Project3_fys-stk3155

December 15, 2020

1 Abstract

2 Introduction

3 Data

3.1 Mushrooms[1]

The mushroom dataset contains descriptive data for (hypothetical) samples of 23 species of gilled mushrooms in the Agaricus and Lepiota Family. The samples are drawn from *The Audubon Society Field Guide to North American Mushrooms* (1981)[2]. Each species is classified as either edible (class e) or poisonous (class p), where the poisonous category includes both species known to be poisonous as well as those where edibility is unknown.

The data set contains a total of 8124 samples, each described with 22 descriptors. To reduce the size of the dataset, each attribute value is coded to a letter. These attributes are as follows:

1. cap-shape:
   bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s

2. cap-surface:
   fibrous=f,grooves=g,scaly=y,smooth=s

3. cap-color:
   brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

4. bruises?:
   bruises=t,no=f

5. odor:
   almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

6. gill-attachment:
   attached=a,descending=d,free=f,notched=n

7. gill-spacing:
   close=c,crowded=w,distant=d

8. gill-size:
   broad=b,narrow=n

9. gill-color:
   black=k,brown=n,buff=b,chocolate=h,gray=g,green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y

10. stalk-shape:
    enlarging=e,tapering=t

11. stalk-root:
    bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?

12. stalk-surface-above-ring:
    fibrous=f,scaly=y,silky=k,smooth=s

13. stalk-surface-below-ring:
    fibrous=f,scaly=y,silky=k,smooth=s

14. stalk-color-above-ring:
    brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

15. stalk-color-below-ring:
    brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

16. veil-type:
    partial=p,universal=u

17. veil-color:
    brown=n,orange=o,white=w,yellow=y

18. ring-number:
    none=n,one=o,two=t

19. ring-type:
    cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z

20. spore-print-color:
    black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y

21. population:
    abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

22. habitat:
    grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

3.1.1 Known Simple Rules

As this data set has been studied extensively, several more or less complex rules have been found
for deciding whether a given mushroom is edible or not. Particularly a set of four markedly simple
rules have been found that together give a 100 % accuracy on classifying poisonous mushrooms
[1]:

- 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`
  0 cases missed, 100% accuracy

3.2 Fetal Health[3][4]

A dataset of 2126 entries, each described by 22 features. There are no missing or `Null` values. Table number shows values for descriptive statistics like min and max values, mean, and standard deviation. The target value is `fetal_health` which can take one of three classes:

- Normal

- Suspect

- Pathological

as determined by three expert obstetricians.

We see from figure number that, not unexpectedly, most entries are in the *Normal* class. As such the dataset is quite unbalanced.

4 Methods

4.1 Decision Trees

A decision tree is a type of supervised learning model that can be used for both regression and classification problems. They are named *trees* as their structure consists of a root node recursively split into nodes, or "branches", ending in the end-nodes also known as "leaves". Each split of a node is based on a choice or decision for one of the features of the data, like for instance "is height>=2.0m?".

When using a decision tree for prediction we move down the tree with our input data, for each node determining whether to move left or right down the tree based on the value of the input data for the relevant "decision feature" at that node. Is the value below or above some threshold, or equal/not equal to some value? When we reach the end of the tree, one of the leaf nodes, this node tells us the resulting prediction.

Decision trees are popular models for real life problems as they produce easily interpretable models that resemble human decision making. They do not require normalization of the inputs, and they can be used to model non-linear relationships.

They are, however, prone to over-fitting and generally do not provide the best predictive accuracy. Other challenges for decision trees are that small changes in the data may lead to a completely different tree structure, and unbalanced datasets with a target feature value that occur much more often than others may lead to biased trees since the frequently occurring feature values are preferred over the less frequently occurring ones. In addition, features with many levels may be preferred over features with fewer levels as it is then easier to split the dataset such that the splits only contain pure target feature values.

Many of these issues can be improved upon by using ensemble methods, methods that aggregate several decision trees. This generally comes at the cost of interpretability.

Available algorithms for building a decision tree include ID3, C4.5 and CART. These algorithms typically use different criteria for how to perform splitting, ID3 uses information gain, C4.5 uses gain ratio, while CART uses the gini index.

4.1.1 CART Algorithm

Originally the term Classification And Regression Tree (CART) was introduced by Breiman et al.[9] as an umbrella term used for analysis of regression as well as classification trees. The CART algorithm is the most commonly used algorithm for building decision trees. It is a non-parametric learning technique.

With the CART algorithm trees are constructed using a top-down approach. We start by looking at all the available training data, and selecting the split that minimizes the cost function. This is then the root node. Split is performed in the same way moving down the tree until a stopping criteria is met.

To decide on the best split, a measure of impurity, G, is used. For CART this is typically the Gini index, while other options include the information entropy, or the miss-classification rate, see the following sections.

At each node we split the dataset into two subsets a and b using a single feature k and a threshold t_k, by finding the pair (k, t_k) giving the lowest impurity for the subsets according to the chosen impurity measure. This minimizes our cost function for this problem,

$$C(k, t_k) = \frac{m_a}{m} G_a + \frac{m_b}{m} G_b,$$

where $G_{a/b}$ measures the impurity of each of the subsets, and $m_{a/b}$ is the number of instances in subset a and subset b, respectively.

There are several possible stopping criteria, like maximum depth of tree, all members of the node belonging to the same class, the impurity factor decreasing by less than some threshold for further splits, or that the minimum number of node members is reached.

