# Building a classification application to save lives

Alejandro Susillo Ridao

alexsurid@gmail.com

Date: 12-03-2019

# Table of Contents

# 1. Introduction

Ingestion of wild and potentially toxic mushrooms is common in the United States and many other parts of the world. US poison centers have been logging cases of mushroom exposure in The National Poison Data System (NPDS) annual publications for over 30 years. This study compiles and analyzes US mushroom exposures as reported by the NPDS from 1999 to 2016. **Over the last 18 years, 133,700 cases (7428 per year)** of mushroom exposure, mostly by ingestion, have been reported. The vast majority of reported ingestions resulted in no or minor harm, although some groups of mushroom toxins or irritants have been deadly. **Misidentification of edible mushroom** species appears to be **the most common cause** and may be preventable.

Therefore, the goal of this project is to **develop an application that will classify a mushroom between poisonous or edible**, depending on the characteristics of it.

# 2. Data

The data set is composed by 8124 entries, each one with 23 features. They are all categorical features and describe the following:

| Variable | Count of different values and values |
|---|---|
| **Class** | 2 -> ['p' 'e'] |
| **Cap-shape** | 6 -> ['x' 'b' 's' 'f' 'k' 'c'] |
| **Cap-surface** | 4 -> ['s' 'y' 'f' 'g'] |
| **Cap-color** | 10 -> ['n' 'y' 'w' 'g' 'e' 'p' 'b' 'u' 'c' 'r'] |
| **Bruises** | 2 -> ['t' 'f'] |
| **Odor** | 9 -> ['p' 'a' 'l' 'n' 'f' 'c' 'y' 's' 'm'] |
| **Gill-attachment** | 2 -> ['f' 'a'] |
| **Gill-spacing** | 2 -> ['c' 'w'] |
| **Gill-size** | 2 -> ['n' 'b'] |
| **Gill-color** | 12 -> ['k' 'n' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o'] |
| **Stalk-shape** | 2 -> ['e' 't'] |
| **Stalk-root** | 5 -> ['e' 'c' 'b' 'r' '?'] |
| **Stalk-surface-above-ring** | 4 -> ['s' 'f' 'k' 'y'] |
| **Stalk-surface-below-ring** | 4 -> ['s' 'f' 'y' 'k'] |
| **Stalk-color-above-ring** | 9 -> ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y'] |
| **Stalk-color-below-ring** | 9 -> ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c'] |
| **Veil-type** | 1 -> ['p'] |
| **Veil-color** | 4 -> ['w' 'n' 'o' 'y'] |
| **Ring-number** | 3 -> ['o' 't' 'n'] |
| **Ring-type** | 5 -> ['p' 'e' 'l' 'f' 'n'] |
| **Spore-print-color** | 9 -> ['k' 'n' 'u' 'h' 'w' 'r' 'o' 'y' 'b'] |
| **Population** | 6 -> ['s' 'n' 'a' 'v' 'y' 'c'] |
| **Habitat** | 7 -> ['u' 'g' 'm' 'd' 'p' 'w' 'l'] |

A we can see they all describe characteristics of a mushroom. For instance, for the class variable, ‘p’ means poisonous and ‘e’ edible.’

This dataset was downloaded form kaggle: https://www.kaggle.com/uciml/mushroom-classification

# 3. Problems to be Solved

The aim of this application is to correctly classify mushrooms between poisonous and edible depending on the characteristics the user provides. Doing so, our users could know when going out to get mushrooms if they can eat a specific mushroom or not, so with this application we are saving lives.

To make this application real, we must perform several steps:

1. **Data exploration**: Once we obtain the data set, we must explore it in order to know the variables we have, how many possible values there are for each variable paying attention to certain special values such as null values, unusual values, etc.
2. **Data cleaning**: There could be null values or missing values, if so, we should remove those instances from the data set.
3. **Data visualization**: Once the data set is clean, we could start making some graphs so that we could know better the distribution of our data set.
4. **Training model**: Selected the target variable, we may start training our algorithm and create our model.
5. **Evaluation model**: When the model is created, we must evaluate it using the accuracy metric. This metric will be obtained by comparing the results returned by the model and the real values.
6. **Conclusions**: Sum up all the information gathered and present results.

