Final Project:

Mushroom Dataset Analysis

Name: Yawen Han
Course: CSE 5243
Introduction to Data Mining
Instructor: Jason Van Hulse
TA: Xiaoqi Wang

Date:
April 18, 2018

Table of Contents

Introduction

This project focuses on the data mining of the Mushroom dataset provided from UCI Machine Learning Repository. In order to identify the best performed modeling approach with the Mushroom dataset, works are carried out in the discovery cycle including preliminary data exploration (explore data and verify data quality), data transformation (outliers, missing value, feature selection), modeling (select modelling technique, generate test design, build and assess the model) and model evaluation.

Background

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. 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. These latter classes are 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.

This project will investigate the data mining of physical characteristics data on the mushroom in order to create one or more classification models which are capable of accurately identifying if the certain mushroom is poisonous or edible. The data contains mining information such as cap-shape, cap-surface and cap-color, describing mushrooms in terms of physical characteristics. The classification attribute in the mushroom dataset is expressed as “e” for “edible” and “p” for “poisonous”, indicating the edibility of a mushroom.

Preliminary Data Analysis

The Mushroom dataset contains 8124 records with 23 attributes. The “class” attribute is a categorical binomial class and expressed as “e” for “edible and “p” for “poisonous”, and is used as the class label. The other 22 attributes consisting of 4 binomial attributes and 18 polynomials attributes. Part of the dataset is shown below in Table 1, and more attribute information is summarized in Table 2.

	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

Table 1. Part of the Dataset

Table 2 lists some column metrics of Table 1, and provides some basic information of each attribute. The binomial attribute “class” describes the edibility of the mushroom. The “cap” series attributes containing “cap-shape”, “cap-surface”, and “cap-color”, gives the cap information of the mushroom. The binomial attribute “bruises” describes the if the mushroom has bruises or not on its surface. The “order” attribute gives an idea of what the mushroom smells like. The “gill” series attributes containing “gill-attachment”, “gill-spacing”, “gill-size” and “gill-color”, gives the gill information of the mushroom. Similarly, the “stalk”, “veil”, “ring” and “spore” series attributes gives the stalk, veil, ring and spore information of the mushroom. At last, the “population” and “habitat” attributes describe how and where the mushroom are grouped together.

Attribute	Type	Values*1	Unique	Missing	Null
class	String	e=edible(52%), p=poisonous(48%)	2	0	0%
cap-shape	String	x=convex(45%), f=flat(39%), k=knobbed(10%), b=bell(6%), s=sunken(0%), c=conical(0%)	6	0	0%
cap-surface	String	y=scaly(40%), s=smooth(30.8%), f=fibrous(29%), g=grooves(0.2%)	4	0	0%
cap-color	String	n=brown(28%), g=gray(23%), e=red(18%), y=yellow(13%), w=white(13%) ,b=buff(0%), c=cinnamon(0%), r=green(0%), p=pink(0%), u=purple(0%)	10	0	0%
bruises	String	f=no(58%), t=bruises(42%)	2	0	0%
order	String	n=none(43%), f=foul(27%), y=fishy(7%), s=spicy (7%), a=almond(5%), l=anise(0%), c=creosote(0%) m=musty(0%), s=spicy(0%)	9	0	0%
gill-attachment	String	f=free(97%), a=attached(3%)	2	0	0%
gill-spacing	String	c=close(84%), w=crowded(16%)	2	0	0%
gill-size	String	b=broad(69%), n=narrow(31%)	2	0	0%
gill-color	String	b=buff(21%), p=pink(18%), w=white(15%), n=brown(13%), g=gray(9%), k=black(0%), h=chocolate(0%), r=green(0%), o=orange(0%), u=purple(0%), e=red(0%), y=yellow(0%)	12	0	0%
stalk-shape	String	t=tapering(57%), e=enlarging(43%)	2	0	0%
stalk-root	String	b=bulbous(46%), ?=missing(31%), e=equal(14%), c=club(7%), r=rooted(2%)	5	2480	0%
stalk-surface-above-ring	String	s=smooth(64%), k=silky(29%), f=fibrous(7%), y=scaly(0%)	4	0	0%
stalk-surface-below-ring	String	s=smooth(61%), k=silky(28%), f=fibrous(7%), y=scaly(3%)	4	0	0%
stalk-color-above-ring	String	w=white(55%), p=oink(23%), g=gray(7%), n=brown(6%), b=buff(5%), o=orange(0%), c=cinnamon(0%), e=red(0%), y=yellow(0%)	9	0	0%
stalk-color-below-ring	String	w=white(54%), p=pink(23%), g=grey(7%), n=brown(6%), b=buff(5%), o=orange(0%), c=cinnamon(0%), e=red(0%), y=yellow(0%)	9	0	0%
veil-type	String	p=partial(100%)	1	0	0%
veil-color	String	w=white(98%), n=brown(1%), o=orange(1%), y=yellow(0%)	4	0	0%
ring-number	String	o=one(92%), t=two(7%), n=none(0%)	3	0	0%
ring-type	String	p=pendant(49%), e=evanescent(34%), l=large(16%), f=flaring(1%), n=none(0%)	5	0	0%
spore-print-color	String	w=white(29%), n=brown(24%), k=black(23%), h=chocolate(20%), r=green(1%), b=buff(0%), o=orange(0%), u=purple(0%), y=yellow(0%)	9	0	0%
population	String	v=several(50%), y=solitary(21%), s=scattered(15%), n=numerous(5%), a=abundant(5%), c=clustered(0%)	6	0	0%
habitat	String	d=woods(39%), g=grasses(26%), p=paths(14%), l=leaves(10%), u=urban(5%), m=meadows(0%), w=waste(0%)	7	0	0%

Note*1: The percentage of each attribute value is rounded to the nearest integer, thus 0% means a very small percentage instead of "no value".

Table 2. Attributes Information Summary

As Table 2 shows, all the attributes have the "String" attribute type, indicating that all of them are categorical attributes. Also, as all the attributes have no "Null" value, no more effort needed later to deal with the null value. However, it shows that the "stalk-root" attribute has missing values with a

total number of 2480, which occupies more than 30% of the total data. Therefore, further data transformation is needed to handle this problem.
To have a clearer and better understanding of each attribute, the distribution of each feature is plotted first. Figure 1 shows the number of value counts for each category feature.

