Northeastern University
College *of* Engineering

Northeastern University
IE7374 Machine Learning
Instructor: Ramin Mohammadi
Spring 2021

Group 8

Lei Duan

Yue Pang

Yang Yang

Abstract

This report provides two methods for identifying whether a mushroom is edible or poisonous. We use Logistic Regression model and Support Vector Machine (SVM) method for this classification problem. We wrote our own code for these two methods, and also compared our results with the results using existing functions in machine learning libraries in python.
Data Source: https://archive.ics.uci.edu/ml/datasets/Mushroom
Github Link: https://github.com/pyusbos/Mushroom-Classification

Introduction

Mushroom is a kind of food that many people like to eat. However, some kind of mushroom can be poisonous. There are many kinds of features that mushroom has, cap-shape, cap-surface, cap-color, odor, etc. Each feature has different types, take cap-shape for example, there are bell, conical, convex, flat, knobbed, and sunken. It is hard for us to distinguish whether a mushroom is edible or poisonous based on the vary features.

The dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). 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.

Data description

There are 8124 entries, and 22 attributes in this dataset. The data type of the attributes are object, and there is no null value in this dataset.

The attribute information is shown below:

Attribute Information: (classes: edible = e, poisonous = p)

1. cap-shape: bell = b, conical = c,convex = x, flat = f, knobbed = k, sunken = s
2. cap-surface: fibrous = f, grooves = g, scaly = y, smooth = s
3. cap-color: brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, yellow = y
4. bruises: yes = t, no = f
5. odor: almond = a, anise = l, creosote = c, fishy = y, foul = f, musty = m, none = n, pungent = p, spicy = s
6. gill-attachment: attached = a, descending = d, free = f, notched = n
7. gill-spacing: close = c, crowded = w, distant = d
8. gill-size: broad = b, narrow = n
9. 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
10. stalk-shape: enlarging = e, tapering = t
11. stalk-rootbulbous = b, club = c, cup = u, equal = e, rhizomorphs = z, rooted = r, missing = ?
12. stalk-surface-above-ring: fibrous = f, scaly = y, silky = k, smooth = s
13. stalk-surface-below-ring: fibrous = f, scaly = y, silky = k, smooth = s
14. stalk-color-above-ring: brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y

15. stalk-color-below-ring: brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y
16. veil-type: partial = p, universal = u
17. veil-color: brown = n, orange = o, white = w, yellow = y
18. ring-number: none = n, one = o, two = t
19. ring-type: cobwebby = c, evanescent = e, flaring = f, large = l, none = n, pendant = p, sheathing = s, zone = z
20. spore-print-color: black = k, brown = n, buff = b, chocolate = h, green = r, orange = o,purple = u, white = w, yellow = y
21. population: abundant = a, clustered = c, numerous = n, scattered = s, several = v, solitary = y
22. habitat: grasses = g, leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d

Methods

Our team used Logistic Regression model and Support Vector Machine (SVM) method for this classification problem. We wrote our own code for logistic regression and Support Vector Machine (SVM). We also used existing functions in machine learning libraries for these two methods in python to verify our results.

As for SVM method, we applied hard margin SVM on our own code, and we also applied gaussian RBF and soft margin SVM using existing functions in sklearn library.

Explanatory data analysis (EDA)

There are 22 features: cap-shape, cap-surface, cap-color etc. and 8124 records in sum. And there are no any missing values in this dataset. We also checked if there is any outliers in this dataset:

```
# check if there is any outliers
feature_value = []
for i in range(df.shape[1]):
    temp = df.iloc[:,i].drop_duplicates().to_list()
    feature_value.append(temp)
    print(df.columns[i],':',temp)
```

We found that the numbers of each class in our dataset: edible and poisonous are not unbalanced (for eatable: 51.8%, for poisonous: 48.2%). But we do notice in some features like cap-color, the numbers of each category are quite unbalanced: for some categories, there are thousands of records, but for others, there are just a few records.

