Project Report
Mushroom Classification

Date : 13/1/2022

Contents

1. Introduction

Mushroom is found to be one of the best nutritional foods with high proteins, vitamins, and minerals. It contains antioxidants that prevent people from heart disease and cancer. Around 45000 species of mushroom are found to be existing worldwide. Among these, only some of the mushroom varieties were found to be edible. Some of them are really dangerous to consume.

In order to distinguish between the edible and poisonous mushrooms in the mushroom dataset which was obtained from UCI Machine Learning Repository, some data mining techniques are used. Weka is a data mining tool with various machine learning algorithms that can pre-process, analyze, classify, visualize and predict the given data. Thus, to select the attributes that help better classify mushrooms, the Wrapper method and Filter method in Weka is used to identify the best attributes for the classification. The attributes ‘odor’ and ‘spore_print_color’ were chosen to be the best ones that contributed to the better classification of edible and poisonous mushrooms. After identifying the key attributes, classification is performed, a decision tree is constructed based on those attributes, and its Precision, Recall, and F-Measure values are analyzed.

2. Problem Statement

The Audubon Society Field Guide to North American Mushrooms contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom (1981).

Each species is labelled as either definitely edible, definitely poisonous, or maybe edible but not recommended. This last category was merged with the toxic category. The Guide asserts unequivocally that there is no simple rule for judging a mushroom's edibility, such as "leaflets three, leave it be" for Poisonous Oak and Ivy. The main goal is to predict which mushroom is poisonous & which is edible.

3. Objective

Mushroom is found to be one of the best nutritional foods with high proteins, vitamins, and minerals. It contains antioxidants that prevent people from heart disease and cancer. Around 45000 species of mushroom are found to be existing worldwide . Among these, only some of the mushroom varieties were found to be edible. Some of them are really dangerous to consume.

In order to distinguish between the edible and poisonous mushrooms in the mushroom dataset which was obtained from Kaggle, some data mining techniques are used. data mining tool with machine learning algorithms that can pre-process, analyze, classify, visualize and predict the given data. Thus, to select the attributes that help better classify mushrooms, and for classifying the mushroom various models are used with their best hyperparameter with best accuracies.

4. Data Information

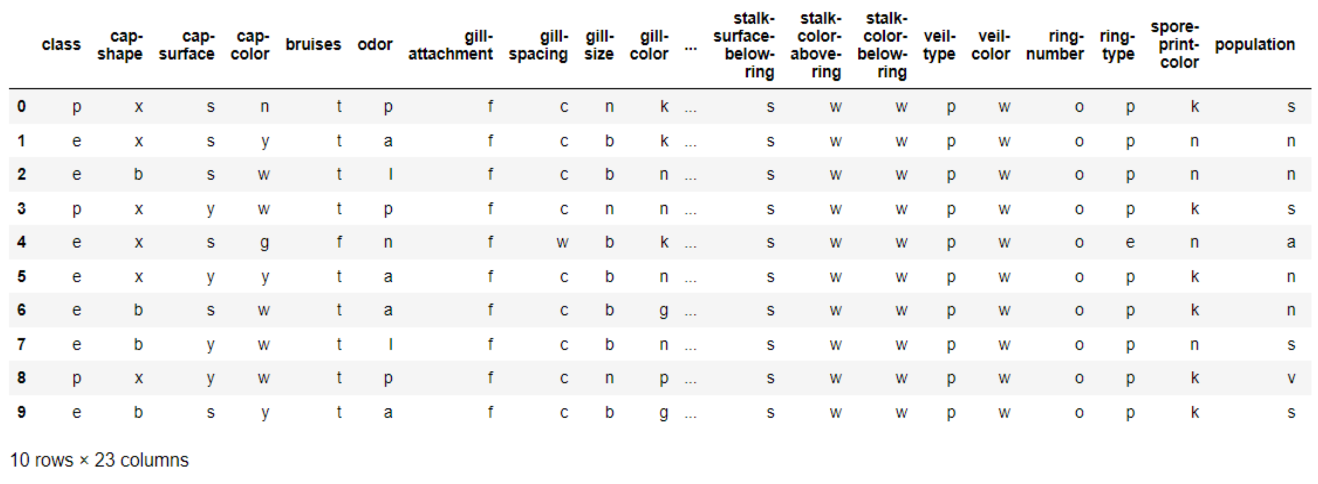
4.1 Data Requirements

Although this dataset was originally contributed to the UCI Machine Learning repository nearly 30 years ago, mushroom hunting (otherwise known as "shrooming") is enjoying new peaks in popularity. Learn which features spell certain death and which are most palatable in this dataset of mushroom characteristics. And how certain can your model be?