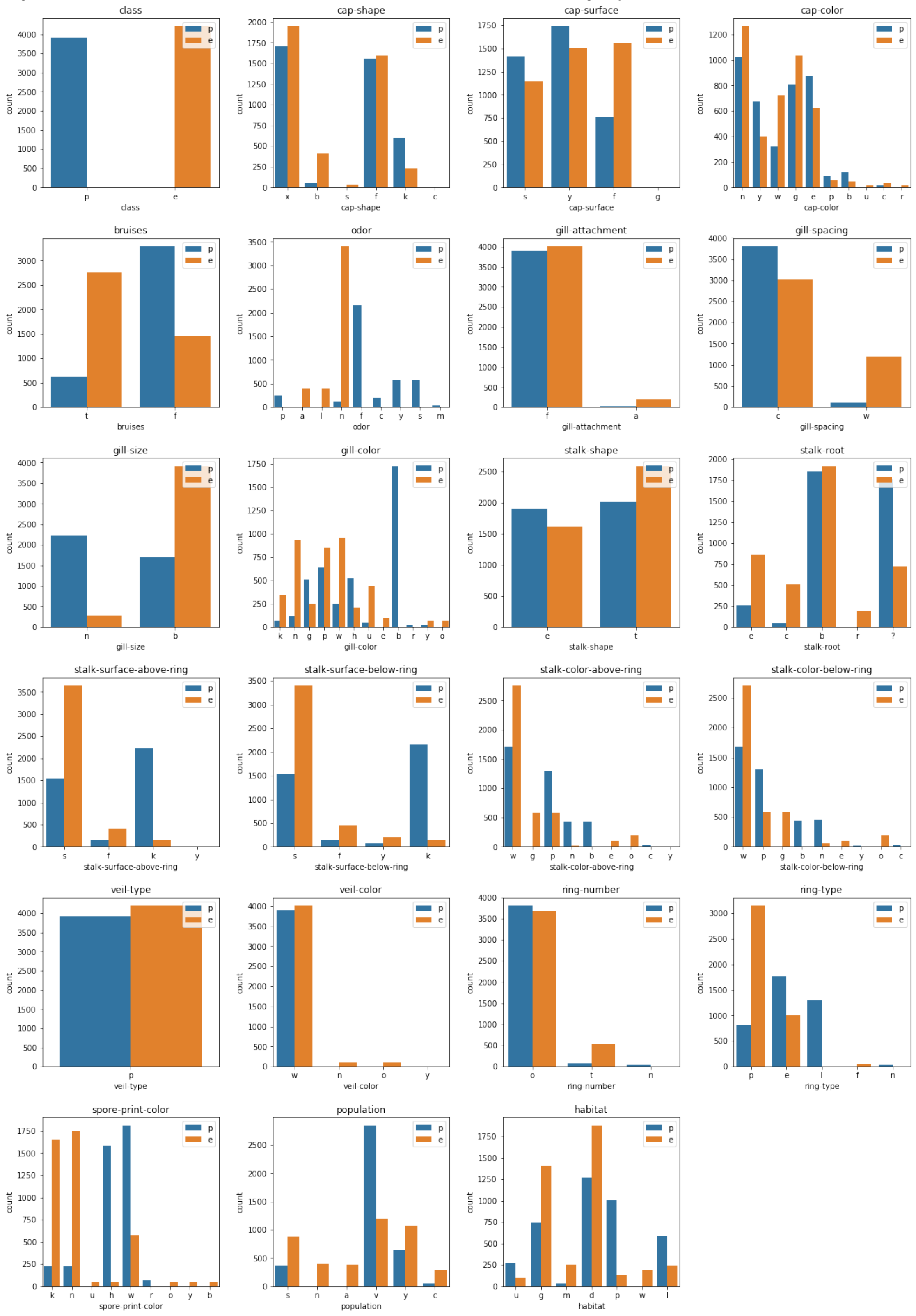


Figure 1. Value Count for Categorical Features

First, in the "class" value count diagram (row 1, column 1) in Figure 1, it shows that two classes "e" and "p" have a balanced distribution. This can also be reinforced from Table 2 with 52% mushrooms in the dataset are edible, and 48% left are poisonous. Therefore, in this project, it won't occur the imbalanced dataset problem that one class is ignored during the classification process.

Then, in the "stalk-root" value count diagram (row 3, column 4) in Figure 1, more than 30% of the values of stalk-root are missing values, marked as "?". In the diagram, the attribute value "b" has the maximum counts, and it's the only attribute value that has a similar edibility distribution as the missing value column. Therefore, the missing values will be replaced by the mode attribute value "b" in later data transformation section.

Next, in the "veil-type" value count diagram (row 5, column 1) in Figure 1, "p" is the only attribute value in "veil-type" attribute. In other words, the veil-type of the mushroom is always "p" no matter it is editable or poisonous, which means this attribute does not contribute to the classification of the dataset. Thus, the "veil-type" attribute column will be deleted in later data transformation section.

In addition, in the "ring-number" value count diagram (row 5, column 3) in Figure 1, the attribute value "o" means there is one ring on the mushroom, then "t" means two rings and "n" means no ring. Thus, the "ring-number" attribute can be expressed as a numerical attribute instead of categorical to set {"o","t","n"} as {1,2,0}, which is shown in Table 3 below.

"Ring-number" attribute value	"o"	"t"	"n"
Numerical expression	1	2	0

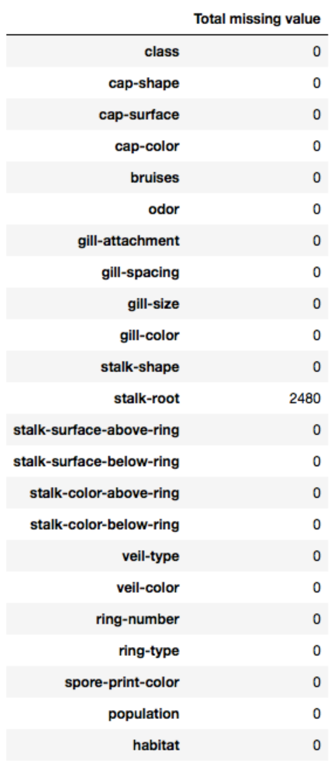
Table 3. Mapping between "Ring-number" Attribute Value *and Numerical Expressions*

Finally, as mentions before, all the attributes have the "String" attribute type, indicating that all of them are categorical attributes. After transforming the "ring-number" attribute from categorical to numerical, the outliers problem will be only considered for "ring-number" attribute (supposed that no outliers problem will be considered for categorical attributes in this project). However, as the transformed "ring-number" attribute only has three discrete values {1,2,0} instead of continuous values, there is still no outlier problem for "ring-number" attribute. Therefore, no outlier problem will be considered in the later data transformation section.

