Sri Lanka Institute Of Information Technology

Identification Of Mushrooms using Random Forest Classification Algorithm

Project Report

SE4060 - Machine Learning

IT16072848

S.M.R.B Saluwadana

Table of Contents

List of Figures

1. Introduction

Mushrooms are considered as vegetables. Most of the people are added mushrooms to their meal. Because mushrooms provide several important factors to human body. They contain some important nutrients to humans like vitamins , proteins and so on. They are also good for some health problems like cancers , heart diseases and so on.

But the problem is some mushrooms are poisonous. More than 90% people died in each year because of eating mushrooms which are poisonous. To reduce this problem identification of mushrooms are needed for society.

Figure 1 - Poisonous mushroom

Figure 2 - Edible mushroom

2 Proposed Solution

In this assignment I have proposed a mechanism (algorithm) to identify mushrooms which are edible and poisonous. To identify mushrooms I have used techniques in Random Forest Classification algorithm in machine learning.

3 Dataset

3.1 Description of a Dataset

Before starting to predict whether given mushroom is edible or poisonous we need to find out a dataset which is related to this context. For prediction purposes I have used a dataset in a UCI machine learning repository. Following link contains mushroom database.

https://archive.ics.uci.edu/ml/datasets/Mushroom

In this dataset have 8124 instances and 22 attributes. And also there are some missing values. And also dataset is a multivariate.

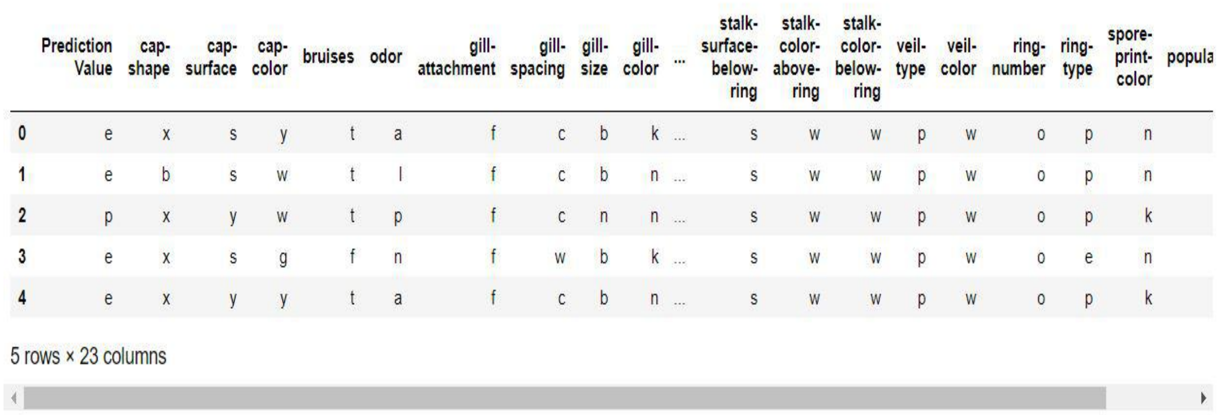
3.2 Description of Features

Mushroom database contains categorical attributes. All attributes are nominally valued. Attributes contain some important features of a mushroom that are more useful to predict whether given mushroom is edible or poisonous.And also their is a class attribute which contains information about a given mushroom is edible or poison. that is the outcome of this prediction.

4 Methodology

4.1 Data Preprocessing

As a first stage of data preprocessing we need to import our dataset to jupyter notebook. Mainly that dataset is read from the Excel (csv) file. Following figure shows the first five instances contains in the mushroom database.

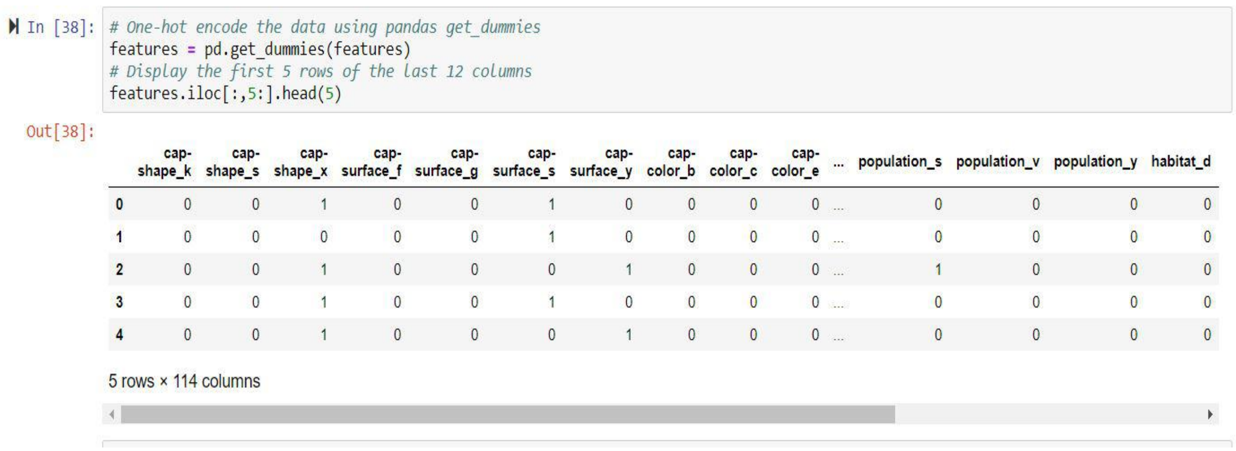


	Prediction Value	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	popula
0	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	
1	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	
2	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	
3	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	
4	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	k	

