Mushroom classification with logistic regression

Miguel Angel Navarro Mata

April 20, 2020

1 Abstract

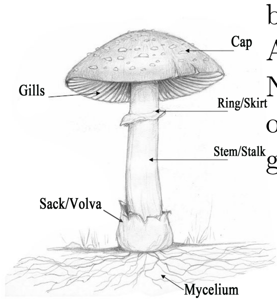
This report presents a logistic regression model to predict if a mushroom is edible or poisonous based on its physical characteristics. It is trained and tested with 8,124 instances that consist of 23 species of gilled mushrooms in the Agaricus and Lepiota Family.

2 Introduction

Mushrooms are a very particular organisms. Many people think that they resemble to the plants. However, they are more close to being bacteria or animals than plants. What people know as a mushroom is only the fruiting body of the mushroom. Fruiting bodies can vary a lot even between same species, therefore it's classification can be very difficult by plain sight. Logistic regression is a method used to model the probability of existing like pass/fail, win/loose and in this case edible/poisonous.

3 The data set

The data-set can be found in here.



It is a collection of records by The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf It contains the description of hypothetical samples according to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.

The data-set contains
a total of 8124 instances, 4208 edible and 3916
poisonous. Each sample has 22 categorical attributes.

3.1 Encoding the columns

Due to the fact that the columns are categories, the data needs to be encoded, for that I used One-Hot encoding. One-Hot encoding transforms a N category column into N boolean column. Saying that if we have a color category consisting of RGB values, it will transform it into 3 different columns, one for each color.
The data-set originally contains a total of 118 categories, so a discrimination of columns need to be done in order to make the model faster and also prevent overfitting.

3.2 Column Selection

The first step I took was to discriminate the columns that are not associated with poisonous species according to rules of what makes a mushroom poisonous established by the data-set author:

```
P_1) odor=NOT(almond.OR.anise.OR.none)
                120 poisonous cases missed, 98.52% accuracy

P_2) spore-print-color=green
     48 cases missed, 99.41% accuracy

P_3) odor=none.AND.stalk-surface-below-ring=scaly.AND.
          (stalk-color-above-ring=NOT.brown)
     8 cases missed, 99.90% accuracy

P_4) habitat=leaves.AND.cap-color=white
         100% accuracy
Rule P_4) may also be

P_4') population=clustered.AND.cap_color=white
```

This tell us that the recommended attributes for this data-set would be:

- Odor
- Spore-print color

- Stalk surface below ring
- Habitat
- Cap color
- Population

I began using only this attributes but later I found that the model was not getting the results I was expecting (as shown in the next section). So I added the following columns according to what I could find online:

- Gill color
- Stalk color above ring
- Ring number

This results in 70 parameters using one-hot encoding.

4 The model

4.1 Logistic regression hypothesis

As said before, logistic regression is a classification method that models the probability of an event happening. The way it predicts values, is using ***Sigmoid*** activation formula:

$$\frac{1}{1+e^{-z}} \tag{1}$$

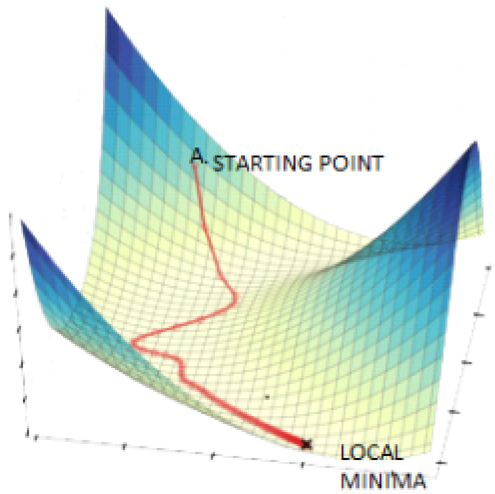
Being Z the sum of all it's attributes multiplied by it's corresponding weights:

$$z = \theta_0 X_o + \theta_1 X_1 + ... + \theta_n X_n \tag{2}$$

also seen as

$$h(x) = \theta_0 X_o + \theta_1 X_1 + ... + \theta_n X_n \tag{3}$$

4.2 Gradient descent



Gradient descent is the responsible to minimizing the error through each iteration. The way it works is as if it were a slope, calculating the way to the minimum. It updates each theta using this formula:

$$\theta_{new} = \theta_{old} - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h(x_i) - y_i) x_i) \tag{4}$$

The formula takes into consideration $\boldsymbol{\alpha}$ as the learning rate, $\boldsymbol{m}$ as our training set size.

4.3 Cross Validation

Cross validation is a procedure in which the total data is split into N sets, one for training and another to test. This, to know if the model is overfitting or if is actually learning. In this case, I divided the data into two sets:

- Training(7800 instances)
- Test(324 instances)

5 Results

As said on page ***iv***, I began testing my model just using the columns the article recommended to use and got the following results:

```
Epoch:  20  train error:  0.1307755443515443  test error:  0.1346993114816575 4
Epoch:  21  train error:  0.1277396494980174  test error:  0.1314093131735778
Epoch:  22  train error:  0.12489576589928401  test error:  0.1283413630783348
Epoch:  23  train error:  0.1222245357742196  test error:  0.1254728633190645
Epoch:  24  train error:  0.11970921197060631  test error:  0.12278429713148845
Epoch:  25  train error:  0.11733521577537827  test error:  0.12025869758723981
Epoch:  26  train error:  0.11508978515017003  test error:  0.11788122647861644
Epoch:  27  train error:  0.11296169183330891  test error:  0.11563883699303126
Epoch:  28  train error:  0.11094101156191141  test error:  0.11352000087464274
Epoch:  29  train error:  0.10901893575119762  test error:  0.11151448576395782
Epoch:  30  train error:  0.10718761587942223  test error:  0.1096131719812411
Epoch:  31  train error:  0.10544003393130663  test error:  0.10780790061161816
Epoch:  32  train error:  0.10376989379438395  test error:  0.10609134665140632
Epoch:  33  train error:  0.10217152964532875  test error:  0.10445691238583227
Epoch:  34  train error:  0.10063982822027794  test error:  0.10289863722567101
Epoch:  35  train error:  0.09917016251250449  test error:  0.1014111210306998
Epoch:  36  train error:  0.09775833493825523  test error:  0.0999894585593606
Epoch:  37  train error:  0.09640052839552588  test error:  0.09862918315531023
Epoch:  38  train error:  0.09509326394042682  test error:  0.09732621814780874
Epoch:  39  train error:  0.0938333640412429  test error:  0.0960768347297946
Epoch:  40  train error:  0.09261792055694895  test error:  0.09487761530390508
```