Data Transformation

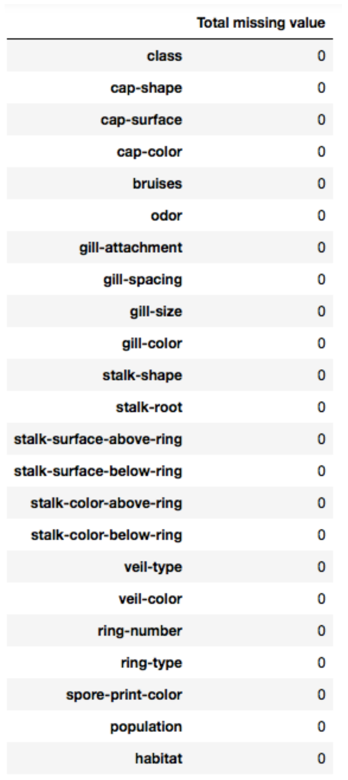
1) Missing value

For the "stalk-root" attribute, there are 2480 out of 8124 data points are missing values. That means more than 30% of the values of stalk-root are missing, so the method to delete all missing value will affect the accuracy of classification model. According to the data analysis in the last section, the missing values will be replaced by the mode attribute value "b", and the missing value information before and after data transformation is shown in Figure 2 below.



	Total missing value
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	2480
stalk-surface-above-ring	0
stalk-surface-below-ring	0
stalk-color-above-ring	0
stalk-color-below-ring	0
veil-type	0
veil-color	0
ring-number	0
ring-type	0
spore-print-color	0
population	0
habitat	0

(a) Missing values before data transformation



	Total missing value
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-type	0
veil-color	0
ring-number	0
ring-type	0
spore-print-color	0
population	0
habitat	0

(b) Missing values after data transformation

Figure 2. Missing value information before and after data transformation

2) Drop "veil-type" attribute

In Table 2, it shows that the "veil-type" attribute only has 1 unique value "p". As the veil-type of the mushroom is always "p" no matter it is editable or poisonous, this attribute does not contribute to the classification of the dataset. Thus, the unwanted "veil-type" attribute is dropped to decrease the dataset dimension. After dropping the veil-type" attribute, there is still 22 attributes left.

3) "ring-number" feature simplification

Transform the categorical values of "ring-number" attribute values {"o","t","n"} to numerical values {1,2,0} according to *Table 3*. The "ring-number" attribute information summary after data transformation is shown in *Table 4* below.

Attribute	Count	Values*1	Unique	Missing	Null	Top	Frequency
ring-number	8124	"1"(92%), "2"(7%), "0"(0%)	3	0	0%	1	7488

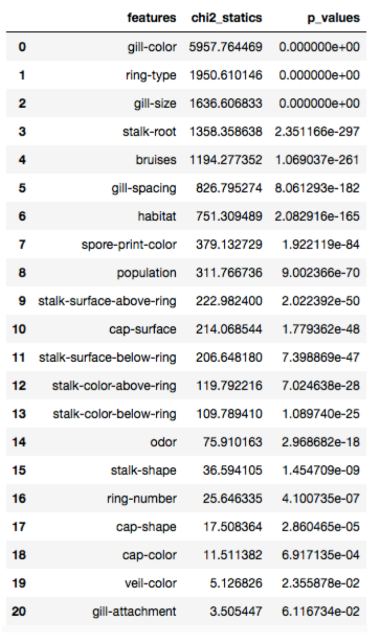
Note*1: The percentage of each attribute value is rounded to the nearest integer, thus 0% means a very small percentage instead of "no value".

Table 4. "ring-number" Attribute Information Summary after Data Transformation

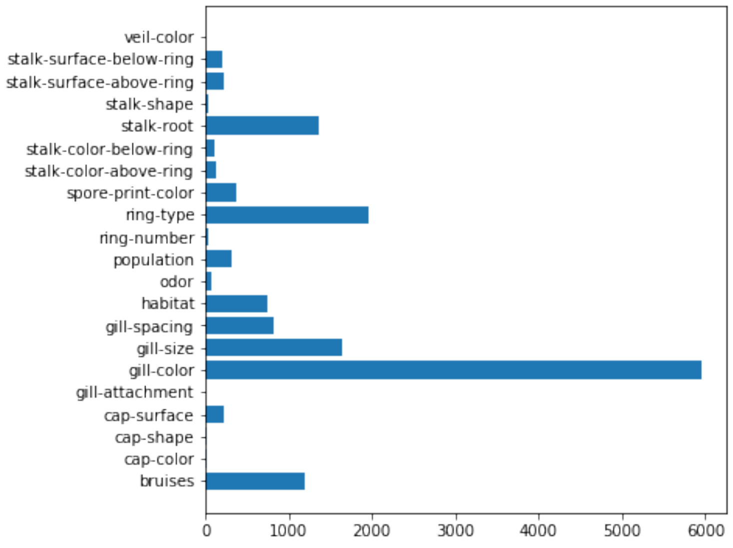
4) Feature subset selection

After the data transformation of step 1~3 above, there is still 21 attributes left ("class" attribute excluded). Check chi-square significance of 21 features to figure out the most useful features for later classification model building.

The chi-square test is a statistical test of independence to determine the dependency of two variables. It shares similarities with coefficient of determination, R^2. The chi-square test can easily deduce the application of chi-square technique in feature selection. By applying the chi-square test, chi-square statistics are calculated between every feature variable and the target "class" variable and observe the existence of a relationship between the variables and the target. If the target variable is independent of the feature variable, we can discard that feature variable. If they are dependent, the feature variable is very important.



	features	chi2_statics	p_values
0	gill-color	5957.764469	0.000000e+00
1	ring-type	1950.610146	0.000000e+00
2	gill-size	1636.606833	0.000000e+00
3	stalk-root	1358.358638	2.351166e-297
4	bruises	1194.277352	1.069037e-261
5	gill-spacing	826.795274	8.061293e-182
6	habitat	751.309489	2.082916e-165
7	spore-print-color	379.132729	1.922119e-84
8	population	311.766736	9.002366e-70
9	stalk-surface-above-ring	222.982400	2.022392e-50
10	cap-surface	214.068544	1.779362e-48
11	stalk-surface-below-ring	206.648180	7.398869e-47
12	stalk-color-above-ring	119.792216	7.024638e-28
13	stalk-color-below-ring	109.789410	1.089740e-25
14	odor	75.910163	2.968682e-18
15	stalk-shape	36.594105	1.454709e-09
16	ring-number	25.646335	4.100735e-07
17	cap-shape	17.508364	2.860465e-05
18	cap-color	11.511382	6.917135e-04
19	veil-color	5.126826	2.355878e-02
20	gill-attachment	3.505447	6.116734e-02



(a) Chi-square test statistics table *(b) Chi-square test statistics bar plot*

Figure 3. Chi-square Test Results Summary

Figure 3 above shows the chi-square test results for 21 attributes ("class" attribute excluded). In Figure 3(a), it shows that there are 5 features that have a chi-square statistic greater than 1000, and there are 6 feature that have a chi-square statistic greater than 500.