5 rows × 23 columns

Figure 3 - First five instances of a dataset

After importing a dataset we should convert those data to numerical format. Because initially we have a dataset which is in a strings format. All machine learning models deals with numbers. That's a reason for doing this. In python we use pandas library to doing this. Following code snip convert data into numerical format. And also we use Onehotencoder class to put those data into categories. As a example in cap-shape column we have different values. Cap-shape can be bell(b) , conical(c) , convex(x) , flat(f) , knobbed(k) and sunken(s). By using onehotencoder this values are categorized into several columns.



```
In [38]: # One-hot encode the data using pandas get_dummies
         features = pd.get_dummies(features)
         # Display the first 5 rows of the last 12 columns
         features.iloc[:,5:].head(5)
```

Out[38]:

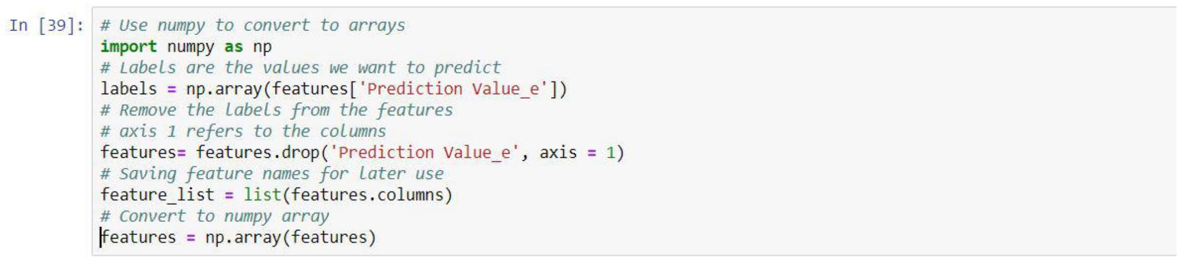
	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_b	cap-color_c	cap-color_e	...	population_s	population_v	population_y	habitat_d
0	0	0	1	0	0	1	0	0	0	0	...	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	...	1	0	0	0
3	0	0	1	0	0	1	0	0	0	0	...	0	0	0	0
4	0	0	1	0	0	0	1	0	0	0	...	0	0	0	0

5 rows × 114 columns

Figure 4 - Converted numerical data

4.2 Split labels and features

Next step is to identify features and labels. Basically label is a value that we want to predict. In here 'Prediction value' is consider as a label. All other columns are consider as features. That means features are attributes which helps to predict final outcome. So that before start applying any learning algorithm we need to remove label from features.

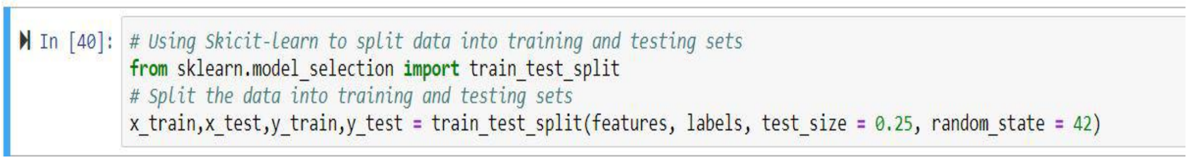


```
In [39]: # Use numpy to convert to arrays
         import numpy as np
         # Labels are the values we want to predict
         labels = np.array(features['Prediction Value_e'])
         # Remove the labels from the features
         # axis 1 refers to the columns
         features= features.drop('Prediction Value_e', axis = 1)
         # Saving feature names for later use
         feature_list = list(features.columns)
         # Convert to numpy array
         features = np.array(features)
```

Figure 5 - Split labels and features

4.3 Split training set and test set

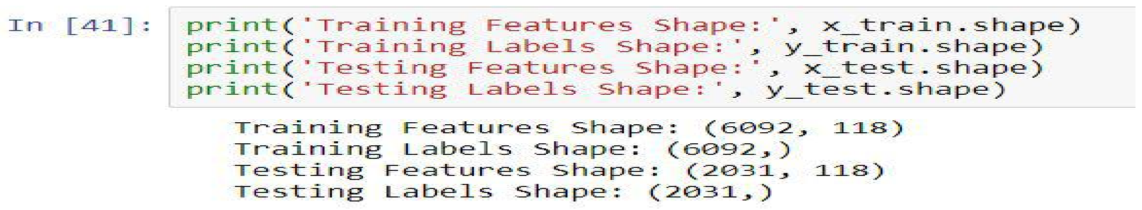
Next, we must divide data into two categories. Training set and Test set. Basically training set is used to build a machine learning model and test set is used to test performance in the machine learning model. We use sklearn library to divide training set and test set.



```
In [40]: # Using Skicit-learn to split data into training and testing sets
         from sklearn.model_selection import train_test_split
         # Split the data into training and testing sets
         x_train,x_test,y_train,y_test = train_test_split(features, labels, test_size = 0.25, random_state = 42)
```

Figure 6 - Split data into training set and test set

Mainly we are specify 0.25 as a test size. So that 25% of the data is consider as a test set. And 75% of the data is consider as a training set.



```
In [41]: print('Training Features Shape:', x_train.shape)
         print('Training Labels Shape:', y_train.shape)
         print('Testing Features Shape:', x_test.shape)
         print('Testing Labels Shape:', y_test.shape)

            Training Features Shape: (6092, 118)
            Training Labels Shape: (6092,)
            Testing Features Shape: (2031, 118)
            Testing Labels Shape: (2031,)
```

Figure 7 - Displaying shape of the features and labels

5 Algorithm

5.1 Introduction of Random Forest Classifier Algorithm

I use Random forest classification algorithm to predict whether given mushroom is edible or poisonous. Random Forest is a supervised learning algorithm and it is an ensemble classifier. It produces multiple decision trees. Basically this ensemble method increases the accuracy of the output that produced by the algorithm. Random forest algorithm uses two methods to combine outputs from multiple decision trees.