# 4. KDD

## 4.1. Data Processing

Firsly, I had to clean my dataset from missing values like '?' present in the variable 'stalk-root'. In order to do so, I used the following piece of code provided by the Numpy library from Python:

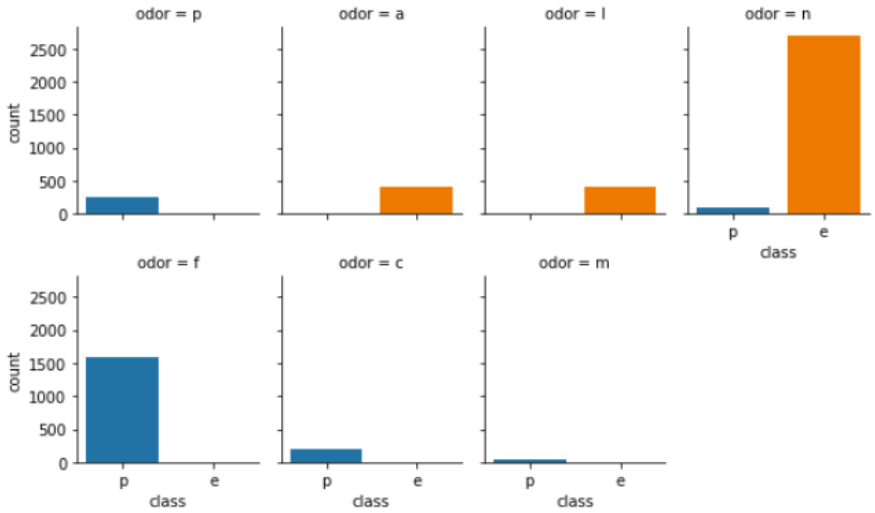
We replace all the '?' values with NaN values. Then, we drop all NaN from the dataset.

```
df = df.replace('?', numpy.NaN)
df.dropna(inplace=True)
```

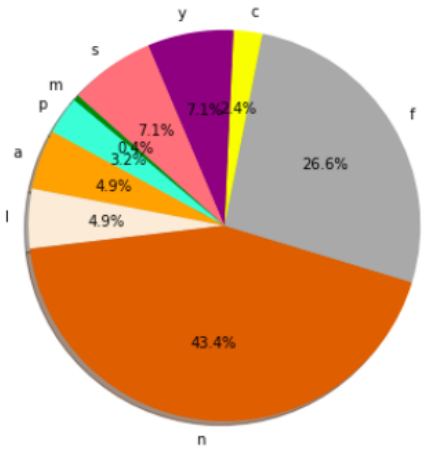
I had to consider null values as well so in order to find which ones were null I perform the following command:

```
df.isnull().sum().sort_values(ascending=False)
```

This time there were not null values. Therefore, the resulting dataset is a clean dataset without missing or null values. Next step will be visualizing the distribution of our features over the class variable.




As we can see from the plot, odor has a very unequal distribution. Most of the values are concentrated in those cases in which odor is none (n) or foul (f). In the first case, most of them are edible, whereas in the second one, most of them are poisonous. We now know how the features are distributed with respect to class, but we do not know yet the distribution of this variable with respect to all the dataset.




This is the distribution of the feature odor over the whole dataset. As we can observe, the none value for this feature supposes the 43.4% of all the posible values of this feature in the dataset. Futhermore, in case this value is equal to foul, this assumes the 26.6% of all the posible values of this feature in all the dataset.

The dataset is composed of 23 features and I have performed this analysis for all of them. Therefore, I am not going to show that analysis in this report to every one of them because I do not want to make the reader feel this is an endless report, in case you are interested in seeing all the individual analysis for each feature, feel free to have a look at the Jupyter notebook.