As we can see, there's a very small sign of overfitting. On the other hand with the updated parameters we can see an improvement in the results avoiding overfitting as we can see the comparison between the test and train error:

```
Epoch:  20  train error:  0.12495045437601174  test error:  0.10747002333164195
Epoch:  21  train error:  0.1220965165514642  test error:  0.10493642394324253
Epoch:  22  train error:  0.1194146188046332  test error:  0.10257988901206447
Epoch:  23  train error:  0.11688768836231309  test error:  0.10038144143087736
Epoch:  24  train error:  0.11450096621924202  test error:  0.09832477627555118
Epoch:  25  train error:  0.11224161358669939  test error:  0.09639579200038893
Epoch:  26  train error:  0.11009839855856372  test error:  0.09458221959346302
Epoch:  27  train error:  0.10806144403988918  test error:  0.09287332616357794
Epoch:  28  train error:  0.10612202303753074  test error:  0.09125967576275668
Epoch:  29  train error:  0.10427239097832551  test error:  0.08973293470740532
Epoch:  30  train error:  0.10250564727380769  test error:  0.08828571184711587
Epoch:  31  train error:  0.10081562020433811  test error:  0.08691142653683362
Epoch:  32  train error:  0.09919677055874375  test error:  0.08560419875934482
Epoch:  33  train error:  0.09764411048013165  test error:  0.08435875709951868
Epoch:  34  train error:  0.096153134731712  test error:  0.08317036121239502
Epoch:  35  train error:  0.09471976217692638  test error:  0.08203473613975223
Epoch:  36  train error:  0.09334028571341502  test error:  0.08094801637468106
Epoch:  37  train error:  0.09201132924538666  test error:  0.07990669799407149
Epoch:  38  train error:  0.09072981054843844  test error:  0.07890759750596187
Epoch:  39  train error:  0.08949290909278533  test error:  0.0779478163150935
Epoch:  40  train error:  0.08829803805925295  test error:  0.07702470991249562
```

Training my model for more than 40 epochs, gets us around a 2% of error for future predictions.


```
Epoch:  373  train error:  0.030262826625449064  test error:  0.024016846264475557
Epoch:  374  train error:  0.03023705187523963  test error:  0.02399663495173619
Epoch:  375  train error:  0.030211321739559625  test error:  0.02397646163194582
Epoch:  376  train error:  0.030185636099828262  test error:  0.023956326192649716
Epoch:  377  train error:  0.03015999483789243 5  test error:  0.02393622852184579
Epoch:  378  train error:  0.03013439783605451  test error:  0.02391616850798244
Epoch:  379  train error:  0.030108844977060643  test error:  0.023896146039956365
Epoch:  380  train error:  0.03008333614410134  test error:  0.02387616100711012
Epoch:  381  train error:  0.030057871220807334  test error:  0.023856213299230194
Epoch:  382  train error:  0.03003245009125042  test error:  0.02383630280654433 5
Epoch:  383  train error:  0.030007072639936957  test error:  0.023816429419719794
Epoch:  384  train error:  0.029994402977280177  test error:  0.023806508944855555
Epoch:  385  train error:  0.029981744184951965  test error:  0.023796597702237823
Epoch:  386  train error:  0.02996909624865177  test error:  0.023786695678275505
Epoch:  387  train error:  0.029956459154107006  test error:  0.023776802859405973
Epoch:  388  train error:  0.029943832887071396  test error:  0.02376691923209518
Epoch:  389  train error:  0.029931217433325986  test error:  0.023757044782837442
Epoch:  390  train error:  0.029918612778678737  test error:  0.023747179498155406
Epoch:  391  train error:  0.02990601890896269  test error:  0.023737323364599806
Epoch:  392  train error:  0.02989343581003909  test error:  0.023727476368749553
Epoch:  393  train error:  0.029880863467794155  test error:  0.02371763849721139
Epoch:  394  train error:  0.029868301868141305  test error:  0.02370780973662001
Epoch:  395  train error:  0.029855750997020404  test error:  0.02369799007363791
Epoch:  396  train error:  0.029843210840397077  test error:  0.02368817949495523
Epoch:  397  train error:  0.029830681384263327  test error:  0.023678377987289594
Epoch:  398  train error:  0.029818162614637694  test error:  0.023668585537386197
Epoch:  399  train error:  0.029805654517564017  test error:  0.023658802132017542
Epoch:  400  train error:  0.029793157079111995  test error:  0.023649027757983457
```

6 References

Brownlee, J., 2020. How To Implement Logistic Regression From Scratch In Python. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/> [Accessed 20 April 2020].

Daziel, C., 2018. How To Identify Poisonous Mushrooms. [online] Sciencing. Available at: <https://sciencing.com/identify-poisonous-mushrooms-2057768.html> [Accessed 20 April 2020].