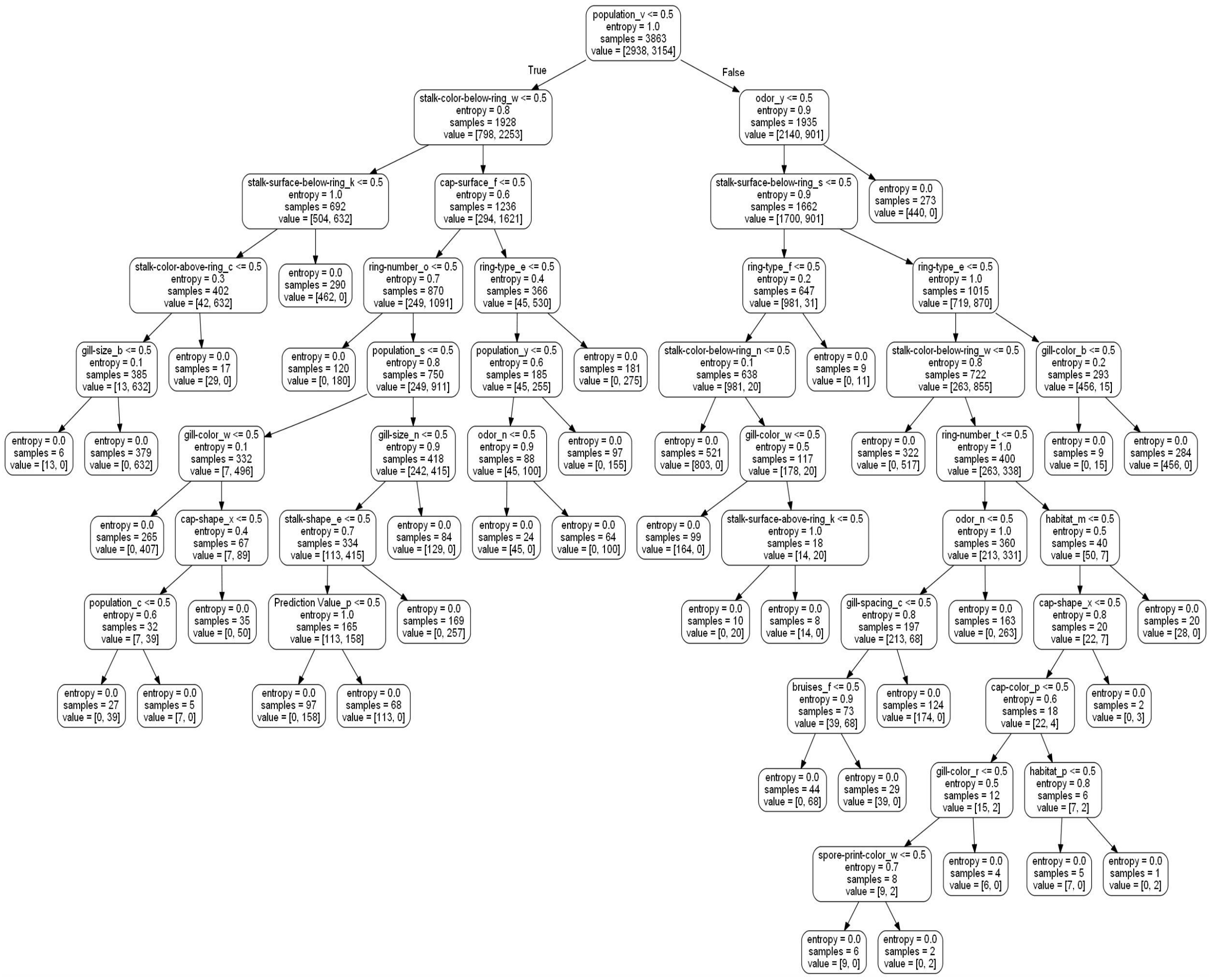
1. Bagging
2. Boosting

5.2 Decision Tree

Basically decision trees are trees which produce outputs based on certain decisions. It used for classification and prediction. There are two types of decision trees based on splitting attributes. It can be univariate as well as multivariate. Mainly decision trees used a top-down approach. To find out the best attribute for split decision trees use multiple methods available.

1. Information Gain
2. Gain Ratio
3. Gini Index

Basically information gain is calculated using the entropy. The attribute with the highest information gain is selected as a splitting attribute. In this scenario "population_v" consider as a splitting attribute. It has a higher entropy value (1.0). Following image shows a decision tree which helps to do our prediction.



