Wild Mushroom Identification

Team member: Xiang Yang, Shikuan Gao, Haoran Du, Ruiwen Shi

Instructor: Prof. Yifan Hu

Contents

Introduction

Mushroom, obtainable and common out in the field, is a good source of protein, vitamin and several minerals. However, only some of them are edible while the others are poisonous even fatal, which makes identifying them critical. Our simulation tries to unravel a relation between certain physical features of a mushroom and its poisonousness.

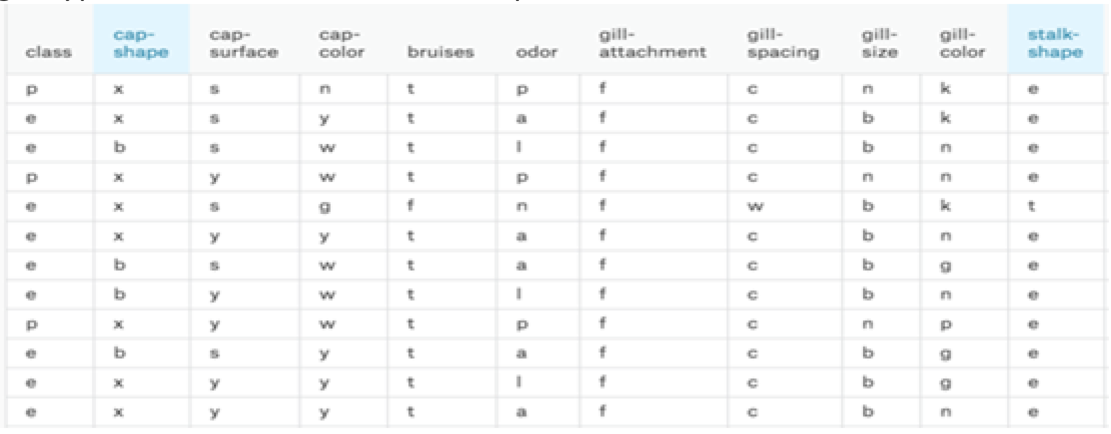
Objectives

Building suitable models to use for identification between edible and poisonous mushrooms. Search most related feature to identify mushrooms' edibility, thus provide a memorable criterion to decide whether it is edible when searching food in the field.

Data

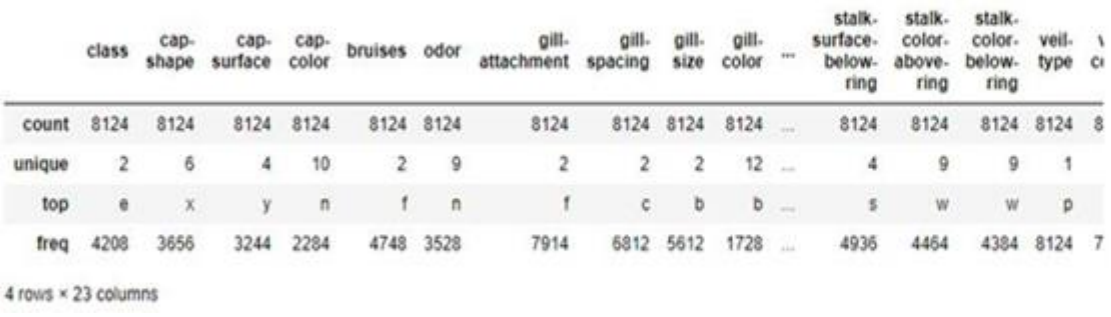
Data Description

The raw data of our project is from Kaggle, it is a dataset with 23 species of gilled mushrooms. By checking the dataset in the below of figure 1 and 2, we can see that there are over 8000 variables and 23 attributes such as cap shape, cap surface, cap color and so on. Each attribute influences on the outcome of a single type of mushroom is edible or poisonous.



class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape
p	x	s	n	t	p	f	c	n	k	e
e	x	s	y	t	a	f	c	b	k	e
e	b	s	w	t	l	f	c	b	n	e
p	x	y	w	t	p	f	c	n	n	e
e	x	s	g	f	n	f	w	b	k	t
e	x	y	y	t	a	f	c	b	n	e
e	b	s	w	t	a	f	c	b	g	e
e	b	y	w	t	l	f	c	b	n	e
p	x	y	w	t	p	f	c	n	p	e
e	b	s	y	t	a	f	c	b	g	e
e	x	y	y	t	l	f	c	b	g	e
e	x	y	y	t	a	f	c	b	n	e

Figure 1. Data set



	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	\ c
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124	8124	8124	8124	8
unique	2	6	4	10	2	9	2	2	2	12	...	4	9	9	1	
top	e	x	y	n	f	n	f	c	b	b	...	s	w	w	p	
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936	4464	4384	8124	7

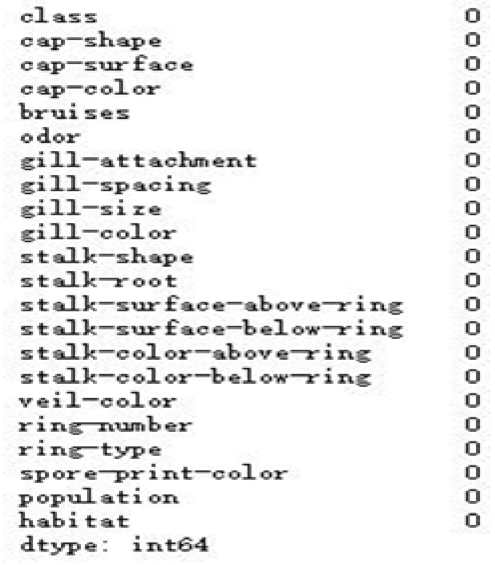
4 rows × 23 columns

Figure 2. Data check

Data Cleaning

Null check

First, we clean the dataset by null check. The reason of this data process is that dataset should have no null values. A null indicates that a variable doesn't point to any object and holds no value. Our project should avoid null otherwise there will be problems while building the model. Fortunately, from figure 3 in the below, it is clearly that the null values of each attributes is zero.