Building the Tree

When building the tree we start with the root node and move recursively down the tree as indicated in the following pseudocode: [6]

```
def find_split(input_data, target):
    start_impurity = find_impurity(data, target)
    split_threshold, split_feature, split_impurity
    for each feature in input_data:
        for each unique_value in feature:
            threshold = value
            impurity = find_impurity(feature, threshold)
            if impurity is better than split_impurity:
                split_threshold = threshold
                split_impurity = impurity
                split_feature = feature
    split_node(split_feature, split_threshold, input_data, target)
```

4

4.1.2 Gini Index

The Gini index is also called the Gini impurity, and it measures the probability of a particular variable that is randomly chosen being wrongly classified. As such it takes values between 0 and 1. Another way to look at it is that it measures the lack of 'purity' of the variables. A node is pure if all its variables or members belong to one class. The gini index then takes the value 0. The more of the members of the node that belong to a different class, the more impure the node is.

Denoting the fraction of observations (or members) of node/region m being classified to a particular class k as p_{mk}, the Gini index, g can be defined as

$$g = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2.$$

The fraction p_{mk} can be calculated as

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k).$$

When building a decision tree using CART with the Gini index as impurity measure we choose the attribute/feature with the smallest Gini index as the root node.

4.1.3 Entropy

Entropy is another measure for impurity. It is known from thermodynamics as a measure of disorder. In the classification case the entropy, or information entropy, is a measure for how much information we gain by knowing the value (or classification) of more features.

The entropy, s, can be defined in terms of the fraction p_{mk} defined in the section above, as

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk}.$$

4.2 Ensemble Methods

Ensemble methods use a set, or ensemble, of so-called weak learners, and use their combined predictive power to make predictions. While each individual model in the ensemble may have a poor performance, say only just above random guessing, the resulting ensemble may perform very well. Ensemble methods can use any (weak) learner as its base learner, and even a combination of different ones. We will, however look exclusively at ensembles of decision trees.

An individual decision tree is prone to over-fitting and high variance. The idea is that when averaging over many of these weak learners the variance of each tree averages out, reducing the total variance and thereby error of the resulting model. We will be exploring three kinds of ensemble methods where decision trees are the base learner; bagging, boosting, and random forests.

4.2.1 Bagging

Bagging is a simple form of an ensemble method. A set of N trees is built from the input data, with the twist that each tree is built only on a subset of the total input data. The subset is chosen by randomly sampling of the data, with substitution. In that way each tree is built on a bootstrap sample of the original training data. This can effectively reduce the variance of the model. This improved performance comes at the expense of the interpretability of the model.

```
Bagging Algorithm:
With training data X and y, and N trees.
for i from 1 to N:
    1. set X_ and y_ equal to a subset of X and Y, drawn from X and y with
       replacement
    2. fit tree i to X_ and y_
    3. store tree i for future prediction
```

4.2.2 Random Forests

Random forests take bagging one step further by adding randomness in what features are available when fitting each tree to the data. In essence, in addition to fitting to a random sample of the input or training data, the fit is done using a random subset of the available features.

Often every tree will be dominated by one or more strong or defining features, with every tree having the same root node. By only looking at a random subset of the features per tree we increase the randomness in the resulting tree ensemble, and hope to further reduce the variance. We are essentially reducing the correlation between the individual trees, and as such expect an improved reduction in variance over bagging where the trees will often remain very correlated.

One typically looks at $m \approx \sqrt{p}$ predictors for each tree, with p being the total number of predictors in the dataset.

```
Random Forest Algorithm:
With training data X and y, with X made up of F features, and
N trees.
for i from 1 to N:
    1. set X_ and y_ equal to a subset of X and Y, drawn with
       replacement
    2. draw f features randomly from the F features
    3. fit tree i to X_[f] and y_
    4. store tree for future prediction
```

4.2.3 Adaptive Boosting

While the ensemble methods described until now can easily be performed in parallel, adaptive boosting, or AdaBoost, uses the resulting prediction from each subsequent fit to improve the fit in the next. After each tree is fit to the data the algorithm makes note of what data is miss-classified in the resulting model, and gives these data increased weight in further model fitting. Optional features include performing a similar weighing of the features used, where features leading to good predictions are given more weight, or discarding individual models if their accuracy is below some level.

```
AdaBoost Algorithm:
With training data X and y, and N trees.
First assign equal weight to each observation
weights = np.ones()*1./X.shape[0]
for i from 1 to N:
    1. fit tree i to the data, weighing the data according to weights
        2. calculate error by summing up the weight of misclassified
           observations
      error = sum(weights of misclassified observation)/sum(weights)
       3. update weights using the quantity
          alpha=log(1 - error)/error:
          weight_i = weight_i*exp(alpha) if incorrectly classified
          weight_i = weight_i*exp(-alpha) if correctly classified
```

4.3   Data Processing

Our datasets in this projects are fairly clean and well-defined already, and do not require much pre-processing. As we are working with decision trees we do not need to normalize our input data, it will not affect the results. As we are using datasets with fairly high dimensionality, we explore some methods of feature selection to see if we can reduce dimensionality without reducing model performance

Variance Threshold

Feature selection by variance threshold is a simple form of feature selection where only features with variance above some threshold are kept. The logic behind this method is that the variance of a features signifies its spread. A feature that is more spread out across the entries will make it easier to separate entries based on this feature. In contrast, imagine if a feature takes only one value. This feature will then provide no information we can use to separate the entries. The strictness of this method is determined by the variance threshold used.

Univariate Feature Selection [8][12]

In univariate feature selection features are selected based on their value according to some set scoring function. While the variance threshold can be used for any problem, we must in univariate feature selection select a scoring function suitable for our needs. We are only looking at classification problems, and our mushroom dataset consists solely of categorical input features. This limits the suitable scoring functions.

We have used the χ^2 test statistic, which measures the dependence between stochastic variables, and can be used for categorical features. The χ^2 test statistic is a statistical hypothesis test and uses the assumption that the observed frequencies for each categorical variable match its expected frequencies. With this scoring function the features that are the most likely to be independent of the class and therefore irrelevant for classification are scored low, and can be removed.