population_v <= 0.5 entropy = 1.0 samples = 3863 value = [2938, 3154]
True
False
stalk-color-below-ring_w <= 0.5 entropy = 0.8 samples = 1928 value = [798, 2253]
odor_y <= 0.5 entropy = 0.9 samples = 1935 value = [2140, 901]
stalk-surface-below-ring_k <= 0.5 entropy = 1.0 samples = 692 value = [504, 632]
cap-surface_f <= 0.5 entropy = 0.6 samples = 1236 value = [294, 1621]
stalk-surface-below-ring_s <= 0.5 entropy = 0.9 samples = 1662 value = [1700, 901]
entropy = 0.0 samples = 273 value = [440, 0]
stalk-color-above-ring_c <= 0.5 entropy = 0.3 samples = 402 value = [42, 632]
entropy = 0.0 samples = 290 value = [462, 0]
ring-number_o <= 0.5 entropy = 0.7 samples = 870 value = [249, 1091]
ring-type_e <= 0.5 entropy = 0.4 samples = 366 value = [45, 530]
ring-type_f <= 0.5 entropy = 0.2 samples = 647 value = [981, 31]
ring-type_e <= 0.5 entropy = 1.0 samples = 1015 value = [719, 870]
gill-size_b <= 0.5 entropy = 0.1 samples = 385 value = [13, 632]
entropy = 0.0 samples = 17 value = [29, 0]
entropy = 0.0 samples = 120 value = [0, 180]
population_s <= 0.5 entropy = 0.8 samples = 750 value = [249, 911]
population_y <= 0.5 entropy = 0.6 samples = 185 value = [45, 255]
entropy = 0.0 samples = 181 value = [0, 275]
stalk-color-below-ring_n <= 0.5 entropy = 0.1 samples = 638 value = [981, 20]
entropy = 0.0 samples = 9 value = [0, 11]
stalk-color-below-ring_w <= 0.5 entropy = 0.8 samples = 722 value = [263, 855]
gill-color_b <= 0.5 entropy = 0.2 samples = 293 value = [456, 15]
entropy = 0.0 samples = 6 value = [13, 0]
entropy = 0.0 samples = 379 value = [0, 632]
gill-color_w <= 0.5 entropy = 0.1 samples = 332 value = [7, 496]
gill-size_n <= 0.5 entropy = 0.9 samples = 418 value = [242, 415]
odor_n <= 0.5 entropy = 0.9 samples = 88 value = [45, 100]
entropy = 0.0 samples = 97 value = [0, 155]
entropy = 0.0 samples = 521 value = [803, 0]
gill-color_w <= 0.5 entropy = 0.5 samples = 117 value = [178, 20]
entropy = 0.0 samples = 322 value = [0, 517]
ring-number_t <= 0.5 entropy = 1.0 samples = 400 value = [263, 338]
entropy = 0.0 samples = 9 value = [0, 15]
entropy = 0.0 samples = 284 value = [456, 0]
entropy = 0.0 samples = 265 value = [0, 407]
cap-shape_x <= 0.5 entropy = 0.4 samples = 67 value = [7, 89]
stalk-shape_e <= 0.5 entropy = 0.7 samples = 334 value = [113, 415]
entropy = 0.0 samples = 84 value = [129, 0]
entropy = 0.0 samples = 24 value = [45, 0]
entropy = 0.0 samples = 64 value = [0, 100]
entropy = 0.0 samples = 99 value = [164, 0]
stalk-surface-above-ring_k <= 0.5 entropy = 1.0 samples = 18 value = [14, 20]
odor_n <= 0.5 entropy = 1.0 samples = 360 value = [213, 331]
habitat_m <= 0.5 entropy = 0.5 samples = 40 value = [50, 7]
population_c <= 0.5 entropy = 0.6 samples = 32 value = [7, 39]
entropy = 0.0 samples = 35 value = [0, 50]
Prediction Value_p <= 0.5 entropy = 1.0 samples = 165 value = [113, 158]
entropy = 0.0 samples = 169 value = [0, 257]
entropy = 0.0 samples = 10 value = [0, 20]
entropy = 0.0 samples = 8 value = [14, 0]
gill-spacing_c <= 0.5 entropy = 0.8 samples = 197 value = [213, 68]
entropy = 0.0 samples = 163 value = [0, 263]
cap-shape_x <= 0.5 entropy = 0.8 samples = 20 value = [22, 7]
entropy = 0.0 samples = 20 value = [28, 0]
entropy = 0.0 samples = 27 value = [0, 39]
entropy = 0.0 samples = 5 value = [7, 0]
entropy = 0.0 samples = 97 value = [0, 158]
entropy = 0.0 samples = 68 value = [113, 0]
bruises_f <= 0.5 entropy = 0.9 samples = 73 value = [39, 68]
entropy = 0.0 samples = 124 value = [174, 0]
cap-color_p <= 0.5 entropy = 0.6 samples = 18 value = [22, 4]
entropy = 0.0 samples = 2 value = [0, 3]
entropy = 0.0 samples = 44 value = [0, 68]
entropy = 0.0 samples = 29 value = [39, 0]
gill-color_r <= 0.5 entropy = 0.5 samples = 12 value = [15, 2]
habitat_p <= 0.5 entropy = 0.8 samples = 6 value = [7, 2]
spore-print-color_w <= 0.5 entropy = 0.7 samples = 8 value = [9, 2]
entropy = 0.0 samples = 4 value = [6, 0]
entropy = 0.0 samples = 5 value = [7, 0]
entropy = 0.0 samples = 1 value = [0, 2]
entropy = 0.0 samples = 6 value = [9, 0]
entropy = 0.0 samples = 2 value = [0, 2]

Figure 8 - Decision tree which helps to predict whether given mushroom is edible or poisonous

Decision trees use different techniques to reduce the size of an decision tree. Basically their are two approaches.

1. Pre-pruning
2. Post-pruning

Decision tree pruning improves accuracy of an algorithm. And after pruning tree becomes smaller and less complex. It helps to avoid the problem like overfitting. Their are some advantages in decision trees.

1. High accuracy
2. Can handle high dimensional data
3. Learning and classification steps of decision tree are simple and fast.

And also their are some disadvantages as well.

1. Decision trees are highly depend on training data
2. Use greedy approach and locally optimal

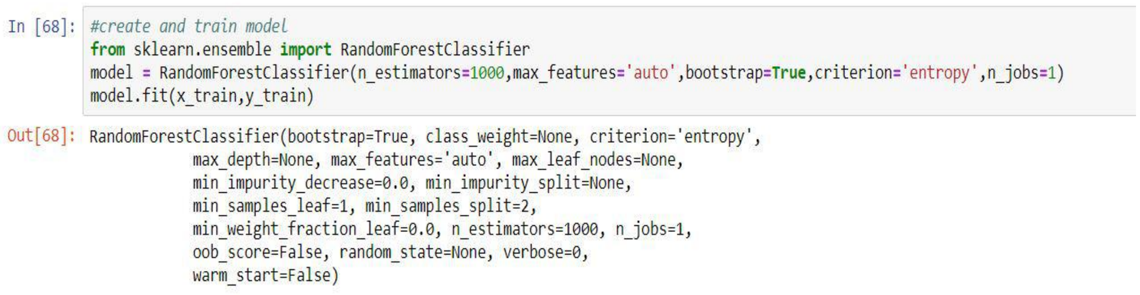
6 Implementation

6.1 Random Forest Classification Algorithm

Following are steps which are useful to create random forest classification model.