4.2 Data Content

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")

data=pd.read_csv('mushrooms.csv')

data.head()
```



	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a
5	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	k	n
6	e	b	s	w	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	n
7	e	b	y	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	s
8	p	x	y	w	t	p	f	c	n	p	...	s	w	w	p	w	o	p	k	v
9	e	b	s	y	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	s

10 rows × 23 columns

4.3 Data Analysis

4.3.1 Attribute Information

(classes: edible=e, poisonous=p)

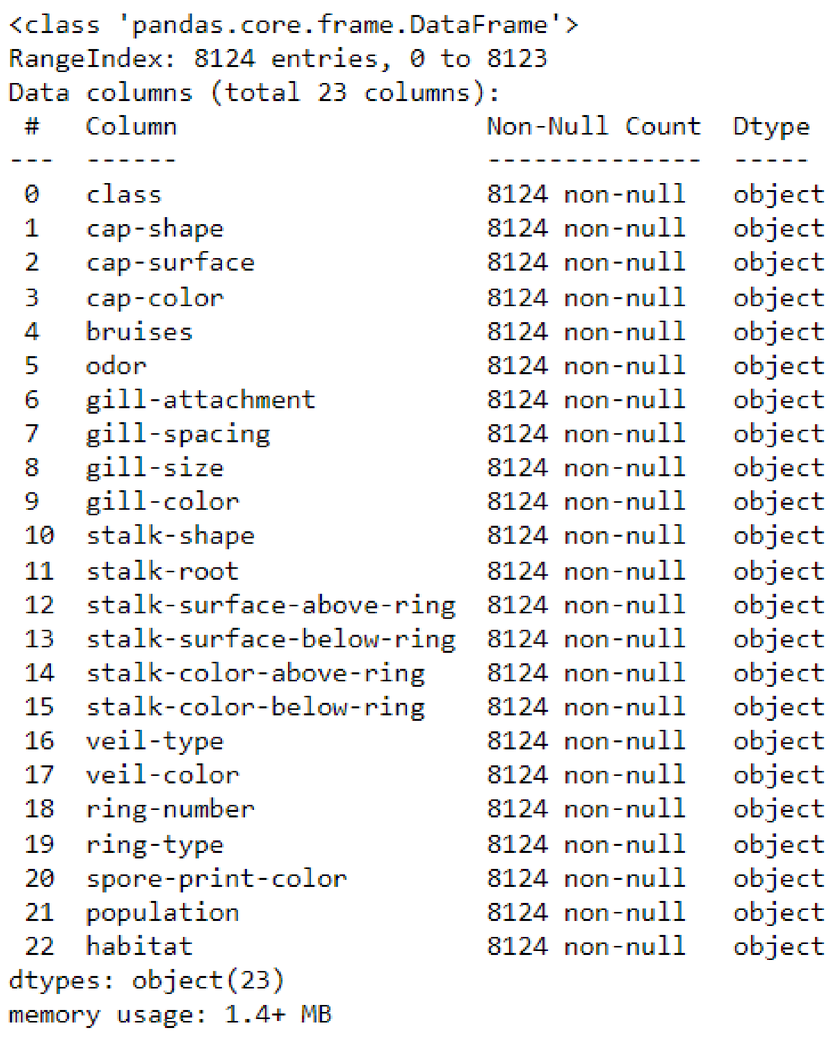
- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color:brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u, red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor:almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,punge nt=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- 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
- stalk-shape: enlarging=e,tapering=t
- stalk-root:bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring:brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e, white=w,yellow=y
- stalk-color-below-ring:brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e, white=w,yellow=y
- veil-type: partial=p,universal=u
- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type:cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing= s,zone=z

- spore-print-color:black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white= w,yellow=y
- population:abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat:grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

4.3.2 Detail information of dataset

Dataset contains 23 columns and 8124 rows. All the columns contain object data type and from below you can see that there are no Null values in any columns.