Our fetal health dataset on the other hand has continuous features, meaning the χ^2 test statistic is not suitable. In place of it we have used the ANOVA F-value. The ANOVA (Analysis of variance) f-value uses the f test statistic to evaluate each individual feature's ability to distinguish the classes of the variables.

For both datasets we also use the mutual information. The mutual information measures the dependency between two random variables, with lower variables indicating more independent variables.

4.4 Performance Measures

4.4.1 Accuracy

See Methods section in project 2[13].

4.4.2 Confusion Matrix

A confusion matrix is a useful tool for visualizing the predictive accuracy of a model. Unlike the measure above, the confusion matrix illustrates the predictive accuracy per class. This way we can see what classes the model performs well of less well for, and information like what class misclassifications en up in instead.

The matrix consists of one row and one column per class. The rows represent the predicted class, while the columns represent the actual or true class (or vice versa). Where the row and column for a class meet we have the number of correctly classified instances. Where rows and columns of different classes meet we have a wrongly classified instance, and we can see trends in misclassifications. See for instance figure number for an example.

4.4.3 ROC Curve

A receiver operating characteristic curve, or ROC curve, is another tool for visualizing a model's predictive performance, more specifically a binary model. It consists of a graphical plot of false positive rate vs true positive rate for a set of discriminative thresholds. The area under the curve is one measure of model performance, with a perfect model getting a score of one.

5 Results and Discussion

5.1 Pre-processing the Data

```
[2]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import sys
     sys.path.append('../Code')
     sys.path.append('../Data')
     import DecisionTree as dt

     df = pd.read_csv('../Data/Mushroom/UCI MLR/agaricus-lepiota.data.csv',
                      names=['class','cap-shape','cap-surface','cap-color',
                      'bruises','odor','gill-attachment','gill-spacing',
                      'gill-size','gill-color','stalk-shape','stalk-root',
                      'stalk-surface-above-ring','stalk-surface-below-ring',
                      'stalk-color-above-ring','stalk-color-below-ring',
                      'veil-type','veil-color','ring-number','ring-type',
```

```
                    'spore-print-color','population','habitat'])

df.head()
```

[2]:   class cap-shape cap-surface cap-color bruises odor gill-attachment \
     0   p   x   s   n   t   p   f
     1   e   x   s   y   t   a   f
     2   e   b   s   w   t   l   f
     3   p   x   y   w   t   p   f
     4   e   x   s   g   f   n   f

        gill-spacing gill-size gill-color ... stalk-surface-below-ring \
     0   c   n   k ...   s
     1   c   b   k ...   s
     2   c   b   n ...   s
     3   c   n   n ...   s
     4   w   b   k ...   s

        stalk-color-above-ring stalk-color-below-ring veil-type veil-color \
     0   w   w   w   p   w
     1   w   w   w   p   w
     2   w   w   w   p   w
     3   w   w   w   p   w
     4   w   w   w   p   w

        ring-number ring-type spore-print-color population habitat
     0   o   p   k   s   u
     1   o   p   n   n   g
     2   o   p   n   n   m
     3   o   p   k   s   u
     4   o   e   n   a   g

     [5 rows x 23 columns]
```

As we know, all the features of this dataset are categorical. We can first explore the data by looking at the unique values for each feature, see table number.

[89]: ```python
print("Table 1: Unique values of the features in the mushroom data set.")
pd.concat({'unique values': df.apply(pd.unique), '# unique values': df.
 →nunique()}, axis=1)
```

Table 1: Unique values of the features in the mushroom data set.

[89]:   unique values \
     class   [p, e]
     cap-shape   [x, b, s, f, k, c]
     cap-surface   [s, y, f, g]

```
cap-color                               [n, y, w, g, e, p, b, u, c, r]
bruises                                                        [t, f]
odor                                 [p, a, l, n, f, c, y, s, m]
gill-attachment                                               [f, a]
gill-spacing                                                  [c, w]
gill-size                                                     [n, b]
gill-color                       [k, n, g, p, w, h, u, e, b, r, y, o]
stalk-shape                                                   [e, t]
stalk-root                                           [e, c, b, r, ?]
stalk-surface-above-ring                                [s, f, k, y]
stalk-surface-below-ring                                [s, f, y, k]
stalk-color-above-ring           [w, g, p, n, b, e, o, c, y]
stalk-color-below-ring           [w, p, g, b, n, e, y, o, c]
veil-type                                                        [p]
veil-color                                           [w, n, o, y]
ring-number                                             [o, t, n]
ring-type                                        [p, e, l, f, n]
spore-print-color                [k, n, u, h, w, r, o, y, b]
population                              [s, n, a, v, y, c]
habitat                             [u, g, m, d, p, w, l]
```

```
                         # unique values
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
```

As we can see, there is only one used value for veil-type, 'p' or partial. This feature then provides

us with no information that we can use to distinguish the different mushrooms, and we remove the feature completely. We also check to make sure none of the entries are missing values.

```
[90]: df.drop(columns=['veil-type'], axis=1, inplace=True)
      print("Table 2: The number of missing values for each features. We see that no␣
       ↪values are missing.")
      df.isnull().sum()
```

Table 2: The number of missing values for each features. We see that no values are missing.

```
[90]: class                     0
      cap-shape                 0
      cap-surface               0
      cap-color                 0
      bruises                   0
      odor                      0
      gill-attachment           0
      gill-spacing              0
      gill-size                 0
      gill-color                0
      stalk-shape               0
      stalk-root                0
      stalk-surface-above-ring  0
      stalk-surface-below-ring  0
      stalk-color-above-ring    0
      stalk-color-below-ring    0
      veil-color                0
      ring-number               0
      ring-type                 0
      spore-print-color         0
      population                0
      habitat                   0
      dtype: int64
```