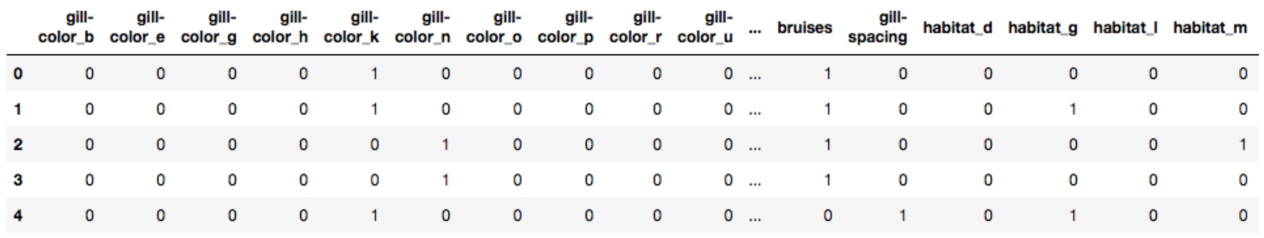
In Figure 3(b), it demonstrates which features are significantly distinct by two classes. In this project, 7 most distinct feature (chi2 static are more than 500) are used for next classification, including "gill-color", "ring-type", "gill-size", "stalk-root", "bruises", "gill-spacing" and "habitat".

5) Categorical Features Discretization

As the dataset has values in strings, the label encoding is performed on the data to transform all the unique values to integers. Two different categorical feature discretization methods are applied for the mushroom dataset: for features with two values, LabelBinarizer is used for label encoding; for features with multiple, OneHot is used for label encoding.

The "class", "gill-size", "gill-spacing", and "bruises" attribute only has two values, so they are label encoded with integer value {0,1} through LabelBinarizer method.

For other categorical features, they all have multiple values. If they are label encoded as an integer like "class" attribute, it will be biased for higher valued features when modelling. One possibility to convert multiple-values categorical features to features that can be used with scikit-learn estimators is to use a one-of-K or one-hot encoding, which is implemented in OneHotEncoder. This estimator transforms each categorical feature with m possible values into m binary features, with only one active.



	gill-color_b	gill-color_e	gill-color_g	gill-color_h	gill-color_k	gill-color_n	gill-color_o	gill-color_p	gill-color_r	gill-color_u	...	bruises	gill-spacing	habitat_d	habitat_g	habitat_l	habitat_m
0	0	0	0	0	1	0	0	0	0	0	...	1	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	...	1	0	0	1	0	0
2	0	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	1
3	0	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	...	0	1	0	1	0	0

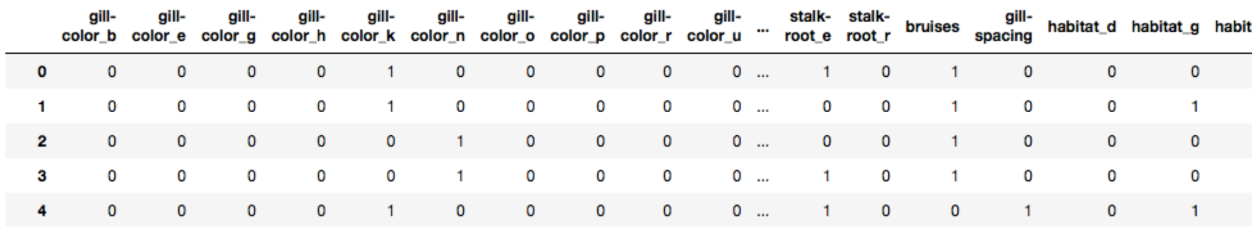
Table 5. Part of the Dataset after Categorical Feature Discretization

Table 5 above shows part of the dataset after applied the OneHot encoding method. This table contains 8124 records with 32 attributes, as each feature value in the previous dataset is transformed to be a new column with binary values.

To have a better understanding of the whole data transformation process, the results of data transformations are summarized below in *Table 6*, and first 5 rows of transformed dataset are shown in *Table 7*.

Number	Attribute/Operation	Transformation
1	"stalk-root" attribute	Replace missing value with the mode value "b"
2	"veil-type" attribute	Dropped
3	"ring-number" attribute	Transform categorical values {"o","t","n"} to numerical values {1,2,0}
4	feature subset selection	Keep 7 most distinct feature and drop others
5	categorical feature discretization	Features being label encoded

Table 6. Summarized Data Transformations



	gill-color_b	gill-color_e	gill-color_g	gill-color_h	gill-color_k	gill-color_n	gill-color_o	gill-color_p	gill-color_r	gill-color_u	...	stalk-root_e	stalk-root_r	bruises	gill-spacing	habitat_d	habitat_g	habit
0	0	0	0	0	1	0	0	0	0	0	...	1	0	1	0	0	0	
1	0	0	0	0	1	0	0	0	0	0	...	0	0	1	0	0	1	
2	0	0	0	0	0	1	0	0	0	0	...	0	0	1	0	0	0	
3	0	0	0	0	0	1	0	0	0	0	...	1	0	1	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	...	1	0	0	1	0	1	

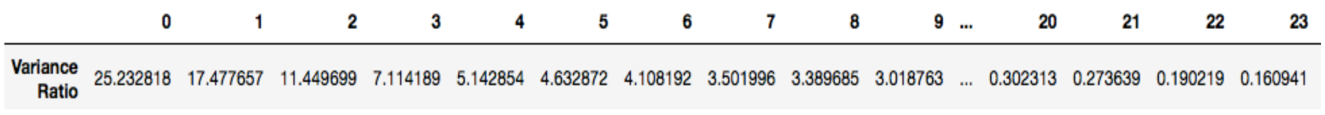
Table 7. First Five Rows in Transformed Dataset

Principal Component Analysis

After the categorical feature discretization in step 5, there are still 32 attributes in the dataset, and some of them are considered to be highly correlated as they come from the same original feature (For example, "gill-color_b", "gill-color_e", "gill-color_g" are considered to be correlated as they come from the same original feature "gill-color").

The simplest way to do dimensionality reduction is to just remove some of the features. However, this is problematic if the features dropped contain valuable diagnostic information about that mushroom. A slightly more sophisticated approach is to reduce the dimensionality of the dataset by only considering its principal components, or the combination of features that explains the most variance in the dataset. Using a technique called principal components analysis (or PCA), the dimensionality of a dataset can be reduced while preserving as much of its precious variance as possible.