```
data.info()
```

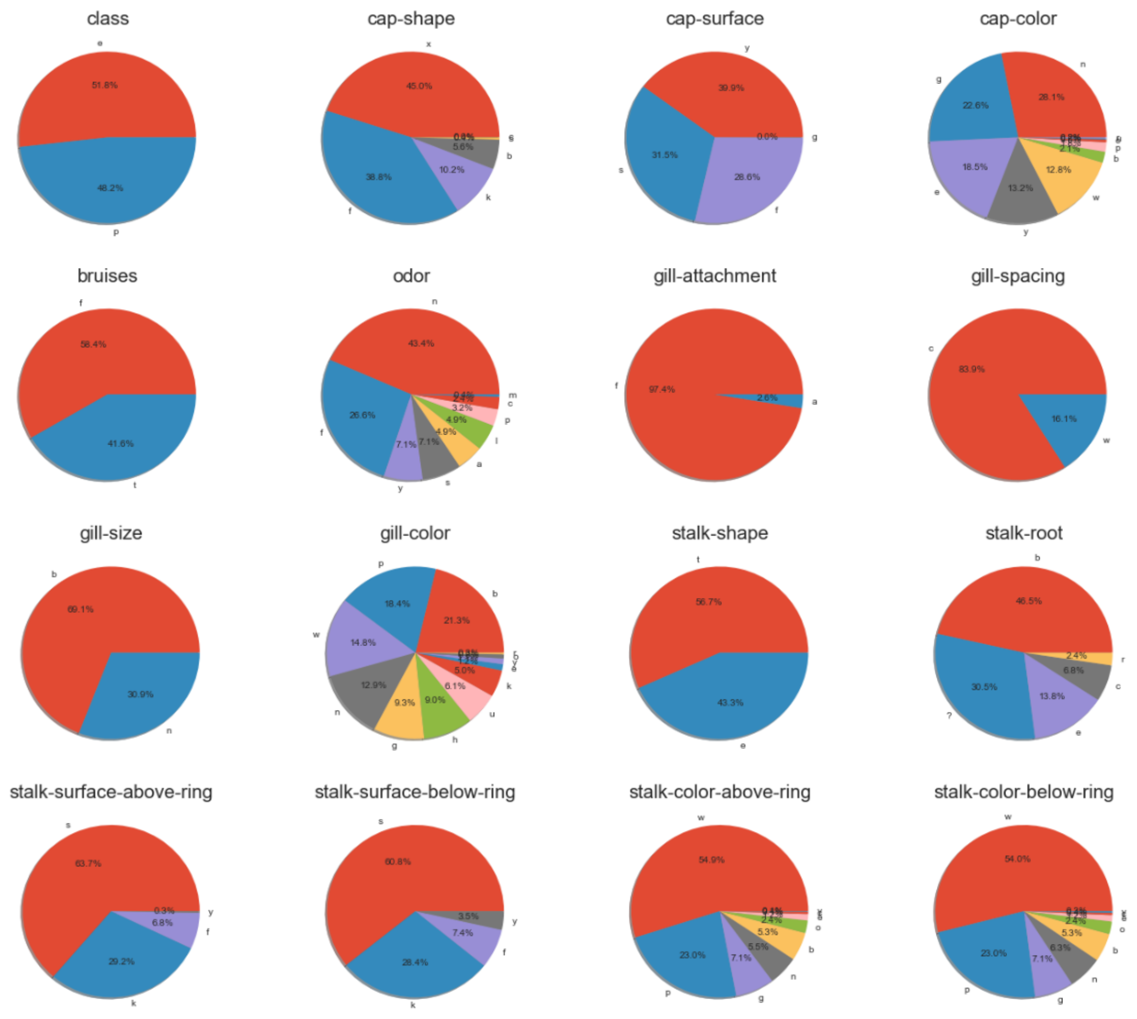


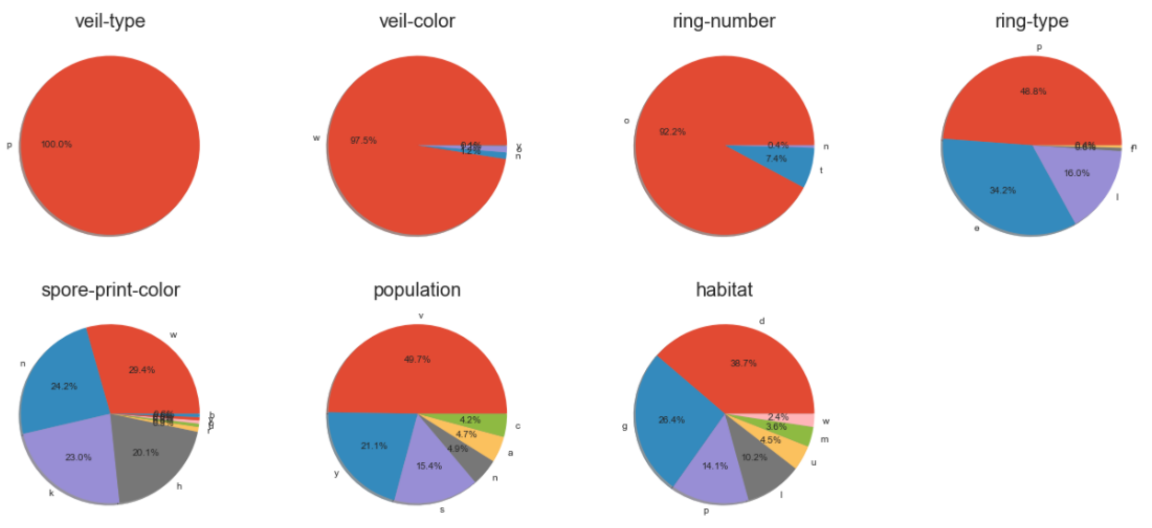
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

4.3.3 Columns Analysis

Value counts for each column with a pie graph. For that ‘matplotlib’ library is used as you can see below.