Another point of interest is whether the data are fairly evenly divided amongst the two categories. The check this by comparing the number of entries belonging to each class.

```
[91]: # Creating histogram
      import seaborn as sns
      s = sns.countplot(x = "class", data = df)
      # Show plot
      plt.show()
      print("Figure 2: Histogram of class distribution. We can see the dataset␣
       ↪contains a fairly even split between the two\
      \nclasses.")
```
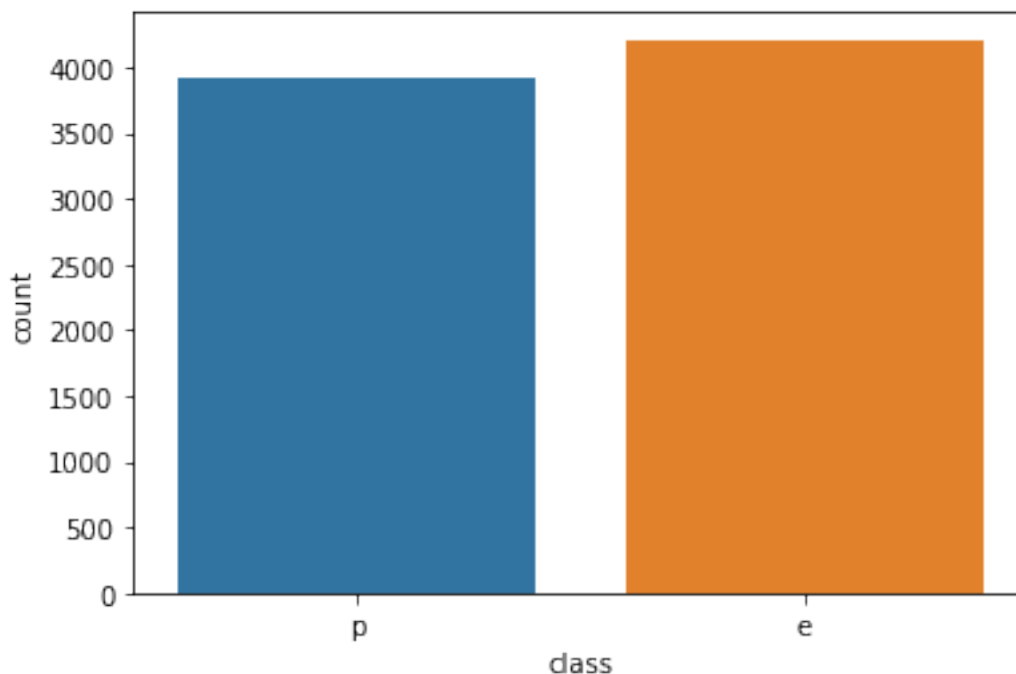
Figure 2: Histogram of class distribution. We can see the dataset contains a
fairly even split between the two
classes.

Figure number shows a heatmat of the correlations between the features.

```
[92]: # Now we have a look at the correlation matrix
      df_numerical = df
      # First we need to map to number-values
      from sklearn.preprocessing import LabelEncoder
      les = {col: LabelEncoder() for col in df_numerical.columns}
      for col in les:
          df_numerical[col] = les[col].fit_transform(df_numerical[col])

      df_numerical.head()
```

```
[92]:    class  cap-shape  cap-surface  cap-color  bruises  odor  gill-attachment  \
      0      1          5            2          4        1     6                1
      1      0          5            2          9        1     0                1
      2      0          0            2          8        1     3                1
      3      1          5            3          8        1     6                1
      4      0          5            2          3        0     5                1

         gill-spacing  gill-size  gill-color  ...  stalk-surface-above-ring  \
      0             0          1           4  ...                         2
```

12

```
1                0                0                4 ...                           2
2                0                0                5 ...                           2
3                0                1                5 ...                           2
4                1                0                4 ...                           2

    stalk-surface-below-ring  stalk-color-above-ring  stalk-color-below-ring  \
0                          2                       7                       7
1                          2                       7                       7
2                          2                       7                       7
3                          2                       7                       7
4                          2                       7                       7

    veil-color  ring-number  ring-type  spore-print-color  population  habitat
0            2            1          4                  2           3        5
1            2            1          4                  3           2        1
2            2            1          4                  3           2        3
3            2            1          4                  2           3        5
4            2            1          0                  3           0        1

[5 rows x 22 columns]
```
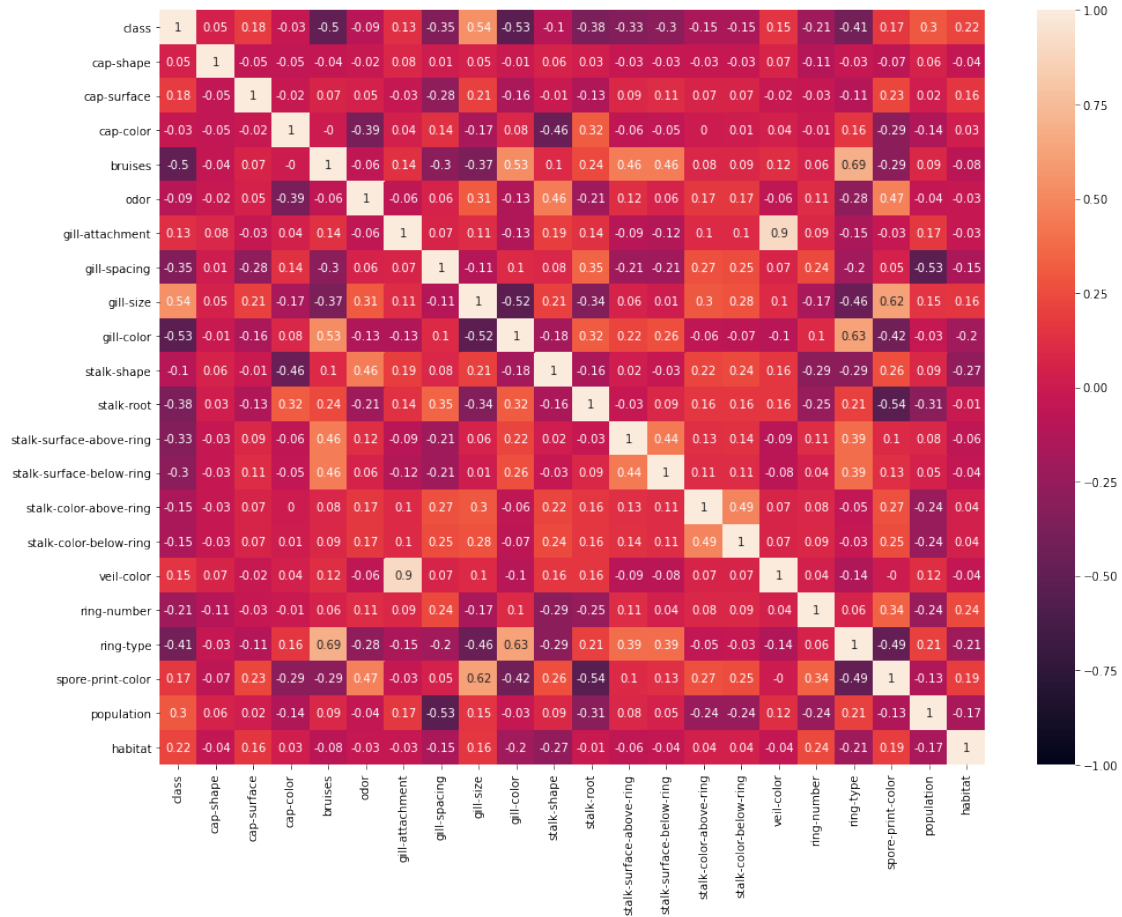
[93]:
```python
correlation_matrix = df_numerical.corr().round(2)
fig, ax = plt.subplots(figsize =(16, 12))
sns.heatmap(data=correlation_matrix, annot=True, vmin=-1, vmax=1)
print("Figure number: A heatmap showing the correlation matrix for the features
 ↪in the mushroom dataset.")
```