Feature Engineering：

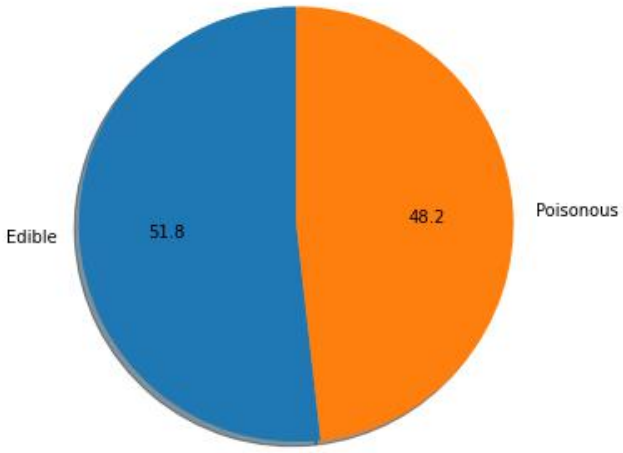
Because all features in the mushroom dataset are all categorical variables, we change these values into numbers with label encoder for our model fitting next step.

```python
from sklearn.preprocessing import LabelEncoder

# encode data

le = LabelEncoder()
arr_ec = df.apply(le.fit_transform).values
```

```python
# transform encoded data into dataframe
df_ec = pd.DataFrame(data = arr_ec,columns=df.columns)
```

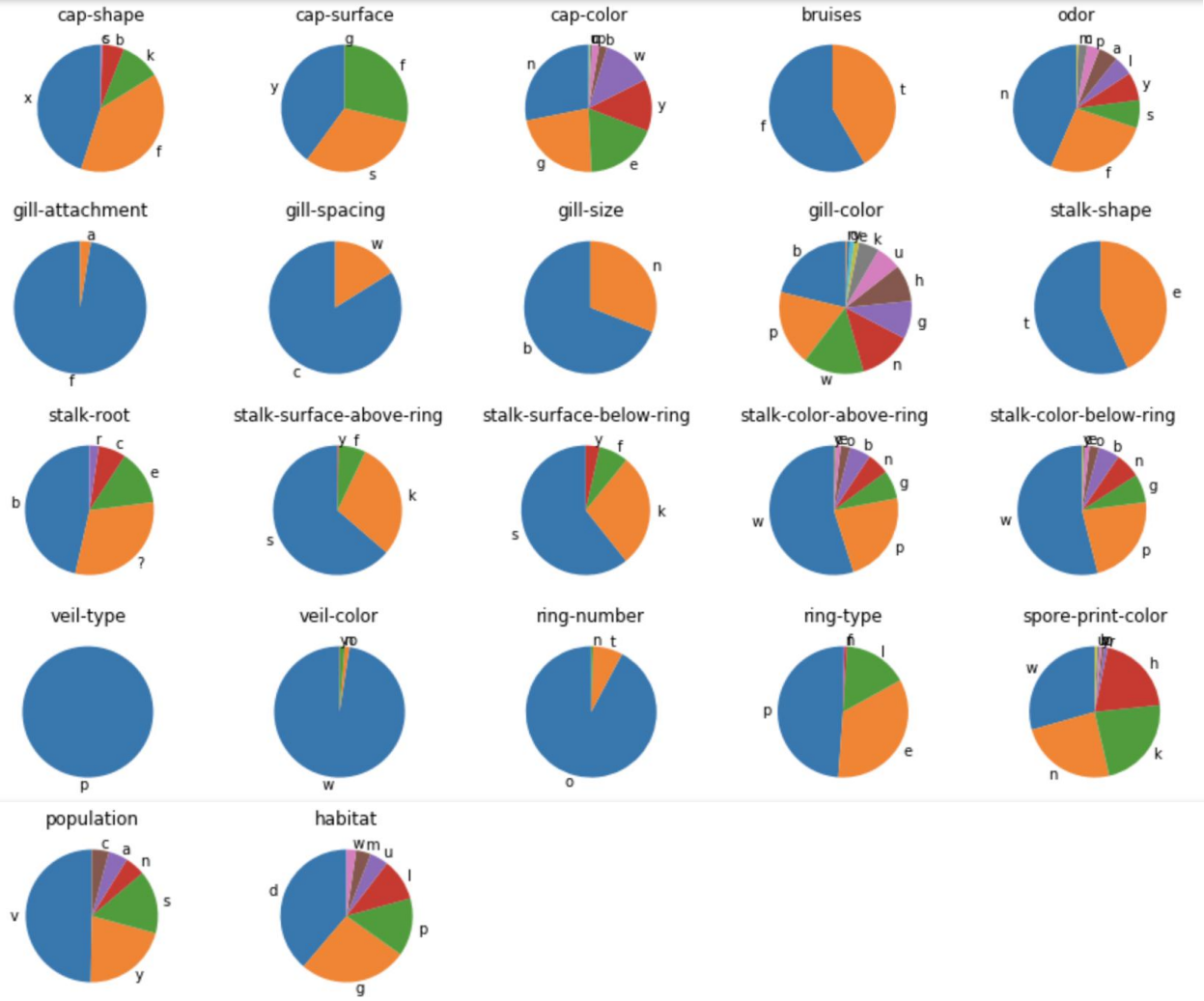
Statistical Analysis:

Among all of the 8124 entries, 51.8% of them are edible, while 48.2% of them are poisonous.

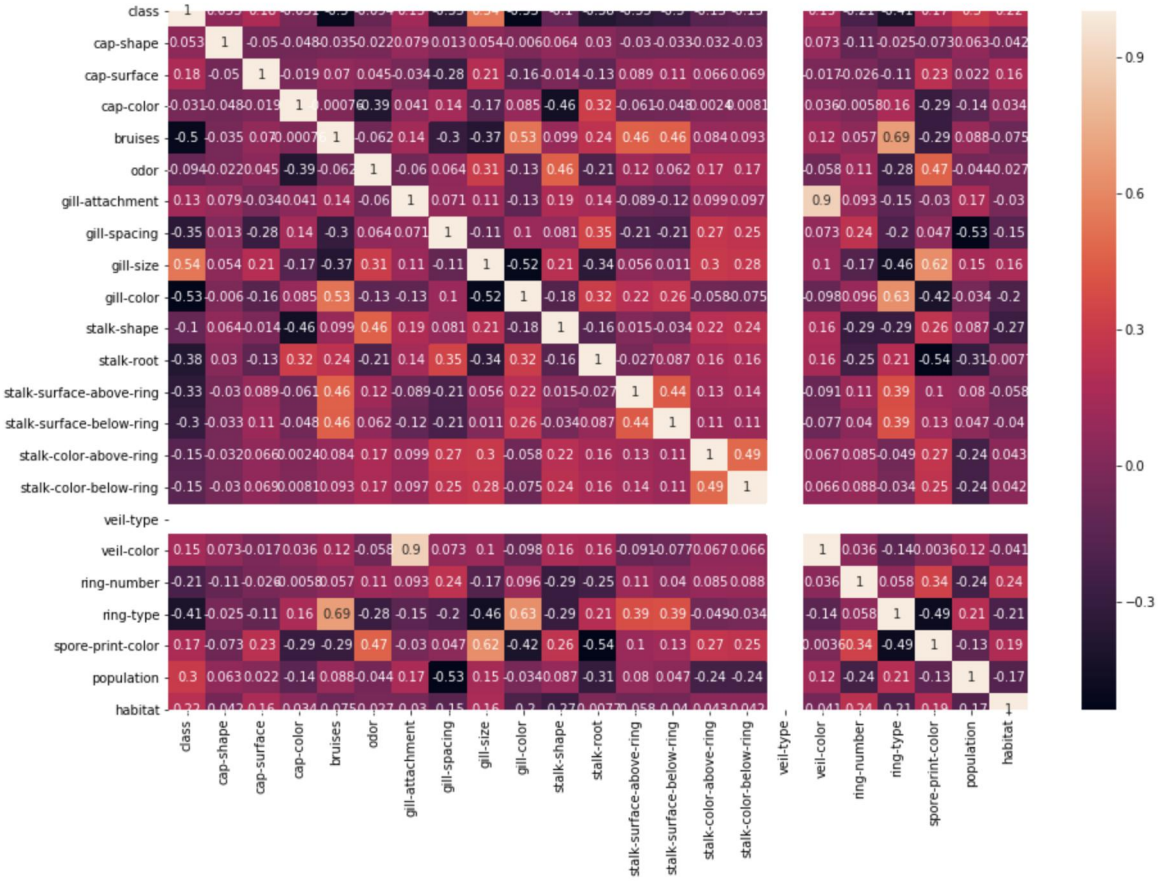


we found our target feature is unbiasd (nearly 1:1), so we could use accuracy as our key evaluation matrix.