In order to condense the information contained in a large number of variables into a smaller set of new composite dimensions, with a minimum loss of information, Principal Component Analysis (PCA) is used in this section as a dimensionality-reduction technique.



	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23
Variance Ratio	25.232818	17.477657	11.449699	7.114189	5.142854	4.632872	4.108192	3.501996	3.389685	3.018763	...	0.302313	0.273639	0.190219	0.160941

Table 8. Part of the Variance Ratio Percentage Table

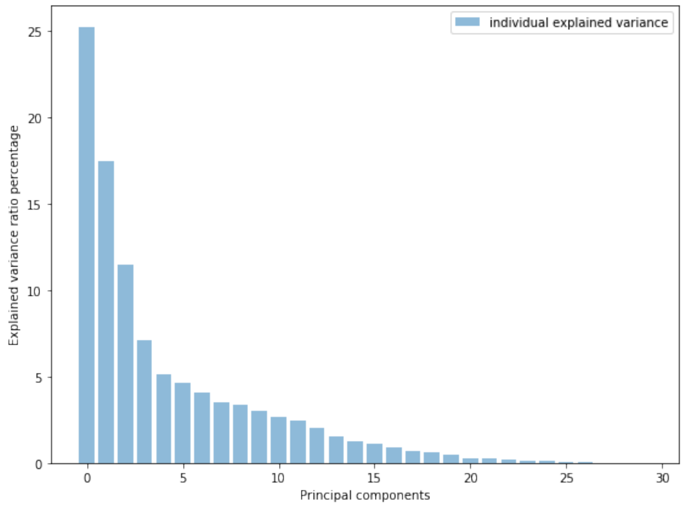


Figure 4. Variance Ratio Percentage Bar Chart

Table 8 above shows part of the variance ratio percentage in a descending order, and Figure 4 shows the same thing with a bar chart. It can be concluded from Table 8 and Figure 4 that the first 16 components retain more than 95% of the data. Therefore, PCA is performed by taking 16 components with maximum variance.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-0.44	0.50	-1.02	0.36	0.52	-0.59	-0.18	0.86	0.57	0.78	-0.27	-0.12	-0.49	-0.14	0.66	0.04
1	-0.89	0.94	-0.49	0.64	-0.02	-0.19	-0.01	0.39	-0.72	-0.13	-0.37	0.12	0.35	-0.39	0.63	0.15
2	-1.01	0.40	-0.66	0.78	-0.11	-0.94	0.03	-0.14	-0.52	-0.06	0.55	-0.07	0.55	0.23	-0.06	-0.48
3	-0.53	0.44	-1.08	0.29	0.38	-1.03	0.09	0.19	0.80	0.66	0.16	0.34	-0.54	0.12	-0.08	-0.30
4	0.69	1.74	-0.35	-0.53	-0.24	0.00	-0.28	0.14	-0.08	-0.06	-0.06	-0.43	-0.13	-0.33	0.60	0.19

Table 9. First 5 Rows of 16 Features Subset

Table 9 above shows the first 5 rows of the 16 features subset after applying the PCA. This transformed dataset will be tested in next section to check if the reduction in dimension affecting a lot to the classifiers' accuracy, and possibly be used to build classification models if passing the validation test.

Transformed Dataset Validation

The data transformations and dimension reduction outlined above are then applied to three classifiers in Software Weka to see if they affect a lot to the classifiers' accuracy. The three classifiers are kNN, Decision Tree and Naïve Bayes. The test results from Python were summarized in *Table 10* below.

Classifier		TP	FP	Precision	Recall	F-Measure	ROC Area
kNN	Unmodified	0.999	0.001	0.999	0.999	0.999	1.000
	Transformed before PCA	0.982	0.018	0.982	0.982	0.982	0.999
	Transformed after PCA	0.976	0.023	0.976	0.976	0.976	0.999
Decision Tree	Unmodified	1.000	0.000	1.000	1.000	1.000	1.000
	Transformed before PCA	1.000	0.000	1.000	1.000	1.000	1.000
	Transformed after PCA	1.000	0.000	1.000	1.000	1.000	1.000
Naïve Bayes	Unmodified	0.954	0.049	0.956	0.954	0.954	0.997
	Transformed before PCA	0.887	0.119	0.894	0.887	0.886	0.939
	Transformed after PCA	0.930	0.095	0.937	0.930	0.930	0.976

Table 10. Three Classifier Test Results for Selected Data Transformation

Three different scenarios of dataset are tested for each classification method, including the unmodified dataset (Table 1), the transformed dataset before PCA (Table 7), the transformed dataset after PCA (Table 9). Table 10 shows that both the transformed dataset before PCA and after PCA are lost somewhat accuracy and precision after the transformation, but the decrease is not significant and still keep the classification results in a high accuracy and precision. The applications of data transformation and dimensionality reduction do provide useful information without significantly impacting the performance of three testing classifiers, which confirms the earlier assertion during the preliminary data analysis that some attributes may not be significantly predictive.

For the Naïve Bayes classification method, the transformed dataset gets a better performance after PCA (Red statistics). In addition, it's interesting to find that the accuracy of Decision Tree keeps to be the best no matter which dataset being used (Yellow Shaded Area). These results indicate that the above approach to dimensionality reduction through PCA do not appear to impact the performance of classifier, and may improve the accuracy for some classifiers such as Naïve Bayes. In conclusion, it shows that classifiers can still provide a good performance with the transformed dataset after PCA. Therefore, the transformed dataset after PCA (Table 9) will be used to build in the model development process.

Model Development

In this section, five modeling approaches are applied to the transformed dataset using the Python, including 1) Logistic Regression, 2) Decision Tree, 3) KNN, 4)Artificial Neural Network, and 5) Support Vector Machine. For each approach, the parameters are experimented and tuned to improve its performance, and the performance statistics for the best experiment are also summarized for each approach.