Secondly, I had to transform my dataset so that it contains numerical values. I used the method *pd.get_dummies(df[cols])* from Pandas library to perform this task. Later, all these transformed columns were joined with the original dataset, substituting the original columns. With this change, I had a dataset with numerical values that range from 0 to 1, as we can see in the image below.

| | class | bruises | gill-attachment | gill-spacing | gill-size | stalk-shape | veil-type | cap-shape_b | cap-shape_c | cap-shape_f | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... |
| **1** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| **2** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| **3** | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... |
| **4** | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | ... |

## 4.2. Data Mining Methods and Processes

### Classification

To perform the classification task, I have chosen the following classification algorithms: Decision Trees, Random Forest, K-Nearest Neighbor, Naïve Bayes and XG Boosting. I will explain briefly why I chose them and the advantages each one has over the others.

#### *Decision Trees*

Decision trees is a predictive model used to go from observation about an item (represented in the branches) to conclusions about the item's target value represented in the leaves. In our case in which the target variable is a discrete value, we will have a classification tree. The reason I have chosen this kind of model is because it simple to understand and interpret, have value even with little hard data and can be combined with other decision techniques.

#### *Random Forest*

Random forests are an ensemble learning method for classification (and regression) that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. Random forests correct for decision trees' habit of overfitting to their training set and usually perform better than these.

#### *K-Nearest Neighbor*

KNN is known as a lazy learning classifier. An object is classified by a plurality vote of its neighbors with the object being assigned to the class most common among its k nearest neighbors. That vote depends on the distance calculated between the particular record and the remaining records, so it is important to normalize the features values. It is fast and perform pretty well.

#### *Naïve Bayes*

It is a probabilistic algorithm based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. It is easy to implement, fast, requires less training data and its predictions are hold to probabilistic models.

### *XG Boosting*

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is well known to provide better solutions than other machine learning algorithms. In fact, since its inception, it has become the "state-of-the-art" machine learning algorithm to deal with structured data. I have chosen this algorithm due to its speed and performance (faster than other ensemble classifiers), its core algorithm is parallelizable (it can harness the power of multi-core computers), consistently outperforms other algorithm methods and wide variety of tuning parameters.

## Association rules

This project is going to include association rules as well, so that while the user is setting the characteristics of a mushroom, if they match with any of the association rules, then the application is going to tell the user that this mushroom is poisonous or edible. Thus, we are using association rules to avoid the user entering all the characteristic of a mushroom. For instance, if we already know that a mushroom with a foul smell is poisonous, the application is going to warm the user that, independently of all the other characteristics, that mushroom is poisonous.

To compute the association rules, we are going to make use of the Apriori algorithm. It proceeds by identifying the frequent individual items in the dataset and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the dataset.

# 5. Evaluations and Results

## 5.1. Evaluation Methods

### Classification

This project focused on three variables to come up with a decision as to which algorithm to choose: accuracy, time, and number of false negative predictions.

Firstly, to obtain an accuracy I considered both Hold-out evaluation and K-Fold evaluation as evaluation methods for the above algorithms, but the selected dataset was too small to perform a Hold-out evaluation (it was always giving me a 100% of accuracy which is unreal). Therefore, I had to use K-Fold evaluation (cv=5) to see the accuracy of each algorithm, compare it with the accuracy from the other algorithms and finally, decide which algorithm was better for my project according to the accuracy metric.

Secondly, I compared the execution times of the algorithms (time that the algorithms took to make the prediction) so that the best algorithm is not only the one with the best accuracy, but also the one which performs faster.

Thirdly and to end up, I took into consideration the number of false negatives obtained from the confusion matrix of each algorithm. This is due to the fact that our application will be used by users that want to know if a mushroom is poisonous or not. If the application tells them that it is not when it actually is, we will be killing people, that is why I had to take those algorithms with the least number of false negatives.

### Association rules

In order to evaluate the association rules and get the correct association rules, we had to consider several metrics like minimum support, confidence and lift. For the first metric, we set a minimum support of 0.07 for all the rules. Support is used to measure the abundance or frequency (often interpreted as significance or importance) of an itemset in the dataset. We refer to an itemset as a "frequent itemset" if our support is larger than a specified minimum-support threshold.