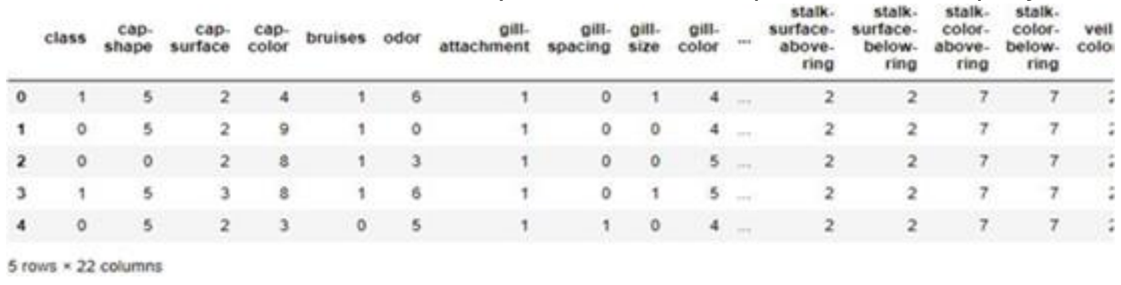


```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                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

Figure 3. Null check

Encoding

Second, in our mushroom dataset, each attribute is a string in python. Since the dataset has string values, it needed to convert all the unique values into integers. Furthermore, label encoding is performed on the data shown in the figure 4 as below. By doing so, each attribute has a sequence of numbers into a specialized format for efficient analyze in the future process of the project.



	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil colo
0	1	5	2	4	1	6	1	0	1	4	...	2	2	7	7	[illegible]
1	0	5	2	9	1	0	1	0	0	4	...	2	2	7	7	[illegible]
2	0	0	2	8	1	3	1	0	0	5	...	2	2	7	7	[illegible]
3	1	5	3	8	1	6	1	0	1	5	...	2	2	7	7	[illegible]
4	0	5	2	3	0	5	1	1	0	4	...	2	2	7	7	[illegible]

5 rows × 22 columns

Figure 4. *Label encoding*

Balance check

Third, we use data balance check technique to make sure data prediction will be unbiased towards to the frequent class. Imbalanced dataset is a special case for classification problem where the class distribution is not uniform among the classes. From the figure 5 shown as below, we can see that our dataset is basically balance between edible and poisonous.

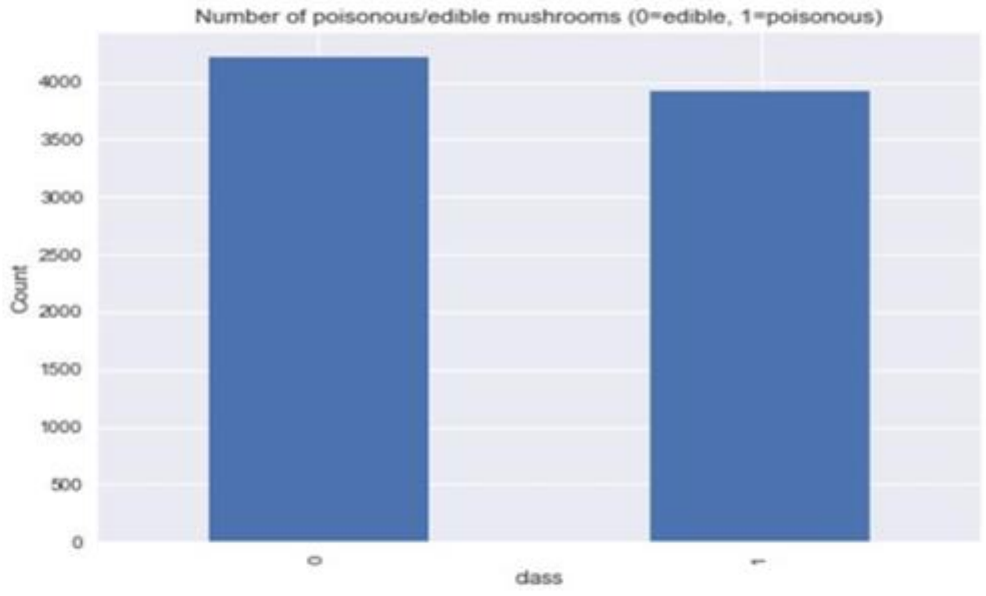


Figure 5. Data balance

Data Splitting

The fourth and the last part of data cleaning, we split the data into train data and test data according to the ratio of 8 to 2 to build and test the model. Also, we set up a plot correlation matrix to check the correlation of each attribute to another. The blue blocks mean positive correlation and red blocks means negative correlation between two attributes. The darker color means stronger correlation. For example, from the figure 6 shown in below, veil-color and gill-attachment have the highest positive correlation which is 0.9.

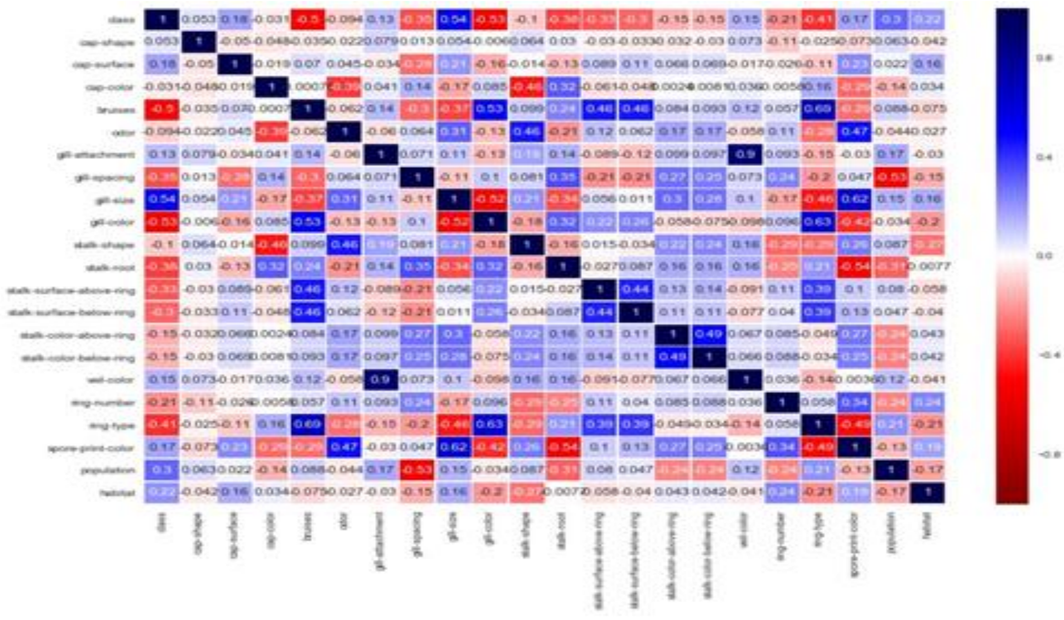
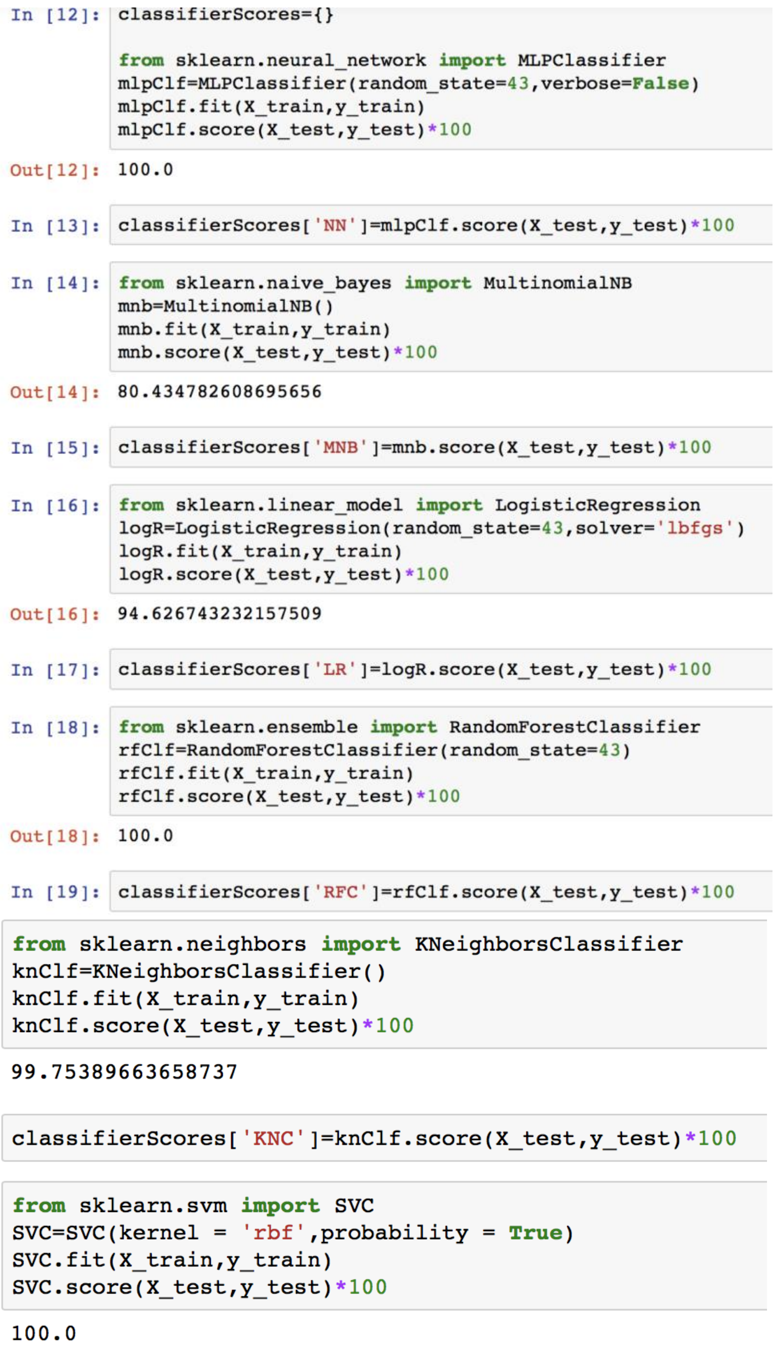


Figure 6. *Plot correlation matrix*

Modeling

We starts our modeling part after data cleaning. The whole modeling can be divided into four parts: approach 1 modeling, feature importance selection, approach 2 modeling and model comparison.

Firstly, we do the approach 1 modeling using python.



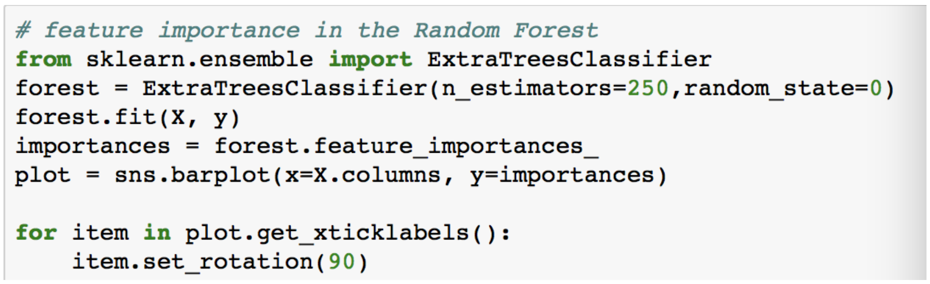
```
In [12]: classifierScores={}