The proportion of the feature values is shown below:



And the correlation heatmap is as follows:

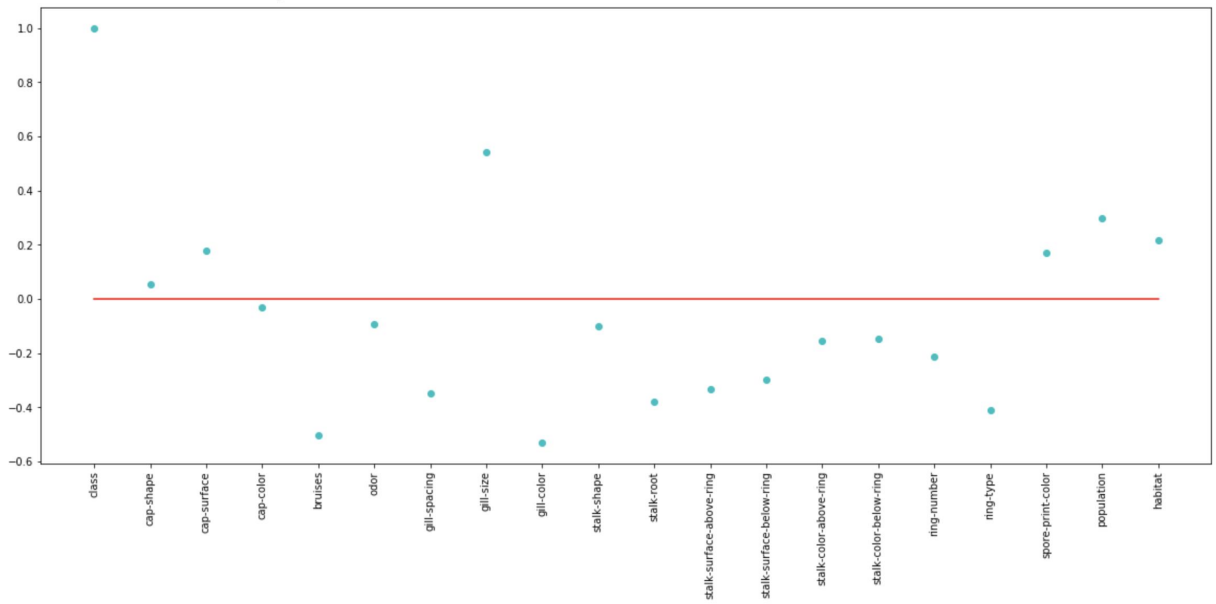


Feature Selection：

Our team also removed features with low variance:

```
mark =[]

for i in range(1,num_features+1):
    dic = dict()
    col = df.columns[i]
    count = df[col].value_counts()
    ind = count.index.to_list()

    for j in range(len(ind)):
        pro = round(count[j]/num_samples,4)
        dic[ind[j]] = pro

        if pro > 0.95:
            mark.append(col)

    print(col,':')
    print(pd.DataFrame([dic]))
```

And we dropped the feature that has a dominant feature value and plotted the correlation between class and all features:



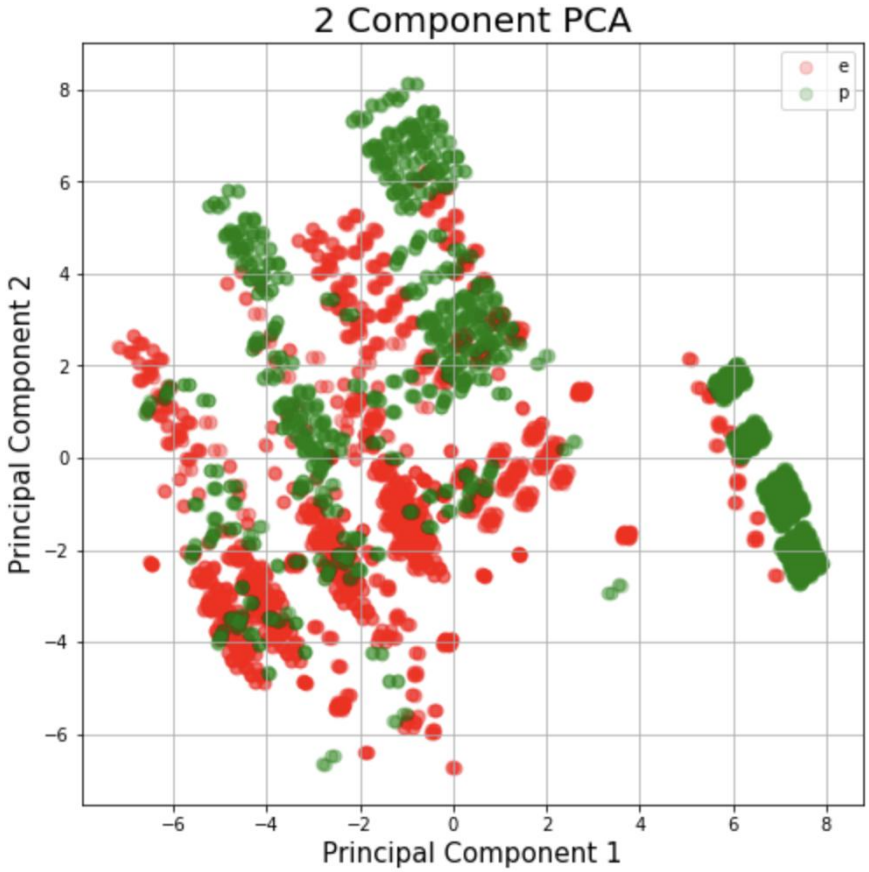
After that, we performed PCA on the dataset:

```
from sklearn.decomposition import PCA

X = df_ec.iloc[:,1:]
pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)

print(sum(pca.explained_variance_ratio_))

principalDf = pd.DataFrame(data = X2D
             , columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, df[['class']]], axis = 1)
finalDf.head(5)
```

```
0.5042015764278096
```

	principal component 1	principal component 2	class
0	0.227967	-0.346115	p
1	-1.937132	4.795939	e
2	-1.654512	2.463813	e
3	-1.252380	1.678875	p
4	1.580976	-1.002830	e

Then we visualized the PCA components:



We chose variables that account for 95% variance, and there are 10 variables in total.

The result is as follows:

```
# Choosing the right number of dimensions
# How many variables could account for 95% variance.
pca = PCA()
pca.fit(X)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1

print(d)
```

```
10
```

```
# PCA

pca = PCA(n_components = d)
X_10D = pca.fit_transform(X)

#print(sum(pca.explained_variance_ratio_))
```

```
df_10D = pd.concat([df_ec['class'],pd.DataFrame(X_10D)],axis=1)
```

```
df_10D.head()
```

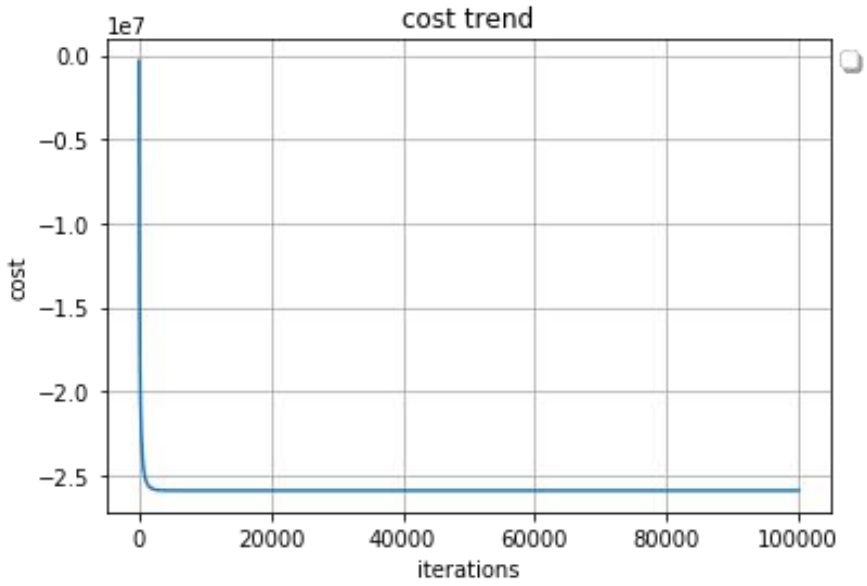
	class	0	1	2	3	4	5	6	7	8	9
0	1	0.227967	-0.346115	-1.424099	-1.248098	2.088075	2.647938	-1.029328	3.642536	-0.925853	0.366047
1	0	-1.937132	4.795939	-3.512184	-1.248272	0.549882	-0.283598	2.550986	-0.620545	0.231919	0.071637
2	0	-1.654512	2.463813	-3.881439	1.218642	1.043769	-2.052202	-2.192067	0.669014	-0.188370	0.330477
3	1	-1.252380	1.678875	-3.565712	0.586422	0.680268	2.793761	-0.393990	4.161018	-1.013840	-0.498848
4	0	1.580976	-1.002829	-1.255122	-1.969164	0.069668	3.303160	-1.256111	-2.007016	0.601937	-1.346899

Results.