Then, confidence is the probability of given the antecedent, knowing the consequent. Finally, the lift metric with a threshold of 1.0 is commonly used to measure how much more often the antecedent and consequent of a rule occur together than we would expect if they were statistically independent.

## 5.2. Results and Findings

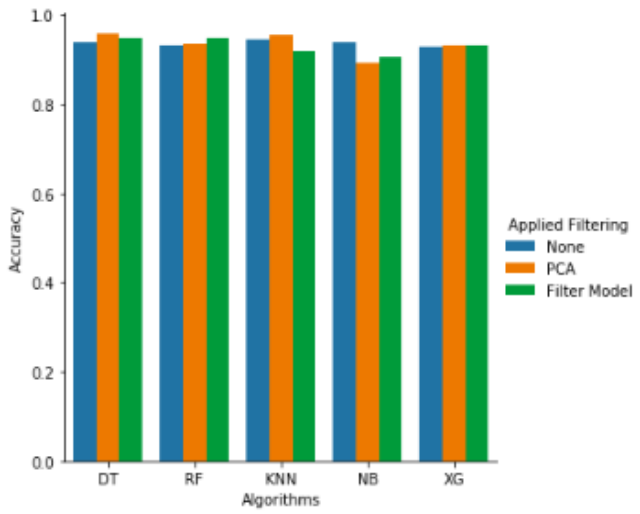
I have executed all the classification algorithms in three different situations. One of them, with the original data transformed so that it contains numerical features. The second situation, I applied feature reduction to the dataset with the PCA technique. We selected the 3 top PCA as it has most of the variance of the dataset (48%). And, the third situation, I applied a feature selection with the Filter Model approach to the dataset, retrieving as the relevant features the following: *class, bruises, stalk-shape, stalk-surface-above-ring, stalk-surface-below-ring, ring-type_l, spore-print-color_h*.

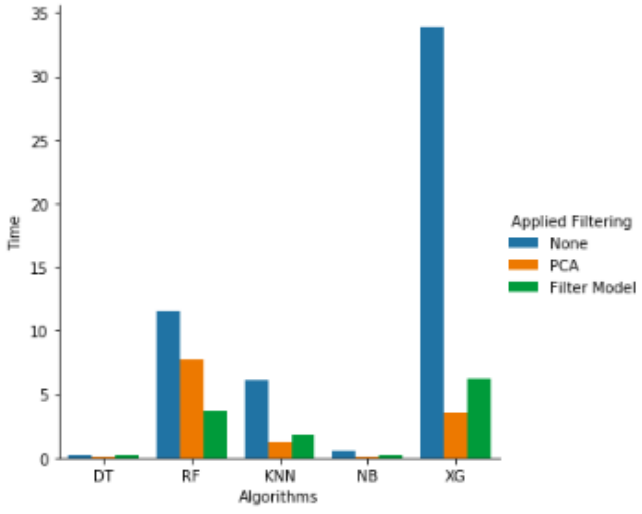
All the results for these situations are presented in the table below where time is expressed in seconds and accuracy as a percentage.

| | Decision Trees | | Random Forest | | K-Nearest Neighbor | | Naïve Bayes | | XG Boosting | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Accuracy | Time | Accuracy | Time | Accuracy | Time | Accuracy | Time | Accuracy | Time |
| Without feature selection or reduction | 93.7% | 0.246 | 93.12% | 11.6 | 94.35% | 6.08 | 93.7% | 0.59 | 93% | 33.9 |
| Feature reduction with PCA | 95.66% | 0.14 | 93.64% | 7.67 | 95.55% | 1.23 | 89.33% | 0.064 | 93.14% | 3.6 |
| Feature selection with Filter Model | 94.76% | 0.2 | 94.76% | 3.7 | 92% | 1.78 | 90.5% | 0.22 | 93.15% | 6.18 |