         from sklearn.neural_network import MLPClassifier
         mlpClf=MLPClassifier(random_state=43,verbose=False)
         mlpClf.fit(X_train,y_train)
         mlpClf.score(X_test,y_test)*100

Out[12]: 100.0

In [13]: classifierScores['NN']=mlpClf.score(X_test,y_test)*100

In [14]: from sklearn.naive_bayes import MultinomialNB
         mnb=MultinomialNB()
         mnb.fit(X_train,y_train)
         mnb.score(X_test,y_test)*100

Out[14]: 80.434782608695656

In [15]: classifierScores['MNB']=mnb.score(X_test,y_test)*100

In [16]: from sklearn.linear_model import LogisticRegression
         logR=LogisticRegression(random_state=43,solver='lbfgs')
         logR.fit(X_train,y_train)
         logR.score(X_test,y_test)*100

Out[16]: 94.626743232157509

In [17]: classifierScores['LR']=logR.score(X_test,y_test)*100

In [18]: from sklearn.ensemble import RandomForestClassifier
         rfClf=RandomForestClassifier(random_state=43)
         rfClf.fit(X_train,y_train)
         rfClf.score(X_test,y_test)*100

Out[18]: 100.0

In [19]: classifierScores['RFC']=rfClf.score(X_test,y_test)*100
```

```
from sklearn.neighbors import KNeighborsClassifier
knClf=KNeighborsClassifier()
knClf.fit(X_train,y_train)
knClf.score(X_test,y_test)*100

99.75389663658737

classifierScores['KNC']=knClf.score(X_test,y_test)*100

from sklearn.svm import SVC
SVC=SVC(kernel = 'rbf',probability = True)
SVC.fit(X_train,y_train)
SVC.score(X_test,y_test)*100

100.0
```

Figure 7. *Modeling Approach 2*

As we can see from the above, all classifier achieve great grades and some of them even get the 100% accuracy. But it looks like models we set up are a little overfitting , next we try to find some ways to solve this problem.

Next, we use random forest to do feature importance selection.



```
# feature importance in the Random Forest
from sklearn.ensemble import ExtraTreesClassifier
forest = ExtraTreesClassifier(n_estimators=250,random_state=0)
forest.fit(X, y)
importances = forest.feature_importances_
plot = sns.barplot(x=X.columns, y=importances)