Logistic Regression model:

Firstly, we wrote our own code for logistic regression model.

Detailed code can be viewed in our code file.



Accuracy: 0.8166153846153846

Recall: 0.7745740498034076

precision: 0.8242677824267782

We applied logistic regression function in sklearn library secondly:

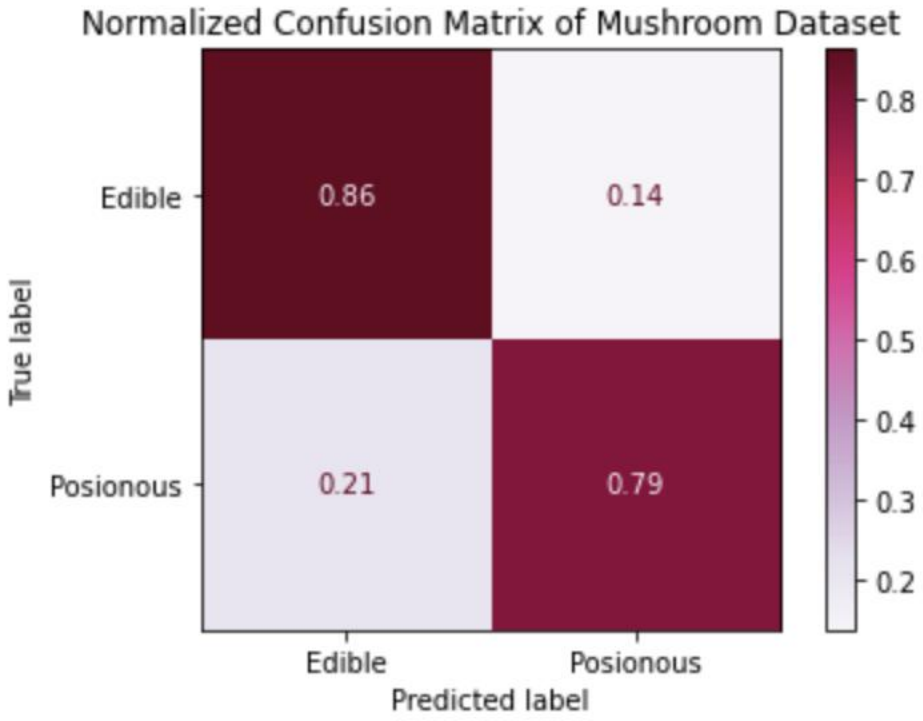
```
# use sklearn package

from sklearn.model_selection import train_test_split

X = df_10D.iloc[:,1:]
y = df_10D.iloc[:,0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2021)
```

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(random_state = 0)
lr.fit(X_train, y_train)

#Make Prediction
y_pred = lr.predict(X_test)
```

```
from sklearn.metrics import accuracy_score

accuracies = dict()
accuracies['Logistic Regression'] = accuracy_score(y_test, y_pred)
print('Accuracy is: ' + str(accuracy_score(y_test, y_pred)))
```

```
Accuracy is: 0.827076923076923
```

And the confusion matrix is:



SVM method:

We use train_test_split function in sklearn library to split training data and test data. 80% of the data is split to training data, and 20% is test data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 2021)
```