Figure number: A heatmap showing the correlation matrix for the features in the
mushroom dataset.
```

We see that `class` is most highly correlated with `gill-size`, `gill-color`, and `bruises`, followed by `ring-type`, `stalk-root`, and `gill-spacing`. The most highly corrolated features (meaning they provide much of the same information) are `gill-attachment` and `ring-number`, with a correlation of 0.9. Other fairly correlated features are `bruises` and `ring-type`, `gill-color` and `ring-type`, and `gill-size` and `spore-print-color`.

## 5.2 Feature Selection

As we have a fairly high-dimensional problem we performed feature selection in order to reduce the number of features. The goal for this was both to increase speed of fitting a model to the data, and to hopefully end up with a simpler model that is easier to interpret, understand, and use.

We have tried three methods for this; a simple variance threshold where features with a variance below a certain threshold or cutoff value are excluded, as well as univariate feature selection using two different scoring functions.

We use a cutoff value of 0.8 as variance threshold, which results in excluding four of the features, namely `gill-attachment`, `gill-spacing`, `veil-color`, and `ring-number`. The number of features is then reduced from 21 to 17. A plot of the variances for the different features is shown in figure number.

```
[113]:  # train - validaton - test - splitting 60 - 20 - 20
        from sklearn.model_selection import train_test_split

        seed = 68
        train_frac = 0.6   # out of the whole set
        test_frac = 0.5    # out of the non-training set

        y = df_numerical['class']
        X = df_numerical.drop(['class'], axis=1)


        X_train, X_test, y_train, y_test = train_test_split(X, y,␣
         ↪test_size=1-train_frac, random_state=seed)
        X_val, X_test, y_val, y_test =␣
         ↪train_test_split(X_test,y_test,test_size=test_frac,random_state=seed*3)

        #train, validate, test = np.split(df.sample(frac=1, random_state=42), [int(.
         ↪6*len(df)), int(.8*len(df))])
```

[113]: (1625, 21)

We can compare this to feature selection by comparing the value of certain scoring functions for each of the features. The result using chi-squared statistic, can be seen in figure number. We find from this result that comparing the features using this statistic the key features to include are features 8, 17, 7, 3, 10, 20, and 6, corresponding to `gill-color`, `ring-type`, `gill-size`, `bruises`, `stalk-root`, `habitat`, and `gill-spacing`.

Using instead the mutual information as scoring function we find that more features are included, Most notably feature four, `odor`, has gone from not being included to being the main feature. A bar plot is shown in figure number. This result correlates more closely with our expectations as we know from the known rules described in the Data section that using only `odor` is enough to get a 98.52% accuracy on this dataset.

Figure number: Bar plot showing the chi-squared scores of the the features in the mushroom dataset. We see that feature eight has the clear highest score. This corresponds to `gill-color`.

Figure number: Bar plot showing the mutual information scores of the the features in the mushroom dataset. We see that feature four, `odor` scores highest.

5.3   Fetal Health

```
Table number: A table showing key statistical values for the features in the
fetal health dataset.
```

```
                                    count       mean  \
baseline value                     2126.0  133.303857
accelerations                      2126.0    0.003178
fetal_movement                     2126.0    0.009481
```

15

```
uterine_contractions                               2126.0     0.004366
light_decelerations                                2126.0     0.001889
severe_decelerations                               2126.0     0.000003
prolongued_decelerations                           2126.0     0.000159
abnormal_short_term_variability                    2126.0    46.990122
mean_value_of_short_term_variability               2126.0     1.332785
percentage_of_time_with_abnormal_long_term_vari... 2126.0     9.846660
mean_value_of_long_term_variability                2126.0     8.187629
histogram_width                                    2126.0    70.445908
histogram_min                                      2126.0    93.579492
histogram_max                                      2126.0   164.025400
histogram_number_of_peaks                          2126.0     4.068203
histogram_number_of_zeroes                         2126.0     0.323612
histogram_mode                                     2126.0   137.452023
histogram_mean                                     2126.0   134.610536
histogram_median                                   2126.0   138.090310
histogram_variance                                 2126.0    18.808090
histogram_tendency                                 2126.0     0.320320
fetal_health                                       2126.0     1.304327