In order to make the previous numbers more understandable, I present here two plots to show these results in a more visual way that is easier to comprehend.

| Accuracy over 1.0 (0.6 is 60%) | Time in seconds |
|---|---|




These times were measured with the following computer specifications:

| | |
|---|---|
| **Operating system** | macOS Mojave version 10.14.6 |
| **Processor** | 1,8 GHz Intel Core i5 |
| **Memory** | 4 GB 1600 MHz DDR3 |
| **Graphics** | Intel HD Graphics 4000 1536 MB |

In general, we could say that the classification algorithms' accuracy is more or less the same. Nevertheless, there are notable differences when comparing them in terms of time. DT and NB are the fastest ones, while KNN and RF show an increase in their times if feature selection or reduction is not applied. With both of them, the time decreases applying feature reduction with PCA, even though, in RF we can see an improvement when applying feature selection whereas KNN gets worse. Finally, XG Boosting is a particular case when observing its time. It is by

far the algorithm that took more time to make the prediction, although when applying feature selection and reduction it improved much more.

Having a closer look to the results from the table above, we can see that the algorithm with the best accuracy without applying any of the feature techniques is the KNN (94.35%). Nonetheless, it is not the fastest one, which is Naïve Bayes (0.59s). Applying PCA, we obtained that this time is DT the algorithm with the best accuracy (95.66%), but not the fastest, that is once again Naïve Bayes (0.064s). Finally, with Filter Model applied to the features of the dataset, DT and RF are the ones with the best accuracy (94.76%). This time Naïve Bayes is not the only one with the fastest time, but also DT with a time of 0.2 seconds.

As mention before in this document, it is important to make emphasis to the false negative retrieved from the confusion matrix. These values were obtained without applying any of the feature techniques.

| **Algorithms** | **Decision Trees** | **Random Forest** | **K-Nearest Neighbor** | **Naïve Bayes** | **XG Boosting** |
|---|---|---|---|---|---|
| **False negatives** | 292 | 292 | 256 | 260 | 300 |

We can observe that KNN has the fewest number of them, while XG Boosting although having a very good accuracy in all the situations presented above, has the highest number of false negatives.

Moving on to the associations rules results, I filtered them so that I got only those rules which consequent value is only *class_p* or *class_e*, that is, poisonous mushroom or edible mushroom. They are ordered in descending order by the lift value, and I limited the number of antecedent the rules could have to three.

In the first rule for example, if the user introduces *spore-print-color* as chocolate (*h*) and *ring_type* as large (*l*), the mushroom will be poisonous (*class_p*) with a confidence of 1.0 (a very high value) and a lift value of 2.61, which means that the consequent and the antecedent are dependent of each other and therefore, this rule would be a good one to make predictions in the future.

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | antecedent_len |
|---|---|---|---|---|---|---|---|---|---|---|
| **533** | (spore-print-color_h, ring-type_l) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 2 |
| **134** | (odor_f, stalk-surface-below-ring_k) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 2 |
| **63** | (ring-type_l) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 1 |
| **364** | (stalk-surface-above-ring_k, stalk-shape_e) | (class_p) | 0.236003 | 0.381999 | 0.236003 | 1.0 | 2.617811 | 0.145850 | inf | 2 |
| **488** | (stalk-shape_e, stalk-surface-below-ring_k) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 2 |
| **49** | (stalk-surface-below-ring_k) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 1 |
| **482** | (stalk-surface-below-ring_k, bruises_f) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 2 |
| **3716** | (spore-print-color_h, stalk-shape_e, stalk-sur... | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 4 |
| **558** | (stalk-shape_e, ring-type_l) | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 2 |
| **1408** | (spore-print-color_h, stalk-surface-above-ring... | (class_p) | 0.229624 | 0.381999 | 0.229624 | 1.0 | 2.617811 | 0.141908 | inf | 3 |

The table below shows the same results as above but with edible as the class. Just point out that this time the lift value has decreased a bit, although the conclusion remains the same.