for item in plot.get_xticklabels():
    item.set_rotation(90)
```

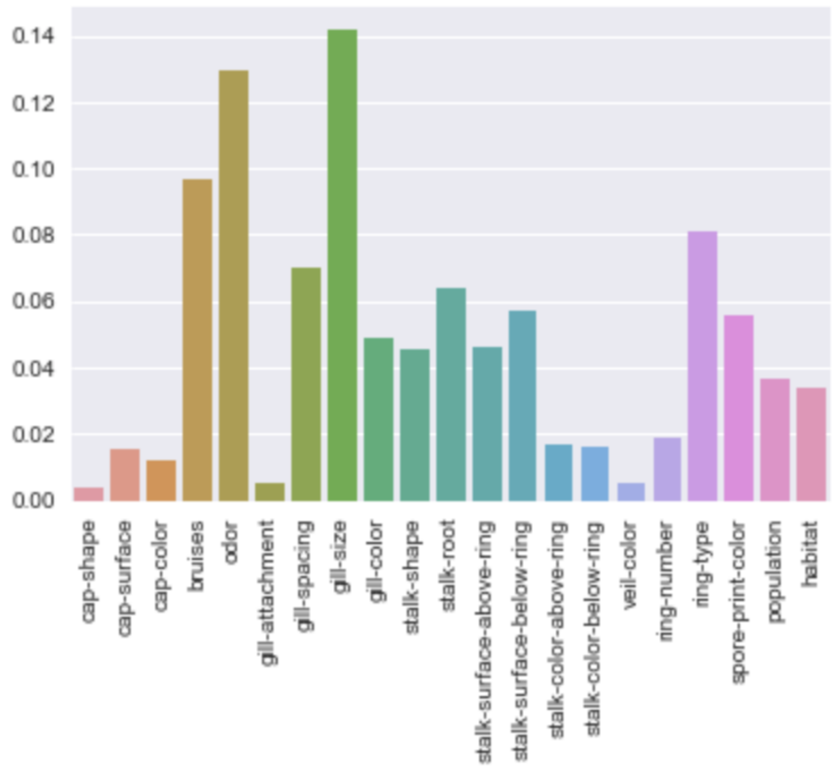
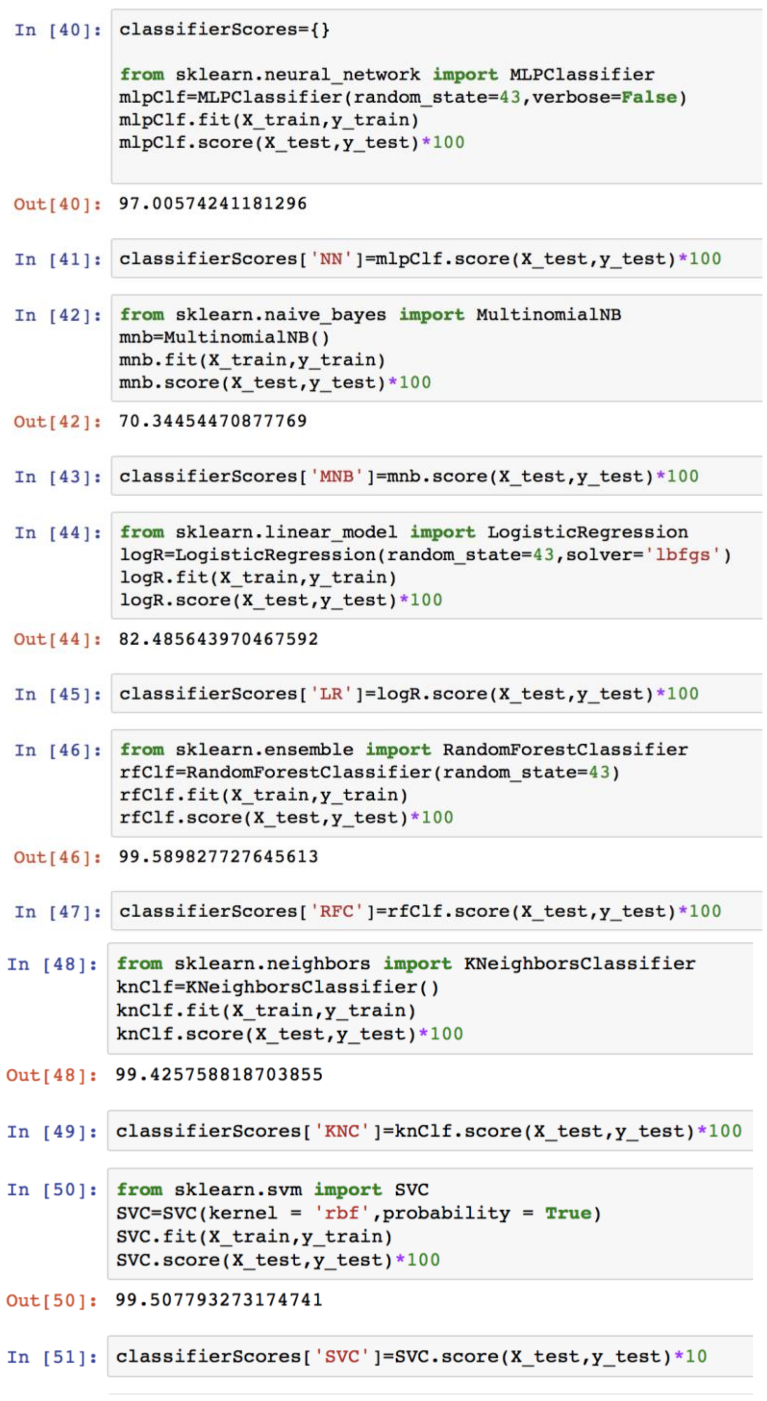


Figure 8. *Plot feature importance*

we can conclude that there are four features are most correlated to the predicted class.They are odor,gill-color, gill-size, spore-print-color respectively.

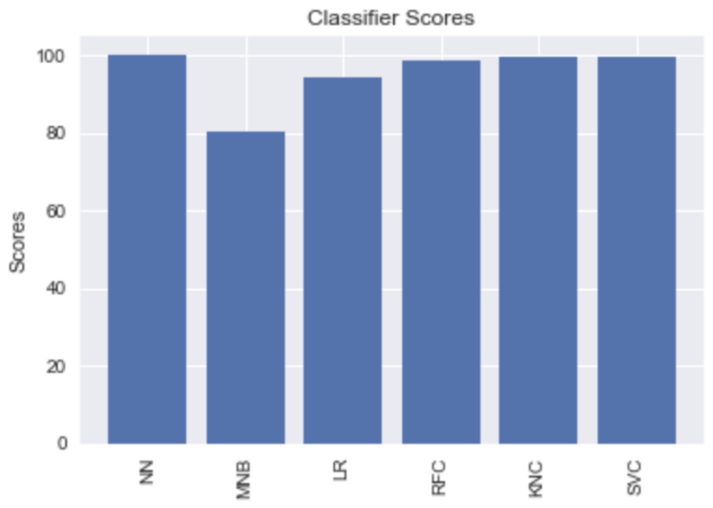
Since we have already got the top four correlated features, we can replace all features with these four feature to do modeling again to see if we can improve the overfitting problem.



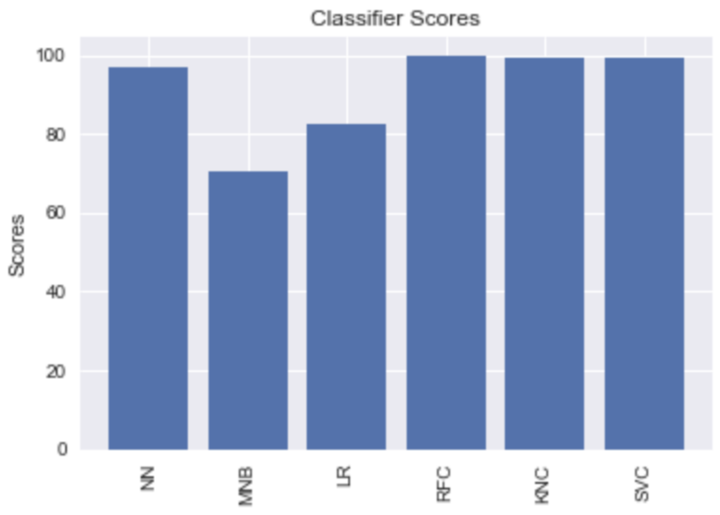
```
In [40]: classifierScores={}

         from sklearn.neural_network import MLPClassifier
         mlpClf=MLPClassifier(random_state=43,verbose=False)
         mlpClf.fit(X_train,y_train)
         mlpClf.score(X_test,y_test)*100

Out[40]: 97.00574241181296

In [41]: classifierScores['NN']=mlpClf.score(X_test,y_test)*100

In [42]: from sklearn.naive_bayes import MultinomialNB
         mnb=MultinomialNB()
         mnb.fit(X_train,y_train)
         mnb.score(X_test,y_test)*100

Out[42]: 70.34454470877769

In [43]: classifierScores['MNB']=mnb.score(X_test,y_test)*100

In [44]: from sklearn.linear_model import LogisticRegression
         logR=LogisticRegression(random_state=43,solver='lbfgs')
         logR.fit(X_train,y_train)
         logR.score(X_test,y_test)*100