Then we standard scaling the data:

```python
# standard scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

After that, we perform hard margin SVM on the dataset:

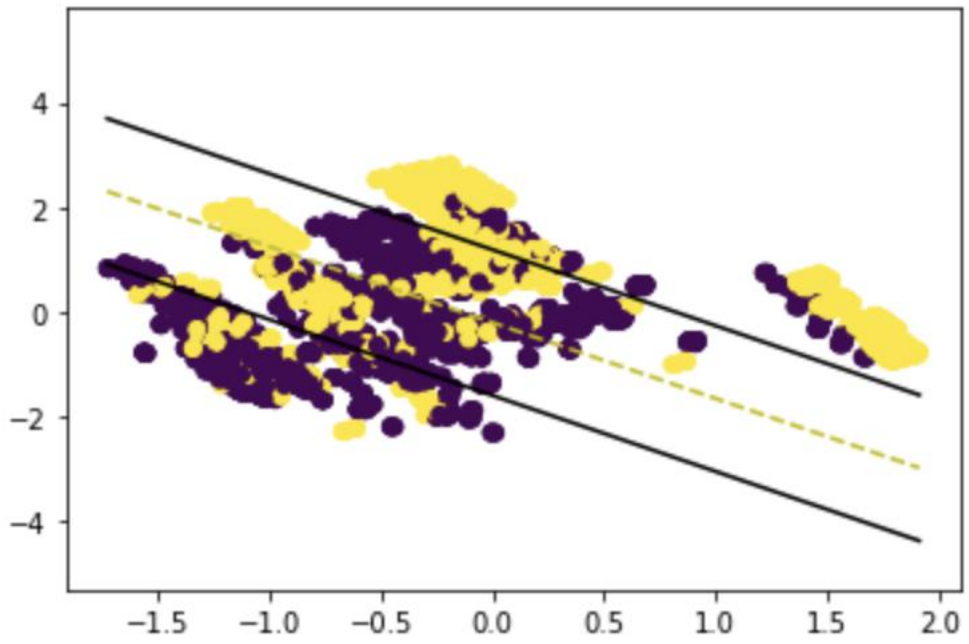
```python
class SVM:
    def __init__(self, learningrate = 0.001, _lambda = 0.001, n_iters = 100):
        self.learningrate = learningrate
        self._lambda = _lambda
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape[0], X.shape[1]
        y_ = np.where(y <= 0, -1, 1)

        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) + self.b) >= 1

                if condition:
                    self.w -= self.learningrate * (2 * self._lambda * self.w)
                else:
                    self.w -= self.learningrate * (2 * self._lambda * self.w - np.dot(y_[idx], x_i))
                    self.b -= self.learningrate * (-y_[idx])

        def predict(self, X):
            y_hat = np.dot(X, sel.w) + self.b
            return np.sign(y_hat)
```

accuracy: 0.8356923076923077

recall: 0.7748756218905473

precision: 0.8787023977433004

We visualized the result of this SVM model:

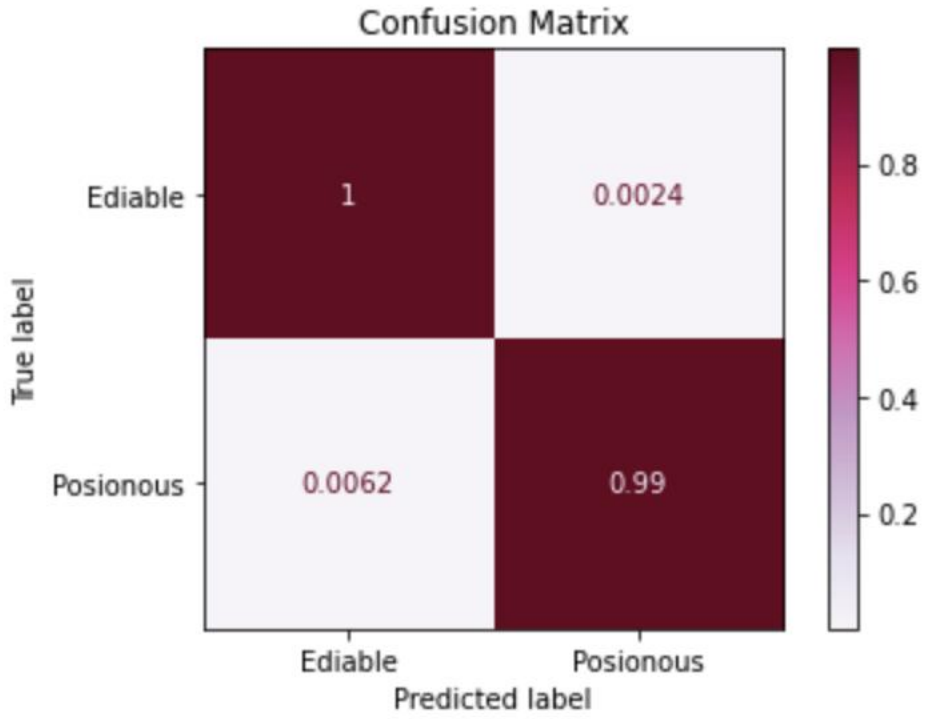


Also, we use scikit package to verify the answer. We use gaussian RBF method.