Step 1 - Pick at random k data points from training set
Step 2 - Build the decision tree associated to these k data points
Step 3 - Choose the number Ntree of trees you want to build and repeat step 1 & 2
Step 4 - For a new data point , make each one of your Ntree trees predict the category to which the data points belongs , and assign the new data point to the category that wins the majority vote.

Following code shows the implementation of random forest classification algorithm.



```
In [68]: #create and train model
         from sklearn.ensemble import RandomForestClassifier
         model = RandomForestClassifier(n_estimators=1000,max_features='auto',bootstrap=True,criterion='entropy',n_jobs=1)
         model.fit(x_train,y_train)

Out[68]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

Figure 9 - Create and train the Random Forest Classification model

As I mention in the above code I use “RandomForestClassifier” class in the sklearn.ensemble package to build up a random forest classification model. After that I created an object of a “RandomForestClassifier” class. I provided different parameters to that object.

n_estimators - Mainly this is a number of trees use in this classification. I use 1000 trees.

Criterion - This is used to split decision tree. I use entropy criteria for splitting. Basically entropy criteria is used for information gain. Their is another value that we can provide. That is "gini". It is used for gini impurity.
bootstrap - I use bootstrap method to build a random forest classifier model. So that bootstrap is "true".

max_features - max_features are the features that we consider when looking for a best split.

n_jobs - This is the number of jobs which runs in parallel for both fit and predict. I set it to 1. That means only 1 job runs in parallel.

And also their are some other parameters which are set by default. Parameters like max_depth , max_leaf_nodes , min_samples_leaf , class_weight , min_impurity_split and so on.

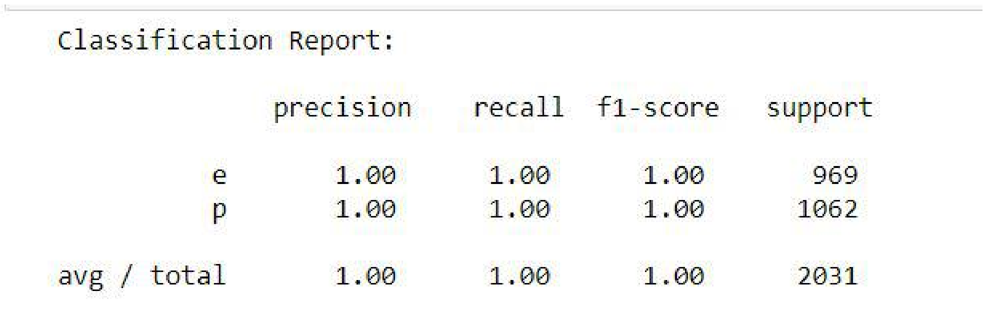
After I created an object of a "RandomForestClassifier" class I fit training sets of x and y to that created model.

7 Test Results

We then use previously created model to do our prediction. After start doing this prediction their are several metrics which came as outputs.

7.1 Classification Report

Basically classification report is a report which shows the main classification metrics like precision , recall , f1-score and support. Following figure sows the classification report of a prediction.



```
Classification Report:

             precision    recall  f1-score   support

          e       1.00      1.00      1.00       969
          p       1.00      1.00      1.00      1062

avg / total       1.00      1.00      1.00      2031
```

Figure 10 - classification report

Basically in our case classification report contains two target values. "e" and "p". That means edible and poisonous. According to classification report their are 969 edible mushrooms and 1062 poisonous mushrooms.

7.2 Confusion Matrix

Next metric is going to be a confusion matrix. Basically confusion matrix is a table which contains summary of prediction results.

```
Text(90.26,0.5,'Predicted value')
```

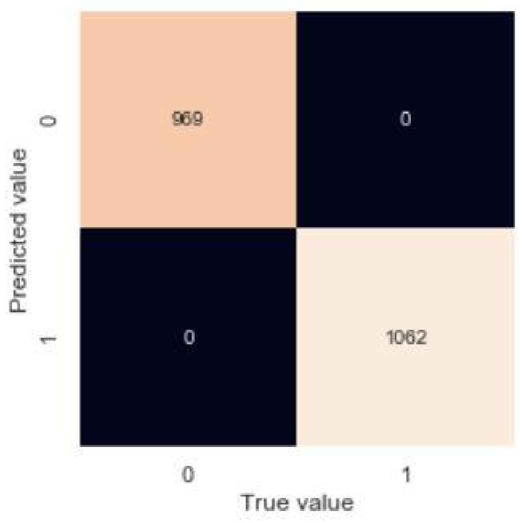


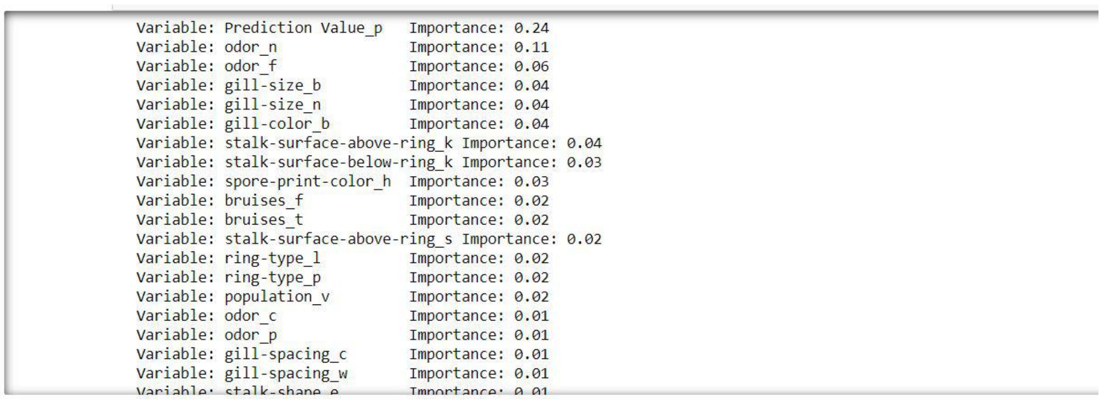
Figure 11 - Confusion Matrix

7.3 Decision Tree

Figure 8 shows a decision tree. This decision tree saved as a png image by using export_graphviz class.

7.4 Feature importance

If you consider about feature importance you can identify attributes which are most useful to predict results. Then by looking at the feature importance you can remove some attributes which are not help to predict the result from features. By removing some attributes from a feature list you can improve accuracy of your algorithm as well.