```python
from matplotlib import pyplot as plt

col= data.columns

plt.figure(figsize=(18,50))
for i in range(23):
    a = data[col[i]].value_counts()
    plt.style.use("ggplot")
    plt.subplot(8,3,i+1)
    plt.pie(a, labels= a.index, shadow= True, autopct= '%1.1f%%')
    plt.title(col[i])
data.head()
```





5. Making Machine Learning Models

The project is all about classification of mushroom whether it is poisonous or not, so for classification I used several classification models like logistic regression, SVM, KNeighbours, Ensemble methods like RandomForest, AdaboostClassifier, GradientBoosting etc.
Following steps I used in coding:

5.1 Importing the necessary libraries

Importing all the libraries which are going to use for further operations or need for creating the models.

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier , AdaBoostClassifier ,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```

5.2 Assigning and Splitting the data

Assigning the dependent variable(x) and independent variables(y) as you can see below. Then creating dummy variables of independent variables, because they are in the form of categorical data.
For the model's training and testing purposes I split the data, 60% of the data is used for training and the remaining 40% is used for testing the model.

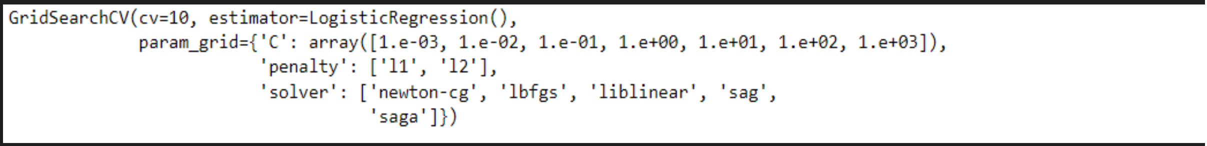
```
# assigning ‘p’ as 0 and ‘e’ as 1
x=data['class'].map({'p':0,'e':1})

#independent variables
y=data.loc[:,data.columns!='class']
y=pd.get_dummies(y)

train_x, test_x, train_y, test_y= train_test_split(x,y,test_size=0.4,
random_state=10)
train_x.shape, test_x.shape, train_y.shape, test_y.shape
```

5.3 Making the models

5.3.1 Making the model using Logistic Regression

Logistic Regression works in such a way that it simply converts the independent variable into the expression of probability ranging 0-1 with respect to dependent variable.
You can see in the code that I used Used GridSearchCV for finding best hyperparameters, for getting better accuracy.