Out[44]: 82.485643970467592

In [45]: classifierScores['LR']=logR.score(X_test,y_test)*100

In [46]: from sklearn.ensemble import RandomForestClassifier
         rfClf=RandomForestClassifier(random_state=43)
         rfClf.fit(X_train,y_train)
         rfClf.score(X_test,y_test)*100

Out[46]: 99.589827727645613

In [47]: classifierScores['RFC']=rfClf.score(X_test,y_test)*100

In [48]: from sklearn.neighbors import KNeighborsClassifier
         knClf=KNeighborsClassifier()
         knClf.fit(X_train,y_train)
         knClf.score(X_test,y_test)*100

Out[48]: 99.425758818703855

In [49]: classifierScores['KNC']=knClf.score(X_test,y_test)*100

In [50]: from sklearn.svm import SVC
         SVC=SVC(kernel = 'rbf',probability = True)
         SVC.fit(X_train,y_train)
         SVC.score(X_test,y_test)*100

Out[50]: 99.507793273174741

In [51]: classifierScores['SVC']=SVC.score(X_test,y_test)*10
```

Figure 9. *Modeling Approach 2*

As we can see from the above, scores of all models change. Next, we need to see the changes between the first classifier score plot and the second one.



Approach 1



Approach 2

Figure 10. Classifier *Score Comparison*

We can easily find that there are some changes happening on MNB model and LR model. And the others also have a little decrease on their scores.

Results

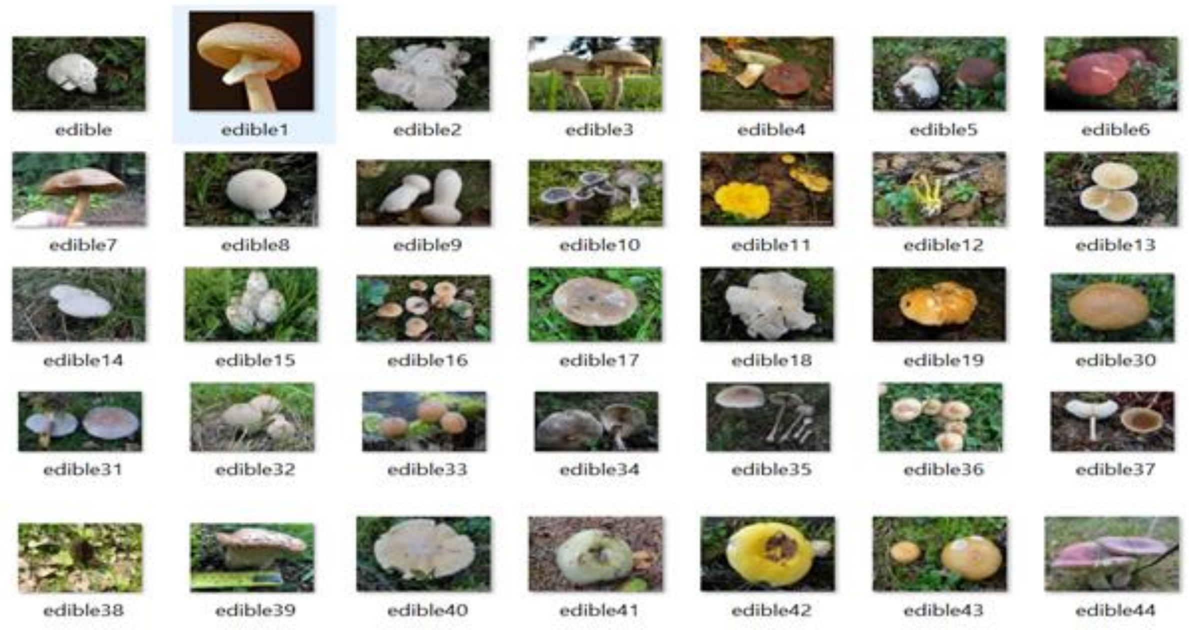
We can conclude that the best classifier is Random Forest whose score is 99.589. And top four features that are most important on predicted class are bruises, odor, gill-size, ring-type. Moreover, all models' scores decrease after we use four features to do modeling again which makes modeling more feasible.

Chanllenge: Image Classifier

From analysis above, we can tell that with only four features that are odor,gill-color, gill-size, spore-print-color, we can build a perfect model that can identify whether mushroom are edible or poisonous. It seems easy to do such identification with textual description of mushroom, but how about classifying mushroom by images?

Data collection

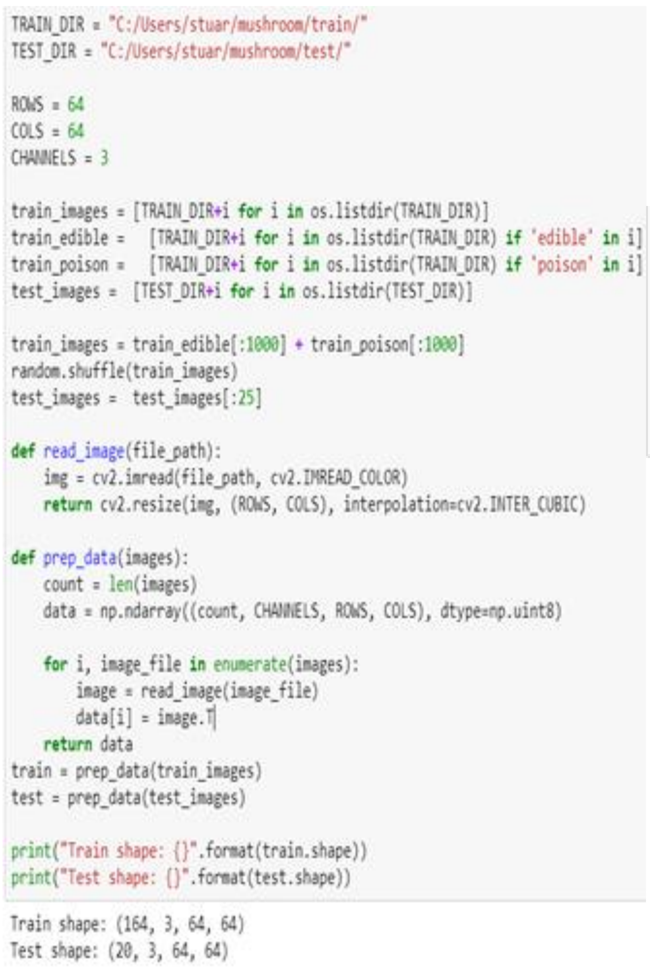
We manually collected 184 mushroom images from Mushroom World (http://www.mushroom.world/home/index), then renamed them by their properties -- “edible” or “poison”, and storage them into two files -- file “train” and file “test”, with 164 images and 20 images, respectively.



Snapshot of image data

Data preparation

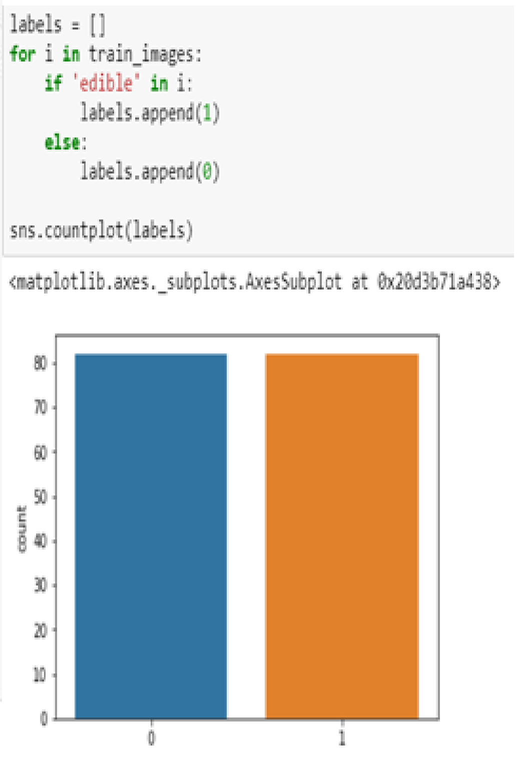
Firstly, we converted our images into numpy array with 64 rows and 64 columns and obtained their shape information. Then, we generated binary labels for our train dataset, assigning “0” for “poison” and “1” for “edible” and checked variable quantity of each label. It was important to have a relatively balance between two labels so that models would not be biased. As we noticed this factor when collecting images, we got 82 variables of each label in train dataset.