1) Logistic Regression

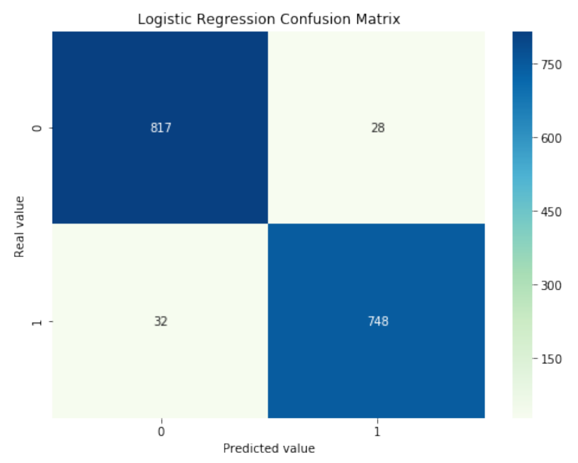
Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

Two parameters are discussed and experimented here, including a) penalty: (default: 'l2') - used to specify the norm used in the penalization. "l1" and "l2" are regularization parameters. They're used to avoid overfitting. Both "l1" and "l2" regularization prevents overfitting by shrinking (imposing a penalty) on the coefficients. "l1" is the first moment norm |x1-x2| (|w| for regularization case) that is simply the absolute dıstance between two points where "l2" is second moment norm corresponding to Euclidian Distance that is |x1-x2|^2 (|w|^2 for regularization case); and b) C: (default: 1.0) - inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization. The experiments results are shown in *Table 11* below:

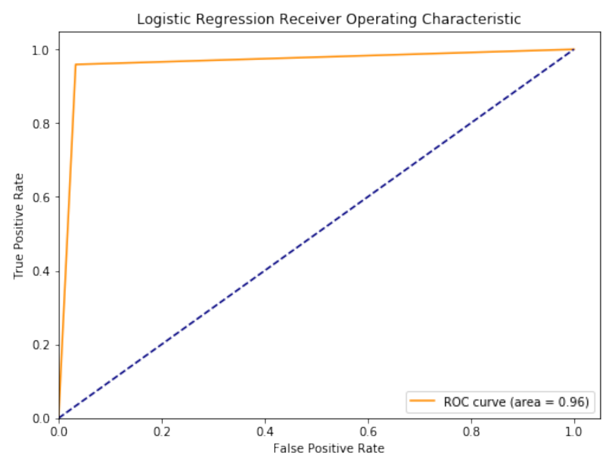
Exp#	penalty	C	Accuracy	F-measure	ROC Area
1(default)	"l2"	1.0	0.963	0.961	0.963
2	"l1"	1.0	0.963	0.961	0.963
3	"l2"	0.1	0.956	0.953	0.955
4	"l1"	0.1	0.963	0.961	0.963

Table 11. Logistic Regression Experiments Results

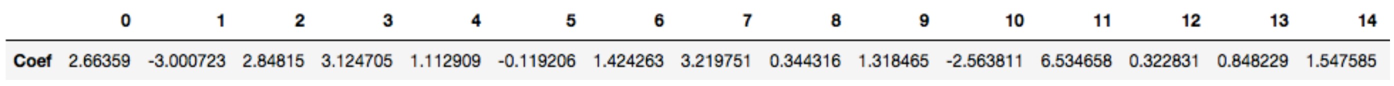
From the above experiments, it shows Logistic Regression in Python has the best performance with the default parameters: penalty = "l2", and C = 0.1. The performance of Logistic Regression classifier with selected parameters from Python is shown in *Figure 5* below. Further comparisons and evaluations will be discussed later.



(a) Confusion Matrix



(b) ROC Curve



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Coef	2.66359	-3.000723	2.84815	3.124705	1.112909	-0.119206	1.424263	3.219751	0.344316	1.318465	-2.563811	6.534658	0.322831	0.848229	1.547585

(c) Coefficients for each Feature in the developed Logistic Regression Function

Figure 5. Performance Summary for Logistic Regression Classifier with Selected Parameters

2) Decision Tree

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

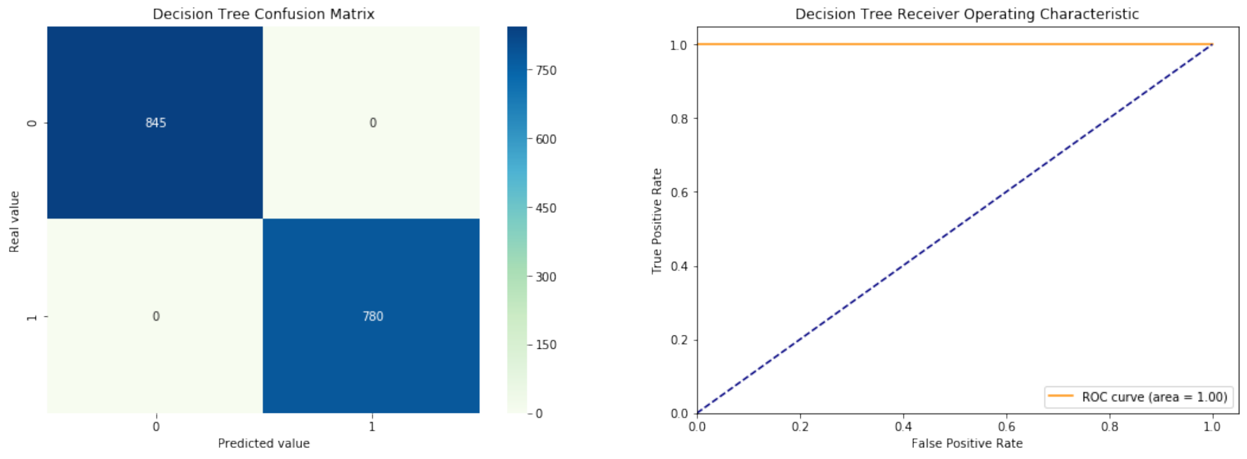
Three parameters are discussed and experimented here, including a) criterion: (default="gini") - the function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain; b) splitter: (default="best") - the strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split; c) max_features : (default=None) - the number of features to consider when looking for the best split. If "sqrt", then max_features=sqrt(n_features). If "log2", then max_features=log2(n_features). If None, then max_features=n_features. The experiments results are shown in *Table 12* below:

Exp#	Criterion	Splitter	max_features	Accuracy	F-measure	ROC Area
1(default)	"gini"	"best"	None	1.000	1.000	1.000
2	"entropy"	"best"	None	1.000	1.000	1.000
3	"gini"	"Random"	None	1.000	1.000	1.000

4	“entropy”	“Random”	“log2”	1.000	1.000	1.000
5	“gini”	“best”	“log2”	1.000	1.000	1.000
6	“entropy”	“best”	“log2”	1.000	1.000	1.000

Table 12. Decision Experiments Results

From the above experiments, it shows the Decision Tree Classifier in Python has the best performance with the default parameters: criterion = “gini”, splitter = “best”, and max_features = None. The performance of the decision tree classifier with selected parameters from Python is shown in *Figure 6* below. Further comparisons and evaluations will be discussed later.