```
Variable: Prediction Value_p   Importance: 0.24
Variable: odor_n               Importance: 0.11
Variable: odor_f               Importance: 0.06
Variable: gill-size_b          Importance: 0.04
Variable: gill-size_n          Importance: 0.04
Variable: gill-color_b         Importance: 0.04
Variable: stalk-surface-above-ring_k Importance: 0.04
Variable: stalk-surface-below-ring_k Importance: 0.03
Variable: spore-print-color_h  Importance: 0.03
Variable: bruises_f            Importance: 0.02
Variable: bruises_t            Importance: 0.02
Variable: stalk-surface-above-ring_s Importance: 0.02
Variable: ring-type_l          Importance: 0.02
Variable: ring-type_p          Importance: 0.02
Variable: population_v         Importance: 0.02
Variable: odor_c               Importance: 0.01
Variable: odor_p               Importance: 0.01
Variable: gill-spacing_c       Importance: 0.01
Variable: gill-spacing_w       Importance: 0.01
Variable: stalk-shape_e        Importance: 0.01
```

Figure 12 - Feature Importance Table

```
Text(0.5,1,'Variable importance')
```

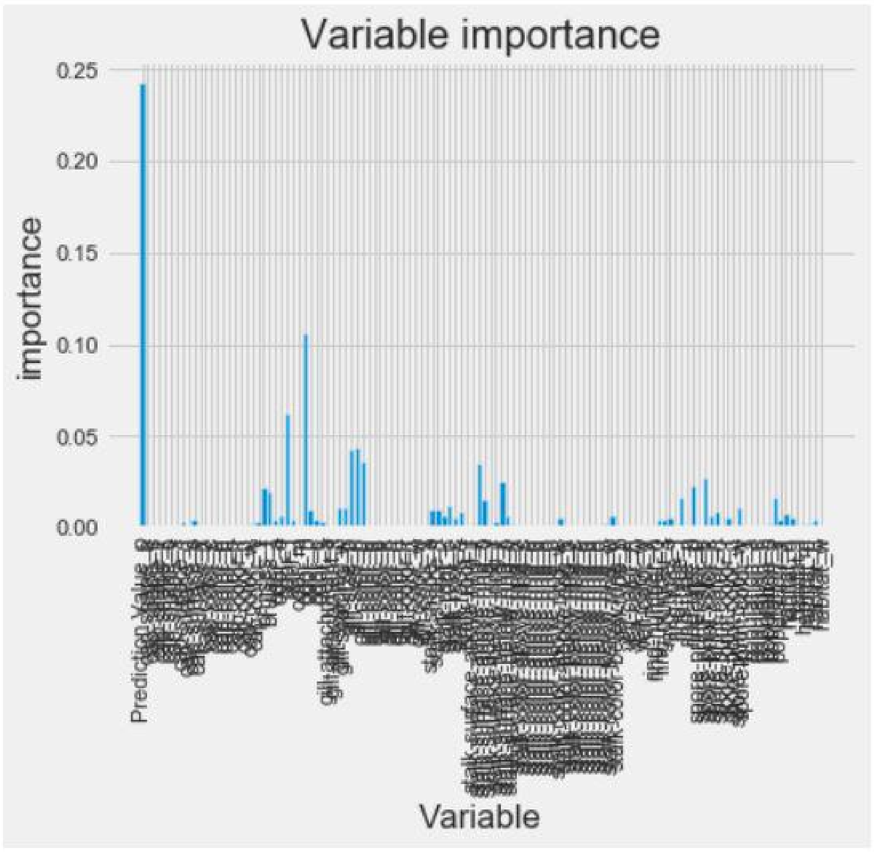


Figure 13 - Feature Importance graph

8 Evaluation

8.1 Future works

- Training this dataset with different training set and test set.
- In future we train this dataset with some other different machine learning algorithms. Then after we can improve accuracy of our algorithm.