```
TRAIN_DIR = "C:/Users/stuar/mushroom/train/"
TEST_DIR = "C:/Users/stuar/mushroom/test/"

ROWS = 64
COLS = 64
CHANNELS = 3

train_images = [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR)]
train_edible =   [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR) if 'edible' in i]
train_poison =   [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR) if 'poison' in i]
test_images =  [TEST_DIR+i for i in os.listdir(TEST_DIR)]

train_images = train_edible[:1000] + train_poison[:1000]
random.shuffle(train_images)
test_images =  test_images[:25]

def read_image(file_path):
    img = cv2.imread(file_path, cv2.IMREAD_COLOR)
    return cv2.resize(img, (ROWS, COLS), interpolation=cv2.INTER_CUBIC)

def prep_data(images):
    count = len(images)
    data = np.ndarray((count, CHANNELS, ROWS, COLS), dtype=np.uint8)

    for i, image_file in enumerate(images):
        image = read_image(image_file)
        data[i] = image.T
    return data
train = prep_data(train_images)
test = prep_data(test_images)

print("Train shape: {}".format(train.shape))
print("Test shape: {}".format(test.shape))
```

```
Train shape: (164, 3, 64, 64)
Test shape: (20, 3, 64, 64)
```

Data preparation



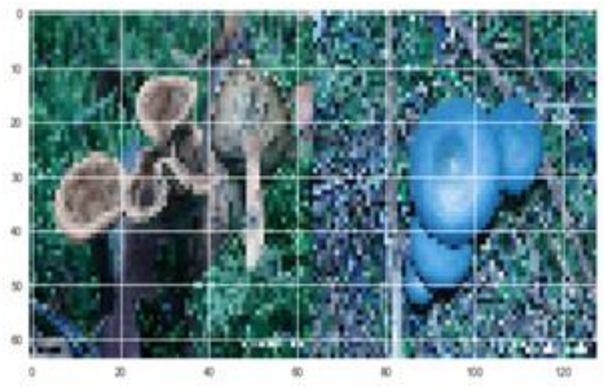
```
labels = []
for i in train_images:
    if 'edible' in i:
        labels.append(1)
    else:
        labels.append(0)

sns.countplot(labels)
```

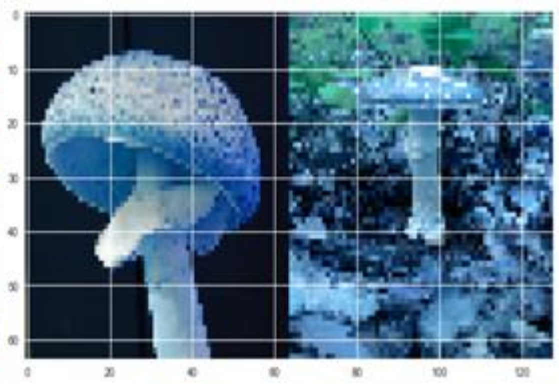
```
<matplotlib.axes._subplots.AxesSubplot at 0x20d3b71a438>
```

Label generation and check

Check input data

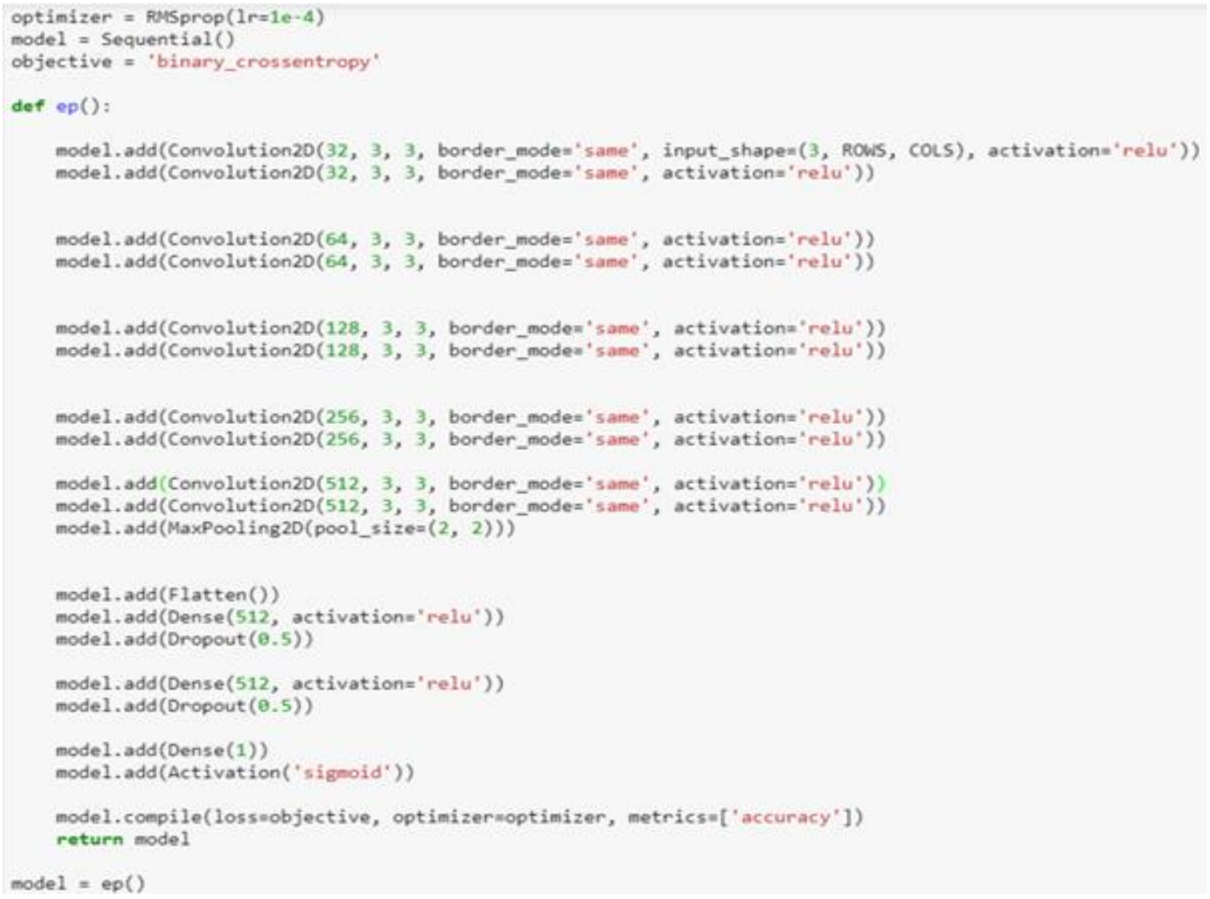
After ensuring that data was ready, we checked our input data by comparing each label to see what did our data look like after being converted into numpy array.





Train model

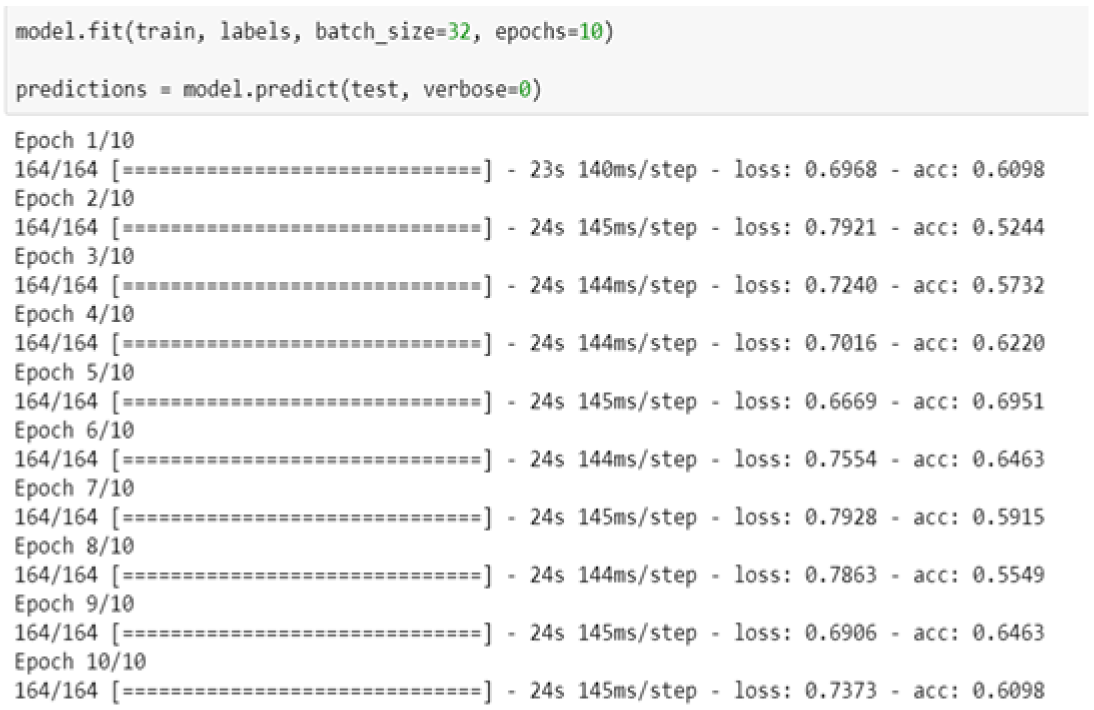
We used keras sequential model as our classifier, which is a linear stack of layers, and RMSProp as our optimizer. We added 5 convolutional layers into the sequential model, starting from applying 32 convolutional filters of size 3x3 each to our 64x64 images and ending at applying 512 convolutional filters. We set out dropout rate at 50%, which means dropping 1 of 2 inputs will randomly be excluded from each update cycle. Thus, we could get a higher learning rate.



