Decay = 0.4 TestAccuracy = 99.11 % Decay = 0.5 TestAccuracy = 99.11 % Decay = 0.6 TestAccuracy = 99.11 % Decay = 0.7 TestAccuracy = 99.11 % Decay = 0.8 TestAccuracy = 99.11 % Decay = 0.9 TestAccuracy = 99.11 % Decay = 1 TestAccuracy = 99.11 % Decay = 1.1 TestAccuracy = 99.11 % Decay = 1.2 TestAccuracy = 99 % Decay = 1.3 TestAccuracy = 99 % Decay = 1.4 TestAccuracy = 99 % Decay = 1.5 TestAccuracy = 99 % Decay = 1.6 TestAccuracy = 99 % Decay = 1.7 TestAccuracy = 99 % Decay = 1.8 TestAccuracy = 99 % Decay = 1.9 TestAccuracy = 99 % Decay = 2 TestAccuracy = 99 %	Size = 3 TestAccuracy = 99.19 % Size = 5 TestAccuracy = 99.22 % Size = 7 TestAccuracy = 99.22 % Size = 9 TestAccuracy = 99.22 % Size = 11 TestAccuracy = 99.22 % Size = 13 TestAccuracy = 99.22 % Size = 15 TestAccuracy = 99.22 % Size = 17 TestAccuracy = 99.22 % Size = 19 TestAccuracy = 99.22 % Size = 21 TestAccuracy = 99.22 % Size = 23 TestAccuracy = 99.22 % Size = 25 TestAccuracy = 99.22 % Size = 27 TestAccuracy = 99.22 % Size = 29 TestAccuracy = 99.22 %

It seems that there is little variation between them. We select one of the models with higher accuracy (in this case will be considered size=4 and decay=0).
Applying a 10-fold cross validation, we get a **99.06%** of validated test accuracy. A little better than the GLM but adding a little complexity.

Conclusion

After all the designed and tested models, a main conclusion is that it's easy to create good classifiers for this dataset. Most of the models tried had very high accuracy, superior than 90%. One of the main challenges is that getting higher accuracies and trying to achieve perfect classificators means creating very complex models that use most of the variables and have larger computational costs.

It has been proven that complex models (full models for example) can give perfect accuracies consistently, but the challenge is to find simple models that can reach the same performance. Although some linear models with less variables than the full model seemed to achieve similar performance, the best model can be considered *mod.sp3*. This model, using only 3 of the 22 variables in the dataset (*odor, gill.spacing, gill.size*), reached a performance of 98.116% as GLM and 99.06% as MLP neural network.

The results are satisfactory enough considering the simplicity of the model. As a conclusion, we can say that it can be confidently determined whether a mushroom is poisonous or edible observing only the odor, the gill.spacing and the gill.size.

Annex

Description of all scripts used in the project:

- **preprocessing.Rmd** : preliminar study of the data and encoding
- **models.Rmd** : Definition of the Generalized Linear Models, it's validation and test accuracy computations
- **NN.Rmd** : Definition of the Neural Networks models, validation and test accuracy computation
- **fulldistance.cc** : the program written in *c++* to compute the distance matrix
- **mds.Rmd** : The computation of the metric multidimensional scaling using the distance matrix of all observations (long runtime)