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | antecedent_len |
|---|---|---|---|---|---|---|---|---|---|---|
| **269** | (odor_n, stalk-shape_t) | (class_e) | 0.442240 | 0.618001 | 0.442240 | 1.0 | 1.618119 | 0.168935 | inf | 2 |
| **3035** | (odor_n, stalk-surface-above-ring_s, bruises_t... | (class_e) | 0.306166 | 0.618001 | 0.306166 | 1.0 | 1.618119 | 0.116955 | inf | 4 |
| **1213** | (odor_n, stalk-surface-below-ring_s, stalk-sha... | (class_e) | 0.374203 | 0.618001 | 0.374203 | 1.0 | 1.618119 | 0.142945 | inf | 3 |
| **587** | (bruises_f, stalk-shape_t) | (class_e) | 0.136074 | 0.618001 | 0.136074 | 1.0 | 1.618119 | 0.051980 | inf | 2 |
| **2975** | (odor_n, stalk-surface-above-ring_s, stalk-sur... | (class_e) | 0.340184 | 0.618001 | 0.340184 | 1.0 | 1.618119 | 0.129950 | inf | 4 |
| **3066** | (odor_n, bruises_t, stalk-surface-below-ring_s... | (class_e) | 0.306166 | 0.618001 | 0.306166 | 1.0 | 1.618119 | 0.116955 | inf | 4 |
| **1188** | (odor_n, bruises_f, stalk-surface-below-ring_s) | (class_e) | 0.089298 | 0.618001 | 0.089298 | 1.0 | 1.618119 | 0.034112 | inf | 3 |
| **1241** | (odor_n, bruises_f, stalk-shape_t) | (class_e) | 0.136074 | 0.618001 | 0.136074 | 1.0 | 1.618119 | 0.051980 | inf | 3 |
| **1253** | (odor_n, bruises_t, stalk-shape_t) | (class_e) | 0.306166 | 0.618001 | 0.306166 | 1.0 | 1.618119 | 0.116955 | inf | 3 |
| **1161** | (odor_n, stalk-surface-above-ring_s, stalk-sha... | (class_e) | 0.374203 | 0.618001 | 0.374203 | 1.0 | 1.618119 | 0.142945 | inf | 3 |

# 6. Conclusions and Future Work

## 6.1. Conclusions

There are several conclusions arisen from the results presented in the previous section:

1. I would use the KNN algorithm to make my predictions in the application.
2. It must be kept in mind that it will take 6 seconds to make the prediction but as it has a very good accuracy it is a risk I would take because I prefer accuracy than speed.
3. KNN is in fact the algorithm with less false negative.
4. I would use those rules which consequent is poisonous class for two reasons: It has a higher lift value than the other ones (class edible), and as the goal of the application is to save lives, I would warm the user just in case it is poisonous.

## 6.2. Limitations

I do not know how far these results can reach, i.e., with another dataset may be larger I do not know if the accuracy of the classification and association rules algorithms would be the same or may get worse. In addition, the times presented here were taken with a computer, which normally has better performance than a mobile phone that where this application would fit better for our goals.

## 6.3. Potential Improvements or Future Work

This is supposed to be in the future an application that every user can use when going out to get mushrooms, so, due to this, it would be great if in the future somebody can implement an application with a front-end capable of allowing the user introduce the characteristics of a mushroom and then, the back-end of the application, use this classification algorithms to make the prediction and show the user the result. Moreover, as it is an application that users will use to know if a mushroom is edible or poisonous, I would do more research on how to improve algorithm's accuracy and may be find patterns like the ones returned by the association rules so that with some particular input variables, we get a safe result about if the user can or cannot eat the mushroom. What I am looking for with this is that the user did not have to introduce one by one all the parameters which in this case are 22 (23 features but one of them is the target), making the application easier to use.

**Note**: I have tried to apply SVM algorithm for the classification but when I use as input variables those from PCA, it gets stuck. It is still implemented in the Jupyter Notebook but as I have not the accuracy and the time value for PCA with this algorithm I considered not to include it in this report.