```
optimizer = RMSprop(lr=1e-4)
model = Sequential()
objective = 'binary_crossentropy'

def ep():

    model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=(3, ROWS, COLS), activation='relu'))
    model.add(Convolution2D(32, 3, 3, border_mode='same', activation='relu'))


    model.add(Convolution2D(64, 3, 3, border_mode='same', activation='relu'))
    model.add(Convolution2D(64, 3, 3, border_mode='same', activation='relu'))


    model.add(Convolution2D(128, 3, 3, border_mode='same', activation='relu'))
    model.add(Convolution2D(128, 3, 3, border_mode='same', activation='relu'))


    model.add(Convolution2D(256, 3, 3, border_mode='same', activation='relu'))
    model.add(Convolution2D(256, 3, 3, border_mode='same', activation='relu'))

    model.add(Convolution2D(512, 3, 3, border_mode='same', activation='relu'))
    model.add(Convolution2D(512, 3, 3, border_mode='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))


    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.compile(loss=objective, optimizer=optimizer, metrics=['accuracy'])
    return model

model = ep()
```

Fit the model

In this section, we set two parameters, which are batch size and epoch, at 32 and 10, respectively, which means in each fitting update, 32 samples will be randomly selected from train dataset to fit the model and the entire train dataset will be passed over 10 times.

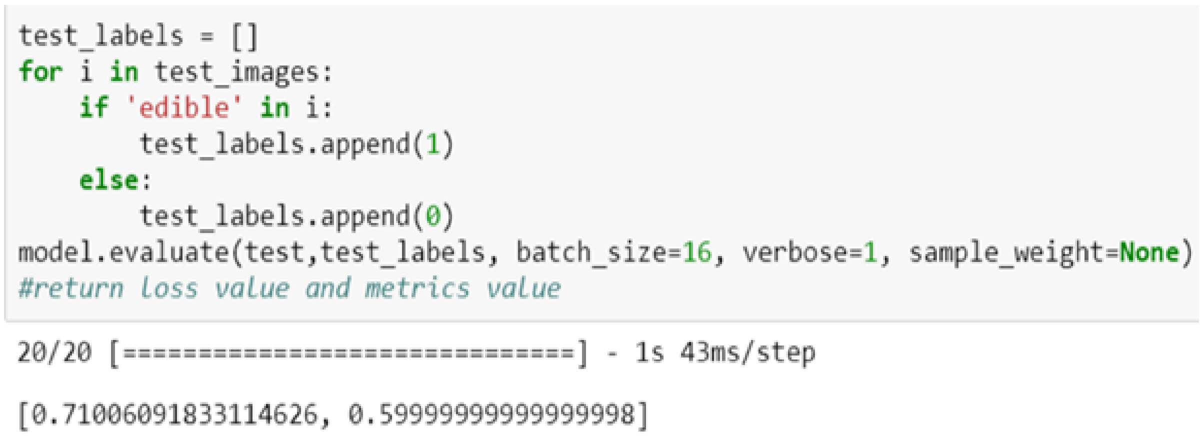