```
# use scikit package to varify the answer
# here we use gussian RBF method
from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

accuracies = dict()

accuracies['RBF_kernel SVM'] = accuracy_score(y_test, y_pred)
print('Accuracy is:' + str(accuracy_score(y_test, y_pred)))
```

```
Accuracy is:0.9956923076923077
```

The confusion matrix is shown below:

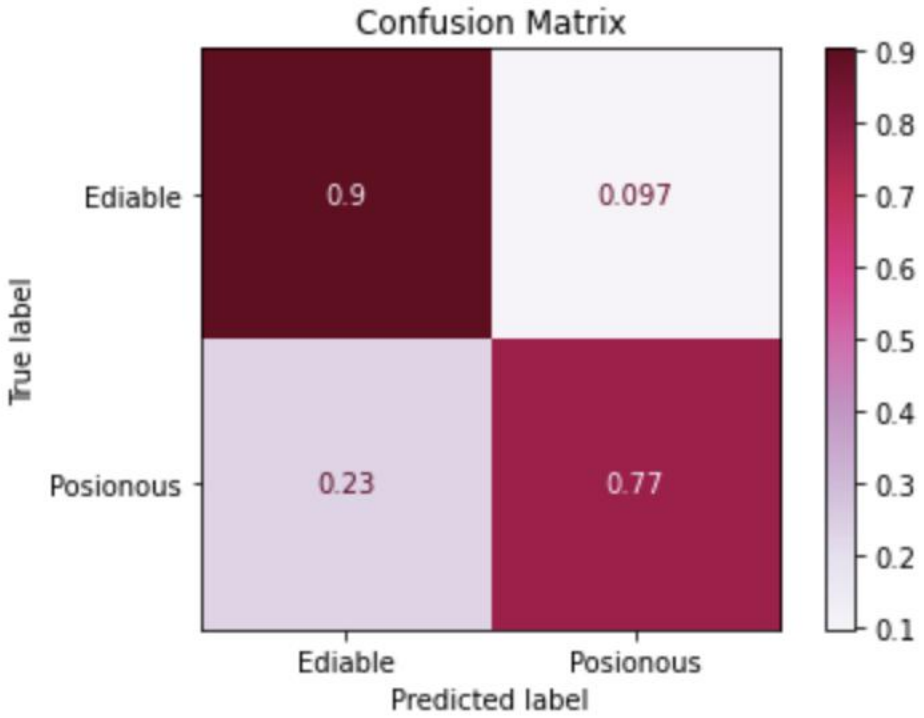


We also performed soft margin SVM using LinearSVC function in sklearn library:

```
from sklearn.svm import LinearSVC
classifier_2 = LinearSVC(C = 1, loss = 'hinge')

classifier_2.fit(X_train, y_train)

y_pred_2 = classifier_2.predict(X_test)

accuracies_2 = dict()

accuracies_2['soft margin SVM'] = accuracy_score(y_test, y_pred_2)
print('Accuracy is:' + str(accuracy_score(y_test, y_pred_2)))
```

The accuracy is 0.8363076923076923

The confusion matrix is shown below:



In conclusion, the accuracy of SVM model using gaussian RBF method in sklearn library is 0.99, the accuracy of soft margin SVM model using sklearn library is 0.8363076923076923, and the accuracy of our hard margin SVM mode using our own code is 0.8356923076923077

As a result, both the logistic regression model and SVM model performed pretty good.

Discussion

We can see the accuracy for gaussian RBF SVM model is 0.99, which means overfitting happens in the model, and the model isn't usable.

Linear SVM is parametric model, but an RBF kernel SVM isn't, so the complexity of the latter grows with the size of the training dataset. We have to keep the kernel matrix around, so the projection into infinite higher dimensional space where the data becomes linearly separable is more complex and expensive. Thus, it is much easier to overfit a complex model.

Model performance：

	Logistic Regression	SVM
Accuracy	0.8166153846153846	0.8356923076923077
Recall	0.7745740498034076	0.7748756218905473
Precision	0.8242677824267782	0.8787023977433004

Above all, the best model for this problem is the hard margin SVM model, because and it's easier to use and it consumes less resources. And complex model may be more expensive and costs more time.