- We will try different dataset with different features to do prediction.
- We will remove some unimportant features by looking at the feature importance and do classification again. After that we will improve accuracy of our algorithm.
- We will use other ensemble methods available to improve accuracy. Such as Bagging and Boosting method.

9 Discussions

9.1 Bagging

Basically Bagging is an ensemble method. We call bagging as a bootstrap aggregation. Bagging has a technique to decrease the variance of the final prediction and it helps to decrease model overfitting. Random forest uses this Bagging technique with multiple decision trees. And finally it predicts the output as a final prediction.

9.2 Boosting

Boosting is also used as an ensemble method. By using boosting you can improve accuracy as well. Mainly in boosting their are tree types.

1. AdaBoost(Adaptive Boosting)
2. Gradient Boosting
3. XGBoost(eXtreme Gradient Boosting)

You can use XGBoost as a gradient boosting algorithm with this scenario. XGBoost is much more better than other boosting algorithms. XGBoost gives model high performance and its execution speed is much more better than other methods with larger dataset.

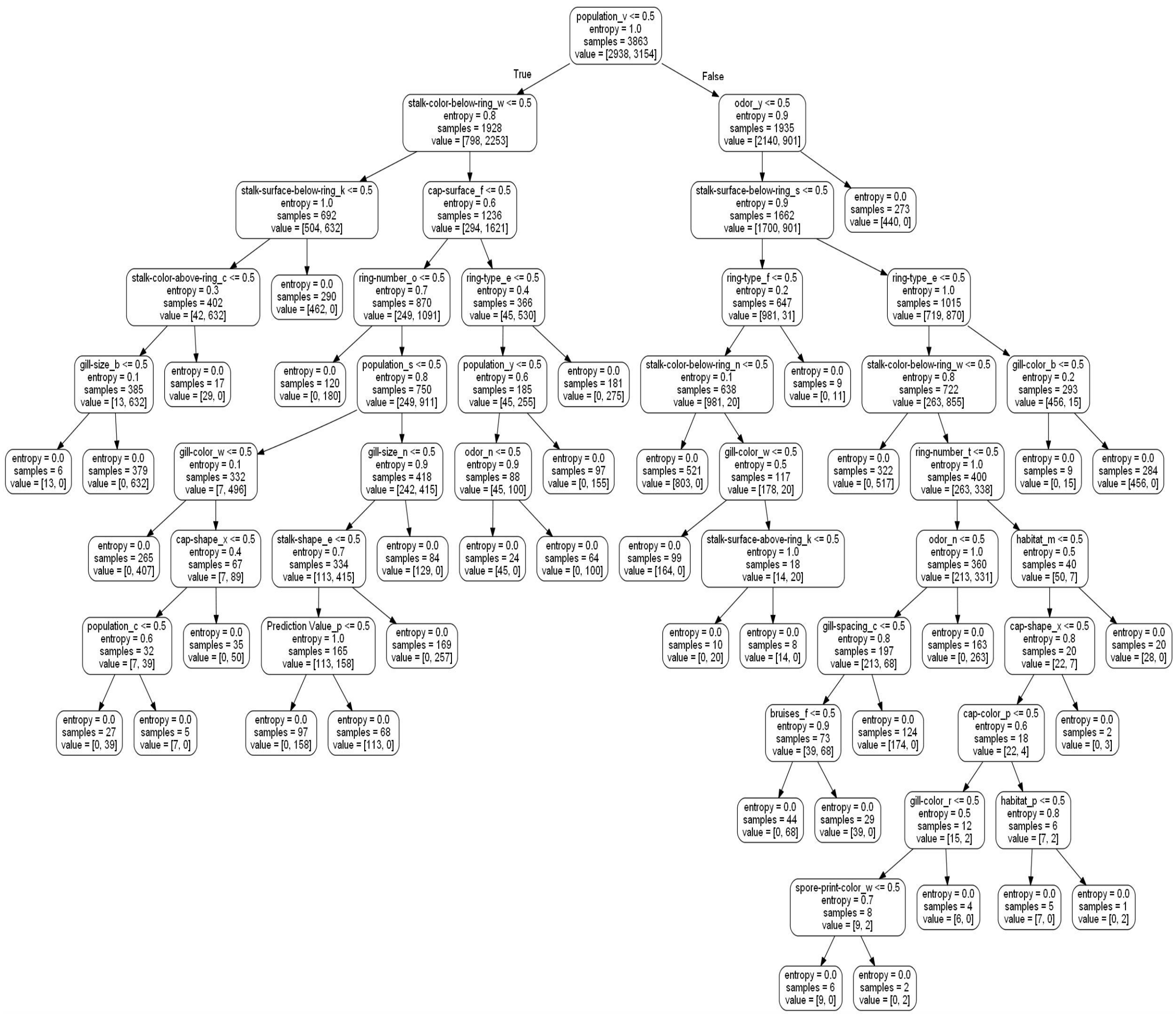
10 Appendix

10.1 Code