                                                         std     min      25%  \
baseline value                                      9.840844   106.0   126.000
accelerations                                       0.003866     0.0     0.000
fetal_movement                                      0.046666     0.0     0.000
uterine_contractions                                0.002946     0.0     0.002
light_decelerations                                 0.002960     0.0     0.000
severe_decelerations                                0.000057     0.0     0.000
prolongued_decelerations                            0.000590     0.0     0.000
abnormal_short_term_variability                    17.192814    12.0    32.000
mean_value_of_short_term_variability                0.883241     0.2     0.700
percentage_of_time_with_abnormal_long_term_vari... 18.396880     0.0     0.000
mean_value_of_long_term_variability                 5.628247     0.0     4.600
histogram_width                                    38.955693     3.0    37.000
histogram_min                                      29.560212    50.0    67.000
histogram_max                                      17.944183   122.0   152.000
histogram_number_of_peaks                           2.949386     0.0     2.000
histogram_number_of_zeroes                          0.706059     0.0     0.000
histogram_mode                                     16.381289    60.0   129.000
histogram_mean                                     15.593596    73.0   125.000
histogram_median                                   14.466589    77.0   129.000
histogram_variance                                 28.977636     0.0     2.000
histogram_tendency                                  0.610829    -1.0     0.000
fetal_health                                        0.614377     1.0     1.000

                                                       50%      75%      max
baseline value                                     133.000  140.000  160.000
accelerations                                        0.002    0.006    0.019
```

fetal_movement	0.000	0.003	0.481
uterine_contractions	0.004	0.007	0.015
light_decelerations	0.000	0.003	0.015
severe_decelerations	0.000	0.000	0.001
prolongued_decelerations	0.000	0.000	0.005
abnormal_short_term_variability	49.000	61.000	87.000
mean_value_of_short_term_variability	1.200	1.700	7.000
percentage_of_time_with_abnormal_long_term_vari...	0.000	11.000	91.000
mean_value_of_long_term_variability	7.400	10.800	50.700
histogram_width	67.500	100.000	180.000
histogram_min	93.000	120.000	159.000
histogram_max	162.000	174.000	238.000
histogram_number_of_peaks	3.000	6.000	18.000
histogram_number_of_zeroes	0.000	0.000	10.000
histogram_mode	139.000	148.000	187.000
histogram_mean	136.000	145.000	182.000
histogram_median	139.000	148.000	186.000
histogram_variance	7.000	24.000	269.000
histogram_tendency	0.000	1.000	1.000
fetal_health	1.000	1.000	3.000

Fetal Health Classes

Normal=1655

Cases

Pathological=176

Suspect=295

Fetal Health

Figure number: A pie plot showing how the target variable fetal health is distributed among the three possible classes.

Figure number: A heatmap showing the correlation matrix for the features in the fetal health dataset.

Table number: Focusing on the correlations between the features and the target, fetal health. We see that `prolongued_decelerations`, `abnormal_short_term_variability`, and `percentage_of_time_with_abnormal_long_term_variability` are most closely correlated with the resulting fetal health classification. Note that the absolute values of the correlations are shown, in descending order.

	fetal_health
fetal_health	1.000000
prolongued_decelerations	0.480000
abnormal_short_term_variability	0.470000
percentage_of_time_with_abnormal_long_term_variability	0.430000
accelerations	0.360000
histogram_mode	0.250000
mean_value_of_long_term_variability	0.230000
histogram_mean	0.230000
histogram_median	0.210000
histogram_variance	0.210000
uterine_contractions	0.200000
baseline value	0.150000
histogram_tendency	0.130000
severe_decelerations	0.130000
mean_value_of_short_term_variability	0.100000
fetal_movement	0.090000
histogram_width	0.070000
histogram_min	0.060000
light_decelerations	0.060000
histogram_max	0.050000
histogram_number_of_peaks	0.020000
histogram_number_of_zeroes	0.020000

Figure number shows the correlation matrix for this dataset, while table number makes the correlation of the features to the target more clear. We see that `prolongued_decelerations`, `abnormal_short_term_variability`, and `percentage_of_time_with_abnormal_long_term_variability` are the features that correlate most strongly witht he resulting classification. Between the features we see that there is naturally a very strong correlation between `histogram_mode`, `histogram_mean`, and `histogram_median`. as well as between `histogram_width`, `histogram_min`, and `histogram_max`. Particularly for the

mode, mean and median we see that they correlate in a very similar manner to all the other features as well and as such contribute with mush the same information, which means we do not gain much, if any, more insight by including all three in our model.