```
model.fit(train, labels, batch_size=32, epochs=10)

predictions = model.predict(test, verbose=0)
```

```
Epoch 1/10
164/164 [==============================] - 23s 140ms/step - loss: 0.6968 - acc: 0.6098
Epoch 2/10
164/164 [==============================] - 24s 145ms/step - loss: 0.7921 - acc: 0.5244
Epoch 3/10
164/164 [==============================] - 24s 144ms/step - loss: 0.7240 - acc: 0.5732
Epoch 4/10
164/164 [==============================] - 24s 144ms/step - loss: 0.7016 - acc: 0.6220
Epoch 5/10
164/164 [==============================] - 24s 145ms/step - loss: 0.6669 - acc: 0.6951
Epoch 6/10
164/164 [==============================] - 24s 144ms/step - loss: 0.7554 - acc: 0.6463
Epoch 7/10
164/164 [==============================] - 24s 145ms/step - loss: 0.7928 - acc: 0.5915
Epoch 8/10
164/164 [==============================] - 24s 144ms/step - loss: 0.7863 - acc: 0.5549
Epoch 9/10
164/164 [==============================] - 24s 145ms/step - loss: 0.6906 - acc: 0.6463
Epoch 10/10
164/164 [==============================] - 24s 145ms/step - loss: 0.7373 - acc: 0.6098
```

Performance of the classifier

We evaluated our classifier by calculating its loss value and matrics value, which are 0.71 and 0.6, respectively. From the evaluation, we can tell our classifier performs not well at classifying mushrooms' edibility from poison by analysing their images.

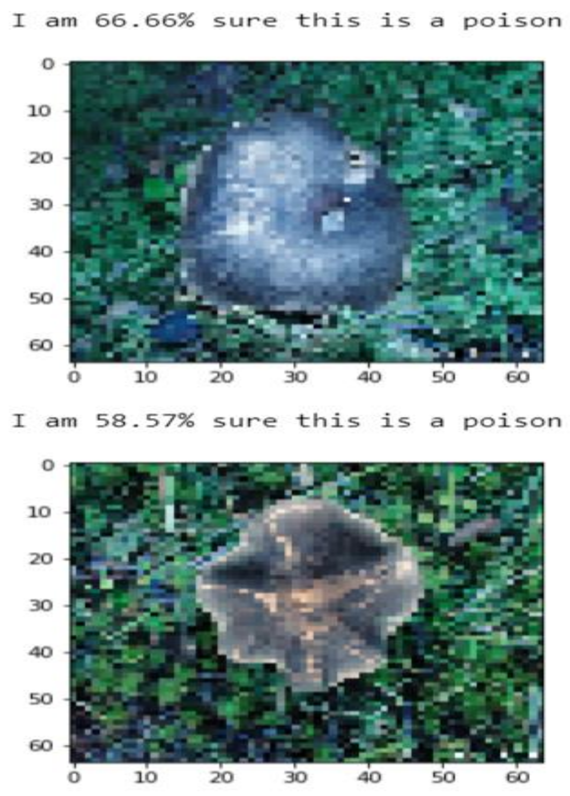


```
test_labels = []
for i in test_images:
    if 'edible' in i:
        test_labels.append(1)
    else:
        test_labels.append(0)
model.evaluate(test,test_labels, batch_size=16, verbose=1, sample_weight=None)
#return loss value and metrics value
```

```
20/20 [==============================] - 1s 43ms/step

[0.71006091833114626, 0.59999999999999998]
```

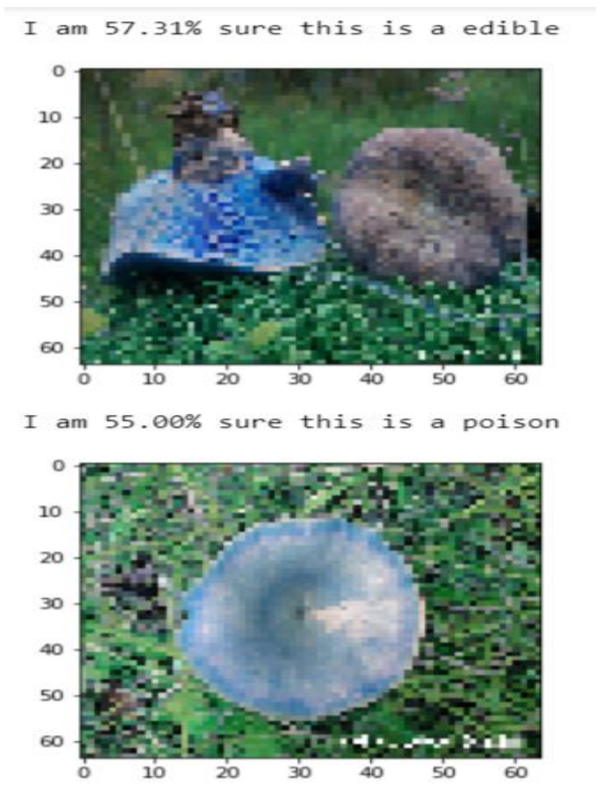
Snapshots of Classified Results

Mushrooms’ true labels in image 1, 2 and 3 are edible, and those in image 4 are poison.



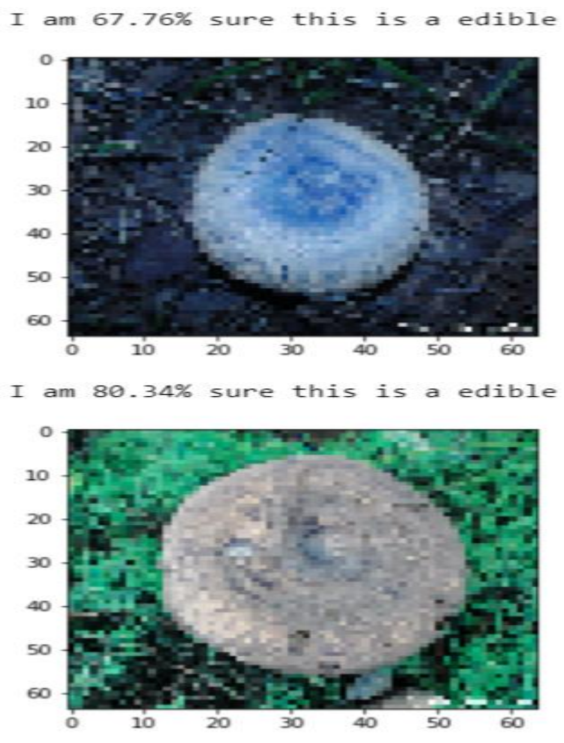
```
I am 66.66% sure this is a poison
I am 58.57% sure this is a poison
```

Image 1



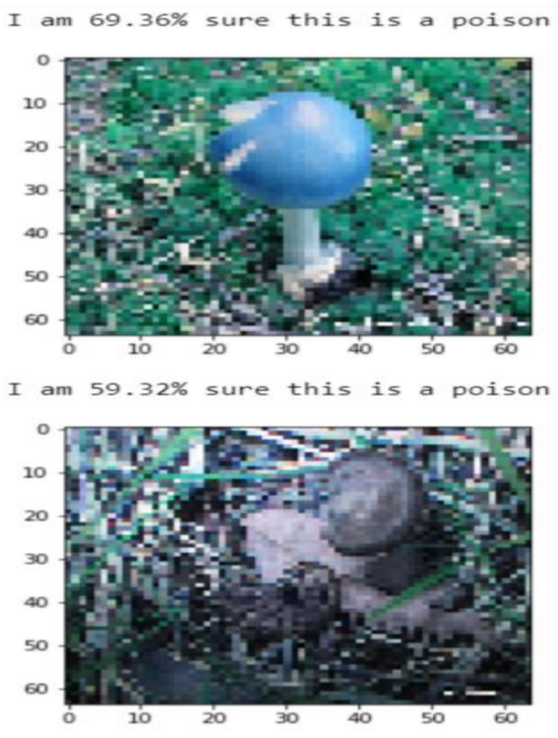
```
I am 57.31% sure this is a edible
I am 55.00% sure this is a poison
```

Image 2



```
I am 67.76% sure this is a edible
I am 80.34% sure this is a edible
```

Image 3



```
I am 69.36% sure this is a poison
I am 59.32% sure this is a poison
```

Image 4