```
# Required Python libraries
import pandas as pd
import numpy as np
import os
from sklearn.tree import export_graphviz
import pydot
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
# Read in data and display first 5 rows
features = pd.read_csv('agaricus-lepiota.csv')
features.columns=['PredictionValue','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-b
elow-ring','stalk-color-above-ring',
'stalk-color-below-ring','veil-type','veil-color','ring-number','ring-type','spore-print-col
or','population', 'habitat']
features.head(5)
print('The shape of our features is:', features.shape)
# Descriptive statistics for each column
features.describe()
# One-hot encode the data using pandas get_dummies
features = pd.get_dummies(features)
# Display the first 5 rows of the last 12 columns
features.iloc[:,5:].head(5)
# Use numpy to convert to arrays
# Labels are the values we want to predict
labels = np.array(features['Prediction Value_e'])
# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('Prediction Value_e', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
# Using Skicit-learn to split data into training and testing sets
# Split the data into training and testing sets
x_train,x_test,y_train,y_test = train_test_split(features, labels, test_size = 0.25,
random_state = 42)
```

```
print('Training Features Shape:', x_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', x_test.shape)
print('Testing Labels Shape:', y_test.shape)
#create and train model
model=RandomForestClassifier(n_estimators=1000,max_features='auto',bootstrap=Tr
ue,criterion='entropy',n_jobs=1)
model.fit(x_train,y_train)
#predict test values
y_pred = model.predict(x_test)
#Print classification Report based on Predicted values
print("ClassificationReport:\n\n",metrics.classification_report(y_pred,y_test,target_na
mes=["e","p"]))
#print the confusion metrix based on the test values
sns.set()
get_ipython().run_line_magic('matplotlib','inline')
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat.T,square=True,annot=True,fmt='d',cbar=False)
plt.xlabel('True value')
plt.ylabel('Predicted value')
#Print tree as a image
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)Graphviz2.38/bin/'
tree = model.estimators_[1]
export_graphviz(tree,out_file='tree.dot',feature_names=feature_list,rounded=True,pre
cision=1)
(graph,) = pydot.graph_from_dot_file('tree.dot')
graph.write_png('tree.png')
#Output feature importance
importences = list(model.feature_importances_)
feature_importances = [(features,round(importences,2)) for features, importences in
zip(feature_list,importences)]
feature_importances = sorted(feature_importances,key=lambda x:x[1],reverse=True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in
feature_importances];
#feature importance ploted
get_ipython().run_line_magic('matplotlib','inline')
plt.style.use('fivethirtyeight')
x_values = list(range(len(importences)))
plt.bar(x_values,importences,orientation='vertical')
plt.xticks(x_values,feature_list,rotation='vertical')
plt.ylabel('importance')
plt.xlabel('Variable')
plt.title('Variable importance')
#print accuracy
print("Accuracy of an algorithm :",metrics.accuracy_score(y_test, y_pred)*100)
```

10.2 Decision Tree


population_v <= 0.5 entropy = 1.0 samples = 3863 value = [2938, 3154]
True
False
stalk-color-below-ring_w <= 0.5 entropy = 0.8 samples = 1928 value = [798, 2253]
odor_y <= 0.5 entropy = 0.9 samples = 1935 value = [2140, 901]
stalk-surface-below-ring_k <= 0.5 entropy = 1.0 samples = 692 value = [504, 632]
cap-surface_f <= 0.5 entropy = 0.6 samples = 1236 value = [294, 1621]
stalk-surface-below-ring_s <= 0.5 entropy = 0.9 samples = 1662 value = [1700, 901]
entropy = 0.0 samples = 273 value = [440, 0]
stalk-color-above-ring_c <= 0.5 entropy = 0.3 samples = 402 value = [42, 632]
entropy = 0.0 samples = 290 value = [462, 0]
ring-number_o <= 0.5 entropy = 0.7 samples = 870 value = [249, 1091]
ring-type_e <= 0.5 entropy = 0.4 samples = 366 value = [45, 530]
ring-type_f <= 0.5 entropy = 0.2 samples = 647 value = [981, 31]
ring-type_e <= 0.5 entropy = 1.0 samples = 1015 value = [719, 870]
gill-size_b <= 0.5 entropy = 0.1 samples = 385 value = [13, 632]
entropy = 0.0 samples = 17 value = [29, 0]
entropy = 0.0 samples = 120 value = [0, 180]
population_s <= 0.5 entropy = 0.8 samples = 750 value = [249, 911]
population_y <= 0.5 entropy = 0.6 samples = 185 value = [45, 255]
entropy = 0.0 samples = 181 value = [0, 275]
stalk-color-below-ring_n <= 0.5 entropy = 0.1 samples = 638 value = [981, 20]
entropy = 0.0 samples = 9 value = [0, 11]
stalk-color-below-ring_w <= 0.5 entropy = 0.8 samples = 722 value = [263, 855]
gill-color_b <= 0.5 entropy = 0.2 samples = 293 value = [456, 15]
entropy = 0.0 samples = 6 value = [13, 0]
entropy = 0.0 samples = 379 value = [0, 632]
gill-color_w <= 0.5 entropy = 0.1 samples = 332 value = [7, 496]
gill-size_n <= 0.5 entropy = 0.9 samples = 418 value = [242, 415]
odor_n <= 0.5 entropy = 0.9 samples = 88 value = [45, 100]
entropy = 0.0 samples = 97 value = [0, 155]
entropy = 0.0 samples = 521 value = [803, 0]
gill-color_w <= 0.5 entropy = 0.5 samples = 117 value = [178, 20]
entropy = 0.0 samples = 322 value = [0, 517]
ring-number_t <= 0.5 entropy = 1.0 samples = 400 value = [263, 338]
entropy = 0.0 samples = 9 value = [0, 15]
entropy = 0.0 samples = 284 value = [456, 0]
entropy = 0.0 samples = 265 value = [0, 407]
cap-shape_x <= 0.5 entropy = 0.4 samples = 67 value = [7, 89]
stalk-shape_e <= 0.5 entropy = 0.7 samples = 334 value = [113, 415]
entropy = 0.0 samples = 84 value = [129, 0]
entropy = 0.0 samples = 24 value = [45, 0]
entropy = 0.0 samples = 64 value = [0, 100]
entropy = 0.0 samples = 99 value = [164, 0]
stalk-surface-above-ring_k <= 0.5 entropy = 1.0 samples = 18 value = [14, 20]
odor_n <= 0.5 entropy = 1.0 samples = 360 value = [213, 331]
habitat_m <= 0.5 entropy = 0.5 samples = 40 value = [50, 7]
population_c <= 0.5 entropy = 0.6 samples = 32 value = [7, 39]
entropy = 0.0 samples = 35 value = [0, 50]
Prediction Value_p <= 0.5 entropy = 1.0 samples = 165 value = [113, 158]
entropy = 0.0 samples = 169 value = [0, 257]
entropy = 0.0 samples = 10 value = [0, 20]
entropy = 0.0 samples = 8 value = [14, 0]
gill-spacing_c <= 0.5 entropy = 0.8 samples = 197 value = [213, 68]
entropy = 0.0 samples = 163 value = [0, 263]
cap-shape_x <= 0.5 entropy = 0.8 samples = 20 value = [22, 7]
entropy = 0.0 samples = 20 value = [28, 0]
entropy = 0.0 samples = 27 value = [0, 39]
entropy = 0.0 samples = 5 value = [7, 0]
entropy = 0.0 samples = 97 value = [0, 158]
entropy = 0.0 samples = 68 value = [113, 0]
bruises_f <= 0.5 entropy = 0.9 samples = 73 value = [39, 68]
entropy = 0.0 samples = 124 value = [174, 0]
cap-color_p <= 0.5 entropy = 0.6 samples = 18 value = [22, 4]
entropy = 0.0 samples = 2 value = [0, 3]
entropy = 0.0 samples = 44 value = [0, 68]
entropy = 0.0 samples = 29 value = [39, 0]
gill-color_r <= 0.5 entropy = 0.5 samples = 12 value = [15, 2]
habitat_p <= 0.5 entropy = 0.8 samples = 6 value = [7, 2]
spore-print-color_w <= 0.5 entropy = 0.7 samples = 8 value = [9, 2]
entropy = 0.0 samples = 4 value = [6, 0]
entropy = 0.0 samples = 5 value = [7, 0]
entropy = 0.0 samples = 1 value = [0, 2]
entropy = 0.0 samples = 6 value = [9, 0]
entropy = 0.0 samples = 2 value = [0, 2]