(a) Confusion Matrix *(b) ROC Curve*

Figure 6. Performance Summary for Decision Tree Classifier with Selected Parameters

3) Naïve Bayes

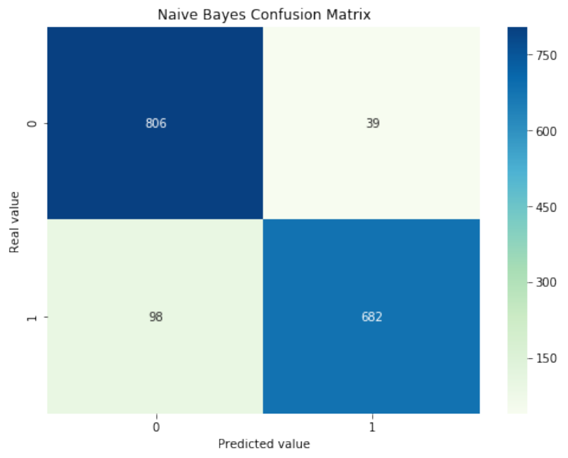
The Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features, and it is particularly suited when the dimensionality of the inputs is high.

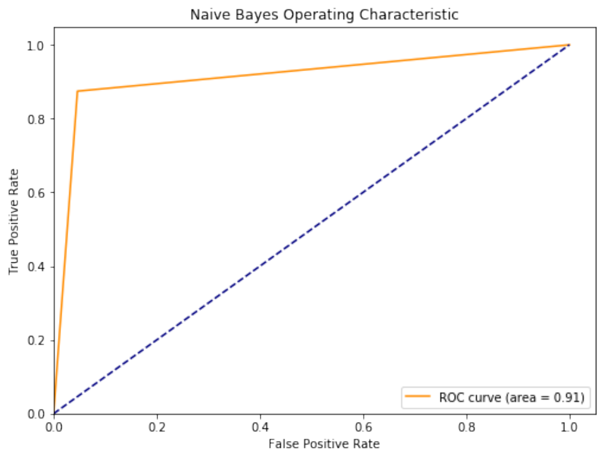
Only one parameer can be tuned in Python for the Naïve Bayes classifier, and it is discussed and experimented below, which is priors: (default: None) - prior probabilities of the classes. If specified the priors are not adjusted according to the data. The experiments results are shown in *Table 13* below:

Exp#	priors	Accuracy	F-measure	ROC Area
1(default)	None	0.906	0.898	0.900
2	[0.4,0.6]	0.907	0.900	0.905
3	[0.6,0.4]	0.908	0.900	0.906
4	[0.3,0.7]	0.916	0.909	0.914
5	[0.7,0.3]	0.914	0.903	0.911
6	[0.2,0.8]	0.883	0.883	0.884

Table 13. Naïve Bayes Experiments Results

From the above experiments, it shows the Naïve Bayes Classifier in Python has the best performance with the parameter: priors=[0.3,07]. The performance of Naïve Bayes classifier with selected parameters from Python is shown in *Figure 7* below. Further comparisons and evaluations will be discussed later.





(a) Confusion Matrix *(b) ROC Curve*

Figure 7. Performance Summary for Naïve Bayes Classifier with Selected Parameters

4) Artificial Neural Network

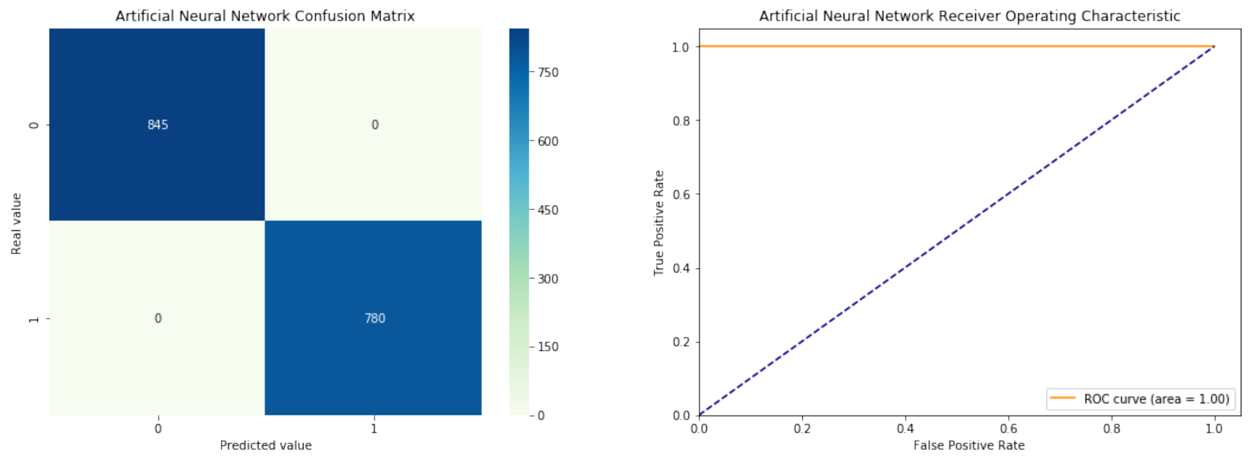
Artificial neural networks are brain-inspired systems which are intended to replicate the way that we humans learn. Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent tools for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize. Class MLP Classifier used here implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

Three parameters are discussed and experimented here, including a) hidden_layer_sizes: (tuple, length = n_layers - 2, default 100) – the ith element represents the number of neurons in the ith hidden layer; b) alpha: (default 0.0001) - L2 penalty (regularization term) parameter; and c) learning_rate_init: (default 0.001) - the initial learning rate used. It controls the step-size in updating the weights. The experiments results are shown in *Table 14* below:

Exp#	hiddenLayerSizes	alpha	learningRateInit	Accuracy	F-measure	ROC Area
1(default)	(5,2)	0.0001	0.001	1.000	1.000	1.000
2	(10,2)	0.0001	0.001	1.000	1.000	1.000
3	(5,2)	0.1	0.001	1.000	1.000	1.000
4	(5,2)	0.0001	0.1	1.000	1.000	1.000

Table 14. MLP Classifier Experiments *Results*

From the above experiments, it shows the MLP Classifier in Python has the best performance with the default parameters: hidden_layer_sizes = (5,2), alpha = 0.0001, and learning_rate_init = 0.001. The performance of MLP classifier with selected parameters from Python is shown in *Figure 8* below. Further comparisons and evaluations will be discussed later.