```
model1= LogisticRegression()
param1={"C":np.logspace(-3,3,7), "penalty":["l1","l2"],'solver':['newton-cg',
'lbfgs', 'liblinear', 'sag', 'saga']}
model1_GCV= GridSearchCV(model1,param1, cv=10, verbose=0)
model1_GCV.fit(train_x, train_y)
```



```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]),
                         'penalty': ['l1', 'l2'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag',
                                    'saga']})
```

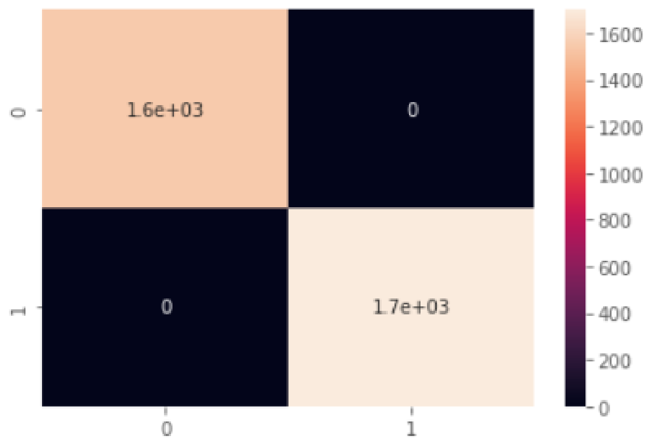
```
# creating a empty dictionary for collecting the accuracies of all the models
final={}

# to get best estimator
print(model1_GCV.best_estimator_)
```

```
LogisticRegression(penalty='l1', solver='liblinear')
```

```
# Testing the model
p1=model1_GCV.predict(test_x)

# Accuracy of the model
final[type(model1).__name__]=accuracy_score(p1, test_y)
print(accuracy_score(p1, test_y))
```

```
1.0
```

```
# Confusion matrix
cm1 = confusion_matrix(p1, test_y)
sns.heatmap(cm1, annot= True)
```



5.3.2 Making the model using RandomForestClassifier

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

```
model2= RandomForestClassifier()
param2={'criterion':['gini', 'entropy'],
        'n_estimators':[100,140,180,200,300],
        'max_depth':[1,2,3],
        'bootstrap':[True, False]}

model2_GCV= GridSearchCV(model2,param2, cv=10,verbose=0)
```

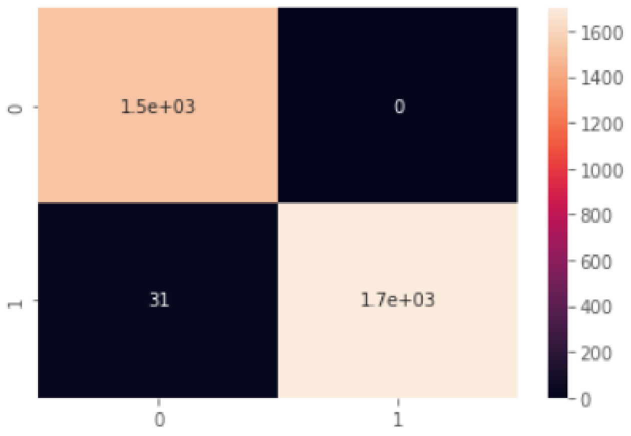
```
model2_GCV.fit(train_x, train_y)

# to get best estimator
print(model2_GCV.best_estimator_)
```

```
RandomForestClassifier(bootstrap=False, max_depth=3, n_estimators=300)
```

```
# Testing the model
p2=model2_GCV.predict(test_x)

# Accuracy of the model
final[type(model2).__name__]=accuracy_score(p2, test_y)
print(accuracy_score(p2, test_y))
```

```
0.9904615384615385
```

```
# Confusion matrix
cm2 = confusion_matrix(p2, test_y)
sns.heatmap(cm2, annot= True)
```



5.3.3 Making the model using AdaBoostClassifier

Adaboost is a boosting algorithm , is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