```
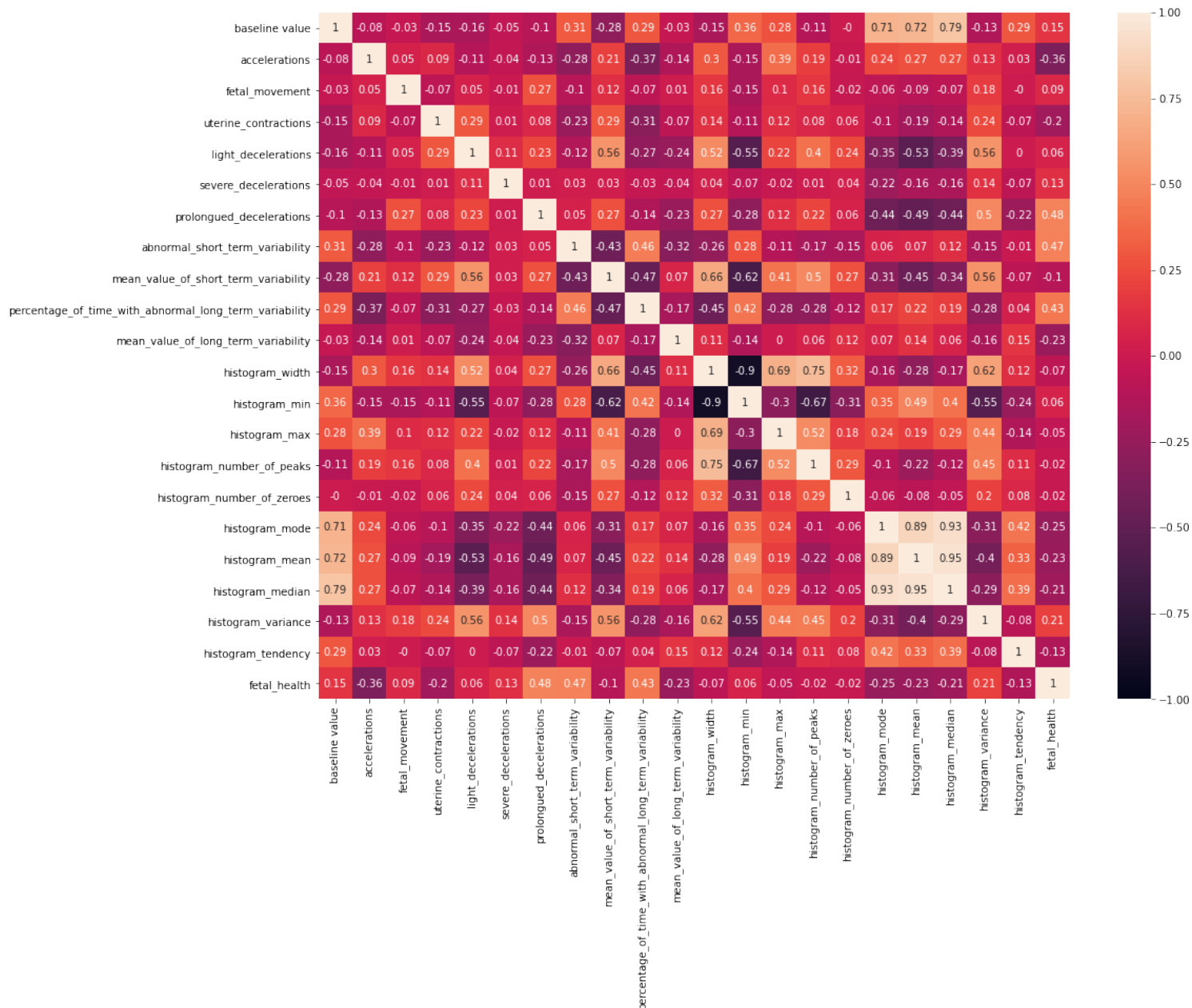* Created a decision tree with 6 leaves, and a depth of 5 at the deepest.*

Testing own decision tree code on fetal_health dataset:
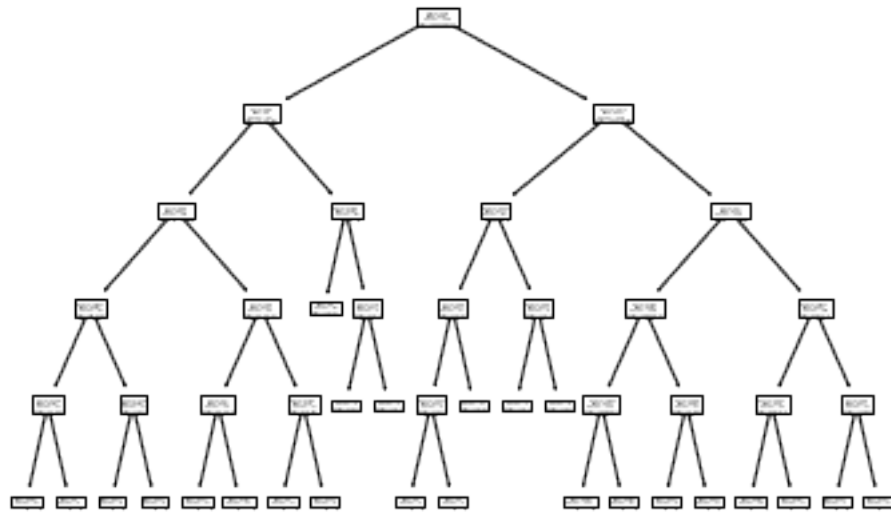Train accuracy:  0.8294117647058824
Test accuracy:  0.8352941176470589

The tree using my own code:
|--- histogram_mean <= 89.00
|   |--- weights:  class: 3.0,    prediction: ['0.0', '0.0', '1.0']
|--- histogram_mean >  89.00
|   |--- mean_value_of_long_term_variability <= 73.00
|   |   |--- mean_value_of_short_term_variability <= 81.00
|   |   |   |--- histogram_median <= 87.00
|   |   |   |   |--- weights:  class: 3.0,    prediction: ['0.0', '0.0', '1.0']
|   |   |   |--- histogram_median >  87.00
|   |   |   |   |--- abnormal_short_term_variability <= 0.00
|   |   |   |   |   |--- weights:  class: 1.0,    prediction: ['0.8', '0.1',
'0.0']
|   |   |   |   |--- abnormal_short_term_variability >  0.00
|   |   |   |   |   |--- weights:  class: 3.0,    prediction: ['0.0', '0.0',
'1.0']
|   |   |--- mean_value_of_short_term_variability >  81.00
|   |   |   |--- weights:  class: 3.0,    prediction: ['0.0', '0.0', '1.0']
|   |--- mean_value_of_long_term_variability >  73.00
|   |   |--- weights:  class: 3.0,    prediction: ['0.0', '0.0', '1.0']