(a) Confusion Matrix (b) ROC Curve

Figure 8. Performance Summary for MLP Classifier with Selected Parameters

5) Support Vector Machine

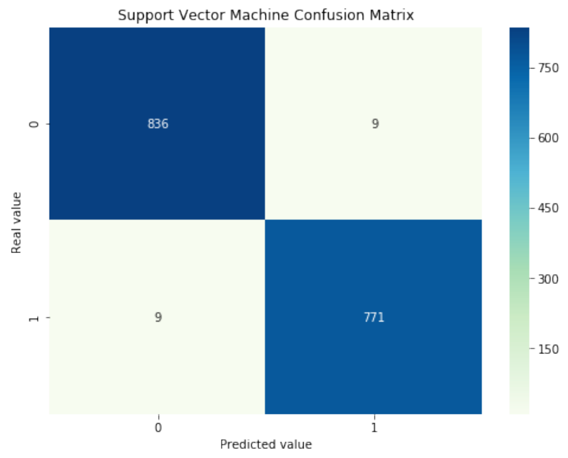
A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. SVC Classifier in Python is a C-Support vector classification. The implementation is based on "libsvm", and the fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples. The multiclass support is handled according to a one-vs-one scheme.

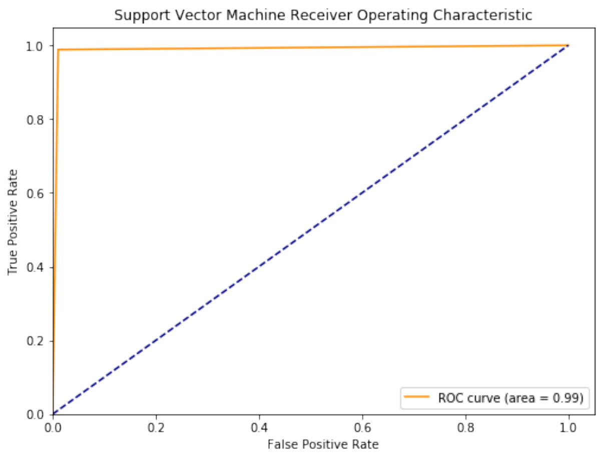
Two parameter are discussed and experimented here, including a) C: (default=1.0) - penalty parameter C of the error term; and b) kernel: (default='rbf') - specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples). The experiments results are shown in *Table 15* below:

Exp#	C	Kernel	Accuracy	F-measure	ROC Area
1(default)	1.0	"rbf"	0.989	0.988	0.990
2	1.0	"poly"	0.931	0.925	0.930
3	1.0	"linear"	0.963	0.961	0.963
4	0.5	"rbf"	0.916	0.904	0.913
5	1.5	"rbf"	0.945	0.940	0.943

Table 15. SVC Classifier Experiments Results

From the above experiments, it shows the SVC Classifier in Python has the best performance with the default parameters: C = 1.0, and kernel = "rbf". The performance of SVC classifier with selected parameters from Python is shown in *Figure 9* below. Further comparisons and evaluations will be discussed later.





(a) Confusion Matrix *(b) ROC Curve*

Figure 9. Performance Summary for SVC Classifier with Selected Parameters

Model Evaluation

During the modelling process, the 10-fold cross validation method is used to evaluate each classifier's performance. In order to select a modeling approach with the best performance, the performance statistics, including accuracy, precision, recall, F-Measure and ROC Area for each model are summarized in *Table 16* for further evaluation and comparison.

Model	Accuracy	Precision	Recall	F-Measure	ROC Area
Logistic Regression	0.963	0.962	0.963	0.961	0.963
Decision Tree	1.000	1.000	1.000	1.000	1.000
Naïve Bayes	0.916	0.903	0.916	0.909	0.914
Artificial Neural Network	1.000	1.000	1.000	1.000	1.000
Support Vector Machine	0.989	0.988	0.989	0.988	0.990

Table 16. Performance Statistics Summary for 5 Models

From *Table 16* above, it is observed that Logistic Regression, Decision Tree, ANN, and SVM approaches all have pretty good performance with the accuracy statistics greater than 95%. Although the Naïve Bayes approach also has good accuracy statistic that greater than 90%, its performance is relatively poor compared to other approaches. The Decision Tree and Artificial Neural Network approaches have the best performance statistics with 100% accuracy, indicating that Decision Tree and ANN approaches have better performance than others.

To further compare the performance of two approaches, ROC Curves for five classification approaches are plotted in the same figure, and the combined ROC Curves are shown in *Figure 10*. It confirms the conclusion above that Decision Tree and ANN approaches have a better performance compared with others and are considered to be candidates for the final model selection.

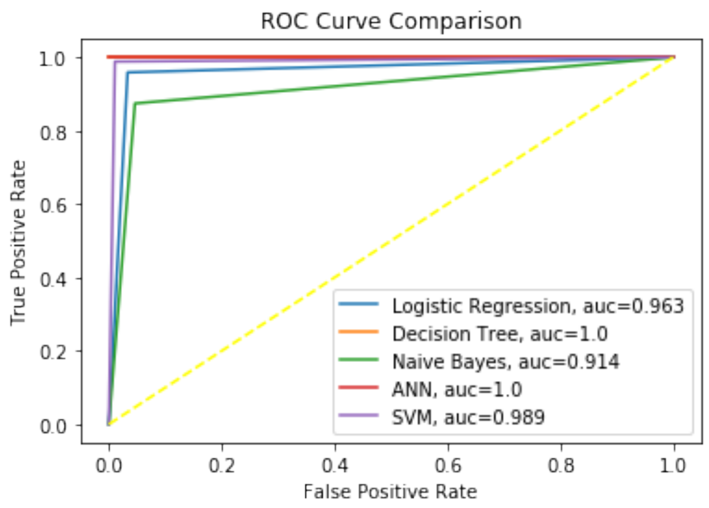


Figure 10. ROC Curves for Five Classification Approaches

Since both Decision Tree and ANN approach has a perfect performance statistics and ROC curve shown in *Table 16* and *Figure 10*, it seems that overfitting may be the next problem we need to considered in the classification process. Usually, overfitting means good performance on the training data, but poor generalization to other data. In the above classification process, the 10-fold cross validation method was used to all five approaches, and the performance results for Decision Tree and ANN approaches all show good on training data and test data. Additionally, both feature selection and dimensionality reduction was applied in the data transformation process, so technically there should be no overfitting problem in Decision Tree and ANN model built above.

In conclusion, the objective of this project is to investigate the data mining of physical characteristics data on the mushroom in order to create one or more classification models which are capable of accurately identifying if the certain mushroom is poisonous or edible. Through the discovery cycle including preliminary data exploration, data transformation, modeling, and model evaluation, two classification approaches are concluded to have the best performance on the Mushroom dataset provided from UCI Machine Learning Repository.

These two Models built in Python and with details listed below:

1) Decision Tree: “tree.DecisionTreeClassifier()” with default setting “criterion = gini”, “splitter = best”, and “max_features = None”;
2) Artificial Neural Network: “neural_network.MLPClassifier()” with default setting “hidden_layer_sizes = (5,2)”, “alpha = 0.0001”, and “learning_rate_init = 0.001”.

Reference

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