```
model3= AdaBoostClassifier()
param3= {'base_estimator':[DecisionTreeClassifier(max_depth=1)],
         'n_estimators':[100,140,180,200,300],
         'learning_rate':[0.01, 0.1, 1, 10,100 ],
         'algorithm':['SAMME', 'SAMME.R']
```

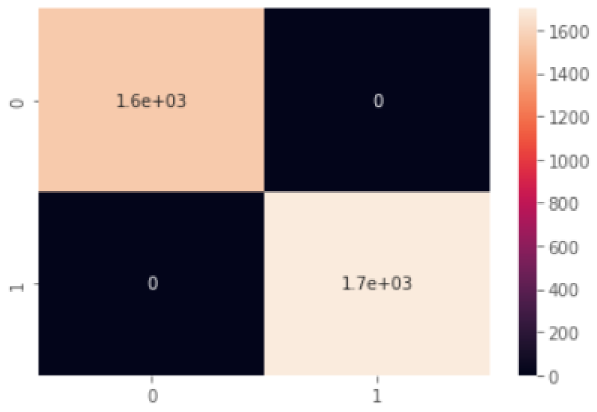
```
        }

model3_GCV= GridSearchCV(model3,param3, cv=10, verbose=0)
model3_GCV.fit(train_x, train_y)

# to get best estimator
print(model3_GCV.best_estimator_)
```

```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=1, n_estimators=100)
```

```
# Testing the model
p3=model3_GCV.predict(test_x)

# Accuracy of the model
final[type(model3).__name__]=accuracy_score(p3, test_y)
print(accuracy_score(p3, test_y))
```

```
1.0
```

```
# Confusion matrix
cm3 = confusion_matrix(p3, test_y)
sns.heatmap(cm3, annot= True)
```



5.3.4 Making the model using GradientBoostingClassifier

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor’s error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of the predecessor as labels.

```
model4 =GradientBoostingClassifier()
```

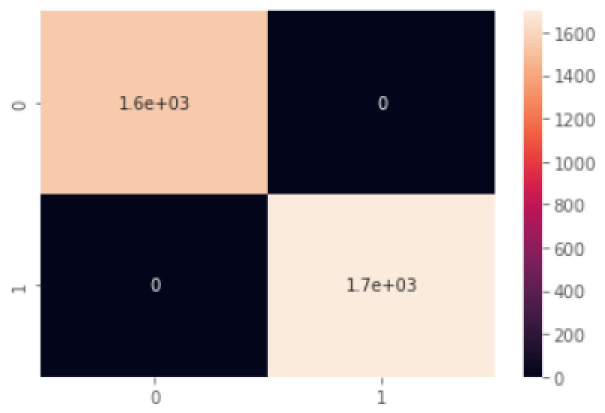
```
param4= {'max_depth':[1,2,3],
         'n_estimators':[100,140,180,200,300],
         'learning_rate':[0.01, 0.1, 1, 10,100 ]}

model4_GCV= GridSearchCV(model4,param4, cv=10, verbose=0)
model4_GCV.fit(train_x, train_y)

# to get best estimator
print(model4_GCV.best_estimator_)
```

```
GradientBoostingClassifier(max_depth=2, n_estimators=300)
```

```
# Testing the model
p4=model4_GCV.predict(test_x)

# Accuracy of the model
final[type(model4).__name__]=accuracy_score(p4, test_y)
print(accuracy_score(p4, test_y))
```

```
1.0
```

```
# Confusion matrix
cm4 = confusion_matrix(p4, test_y)
sns.heatmap(cm4, annot= True)
```



5.3.5 Making the model using SVC

SVC, or Support Vector Classifier, is a supervised machine learning algorithm typically used for classification tasks. SVC works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes.

```
model5= SVC()
```

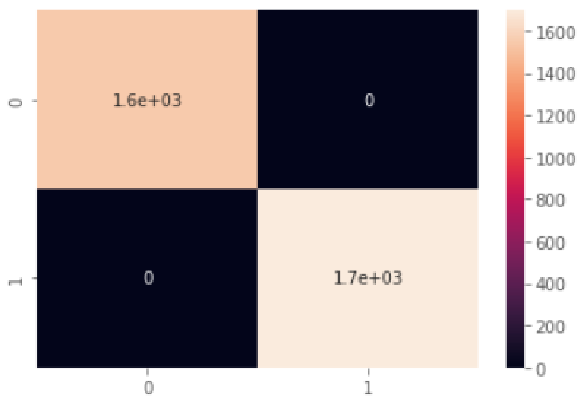
```
param5= {'C':[0.05,0.1,1,10,100],
         'gamma':[0.05,0.1,1,10,100]
        }

model5_GCV= GridSearchCV(model5,param5, cv=10, verbose=0)
model5_GCV.fit(train_x, train_y)