Testing scikit-learn's DecisionTreeClassifier on fetal_health dataset:
Train accuracy:  0.941764705882353
Test accuracy:  0.9458823529411765
```

```
The tree using scikitlearn's DecisionTreeClassifier:
|--- percentage_of_time_with_abnormal_long_term_variability <= 0.55
|    |--- mean_value_of_long_term_variability <= 68.50
|    |    |--- mean_value_of_short_term_variability <= 59.50
|    |    |    |--- histogram_max <= 138.50
|    |    |    |    |--- uterine_contractions <= 0.02
|    |    |    |    |    |--- class: 1.0
|    |    |    |    |--- uterine_contractions >  0.02
|    |    |    |    |    |--- class: 2.0
|    |    |    |--- histogram_max >  138.50
|    |    |    |    |--- histogram_min <= 17.50
|    |    |    |    |    |--- class: 1.0
|    |    |    |    |--- histogram_min >  17.50
|    |    |    |    |    |--- class: 2.0
|    |    |--- mean_value_of_short_term_variability >  59.50
|    |    |    |--- mean_value_of_short_term_variability <= 79.50
|    |    |    |    |--- mean_value_of_long_term_variability <= 6.50
|    |    |    |    |    |--- class: 1.0
|    |    |    |    |--- mean_value_of_long_term_variability >  6.50
|    |    |    |    |    |--- class: 2.0
|    |    |    |--- mean_value_of_short_term_variability >  79.50
|    |    |    |    |--- light_decelerations <= 0.00
|    |    |    |    |    |--- class: 3.0
|    |    |    |    |--- light_decelerations >  0.00
|    |    |    |    |    |--- class: 1.0
|    |--- mean_value_of_long_term_variability >  68.50
```

```
|   |   |--- light_decelerations <= 0.00
|   |   |   |--- class: 3.0
|   |   |--- light_decelerations >  0.00
|   |   |   |--- mean_value_of_long_term_variability <= 77.00
|   |   |   |   |--- class: 1.0
|   |   |   |--- mean_value_of_long_term_variability >  77.00
|   |   |   |   |--- class: 3.0
|--- percentage_of_time_with_abnormal_long_term_variability >  0.55
|   |--- histogram_median <= 107.50
|   |   |--- histogram_number_of_peaks <= 220.50
|   |   |   |--- mean_value_of_short_term_variability <= 25.00
|   |   |   |   |--- histogram_median <= 102.00
|   |   |   |   |   |--- class: 2.0
|   |   |   |   |--- histogram_median >  102.00
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- mean_value_of_short_term_variability >  25.00
|   |   |   |   |--- class: 3.0
|   |   |--- histogram_number_of_peaks >  220.50
|   |   |   |--- histogram_tendency <= 23.50
|   |   |   |   |--- class: 3.0
|   |   |   |--- histogram_tendency >  23.50
|   |   |   |   |--- class: 1.0
|   |--- histogram_median >  107.50
|   |   |--- mean_value_of_long_term_variability <= 6.50
|   |   |   |--- abnormal_short_term_variability <= 0.00
|   |   |   |   |--- histogram_max <= 134.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- histogram_max >  134.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- abnormal_short_term_variability >  0.00
|   |   |   |   |--- accelerations <= 129.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- accelerations >  129.50
|   |   |   |   |   |--- class: 3.0
|   |   |--- mean_value_of_long_term_variability >  6.50
|   |   |   |--- histogram_number_of_peaks <= 185.50
|   |   |   |   |--- histogram_median <= 143.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- histogram_median >  143.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- histogram_number_of_peaks >  185.50
|   |   |   |   |--- mean_value_of_short_term_variability <= 49.00
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- mean_value_of_short_term_variability >  49.00
|   |   |   |   |   |--- class: 2.0
```

6 Conclusion

7 Bibliography

[1] Mushrooms data set: https://archive.ics.uci.edu/ml/datasets/Mushroom, downloaded 04.12.2020

[2] Source mushroom data: Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

[3] Fetal health data set: https://www.kaggle.com/andrewmvd/fetal-health-classification, downloaded 11.12.2020

[4] Source Fetal health data: Ayres de Campos et al. (2000) SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms. J Matern Fetal Med 5:311-318 (link)

[5] Fetal health walkthrough: https://www.kaggle.com/pariaagharabi/step-by-step-fetal-health-prediction-99-detailed

[6] Guide to building a decision tree: https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/, visited 04.12.2020

[7] Decision tree learning and the CART algorithm: https://en.wikipedia.org/wiki/Decision_tree_learning, visited 04.12.2020

[8] Feature selection with scikit-learn: https://scikit-learn.org/stable/modules/feature_selection.html#univariat feature-selection, visited 08.12.2020

[9] Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. ISBN 978-0-412-04841-8.

[10] Logical rules for classifying mushrooms: logical rules for mushrooms: Duch W, Adamczak R, Grabczewski K, Ishikawa M, Ueda H, Extraction of crisp logical rules using constrained backprop-agation networks - comparison of two new approaches, in: Proc. of the European Symposium on Artificial Neural Networks (ESANN'97), Bruge, Belgium 16-18.4.1997, pp. xx-xx

[11] On pruning decision trees: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_prun glr-auto-examples-tree-plot-cost-complexity-pruning-py, visited 03.12.2020

[12] Decision tree code: https://medium.com/datadriveninvestor/easy-implementation-of-decision-tree-with-python-numpy-9ec64f05f8ae, visited 06.12.2020

[12] Guide on feature selection for categorical variables: https://machinelearningmastery.com/feature-selection-with-categorical-data/, visited 08.12.2020

[13] Project 2: https://github.com/emiliefj/FYS-STK3155/blob/master/Project2/Report/Project