# to get best estimator
print(model5_GCV.best_estimator_)
```

```
SVC(C=1, gamma=0.05)
```

```
# Testing the model
p5=model5_GCV.predict(test_x)

# Accuracy of the model
final[type(model5).__name__]=accuracy_score(p5, test_y)
print(accuracy_score(p5, test_y))
```

```
1.0
```

```
# Confusion matrix
cm5 = confusion_matrix(p5, test_y)
sns.heatmap(cm5, annot= True)
```



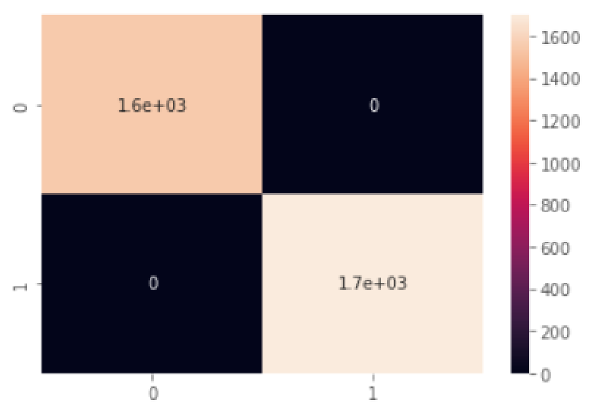
5.3.6 Making the model using KNeighborsClassifier

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

```
model6=KNeighborsClassifier()
model6.fit(train_x, train_y)

# Testing the model
p6=model6.predict(test_x)

# Accuracy of the model
final[type(model6).__name__]=accuracy_score(p6, test_y)
accuracy_score(p6, test_y)
```

```
1.0
```

```
# Confusion matrix
cm6 = confusion_matrix(p6, test_y)
sns.heatmap(cm6, annot= True)
```

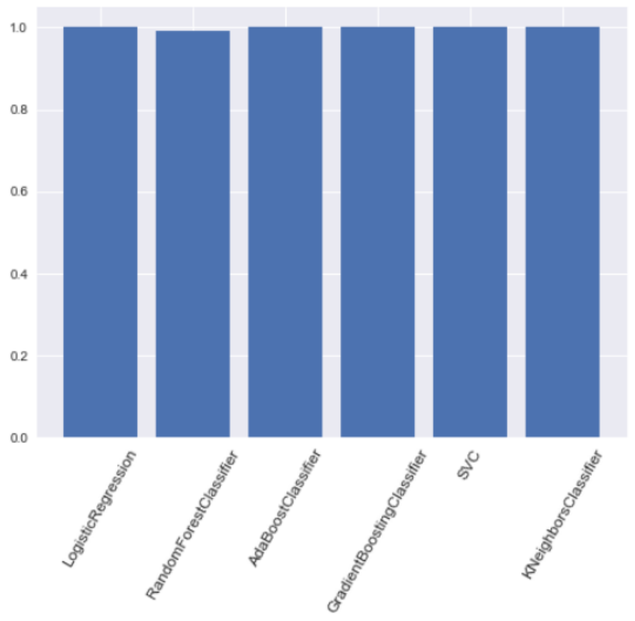


5.3.7 Accuracies of the models

```
print(final)
```

```
{'LogisticRegression': 1.0,
 'RandomForestClassifier': 0.9904615384615385,
 'AdaBoostClassifier': 1.0,
 'GradientBoostingClassifier': 1.0,
 'SVC': 1.0,
 'KNeighborsClassifier': 1.0}
```

```
plt.style.use("seaborn")
plt.figure(figsize=[8,6])
plt.bar(final.keys(),final.values() )
plt.xticks(rotation=60,fontsize=12);
```



6. Next steps

This project is about the methods of pre-processing, steps to identify the key attributes that help in the better classification of edible and poisonous mushrooms. From above accuracies I can say that models with their hyperparameters give incredible results.
In the next step I'm going to start to make a website which can predict whether the mushroom is poisonous or not , using tools like flask, MangoDB, HTML, CSS and to make this website live I am going to use Heroku.