Mushroom Classification

1. Description of the problem:

We want to classify a mushroom into two groups: edible or poisonous. The dataset contains 7312 observations with 21 attributes and 1 class variable. The class variable is binary with poisonous (coded as p) and edible instances (coded as e): 3766 and 3546, respectively. The class variable follows a binomial distribution with the probability of being edible of 0.485. There are no missing values in the dataset (Table 1). The 21 attributes associated with mushroom and the levels for each attribute are illustrated in Table 1.

|  |  |  |
| --- | --- | --- |
| Attributes | The number of missing values | The levels of factor |
| cap-shape | 0 | 6 |
| cap-surface | 0 | 4 |
| cap-color | 0 | 10 |
| bruises | 0 | 2 |
| odor | 0 | 9 |
| gill-attachment | 0 | 2 |
| gill-spacing | 0 | 2 |
| gill-size | 0 | 2 |
| gill-color | 0 | 12 |
| stalk-shape | 0 | 2 |
| stalk-surface-above- | 0 | 4 |
| stalk-surface-below- | 0 | 4 |
| stalk-color-above-ri | 0 | 9 |
| stalk-color-below-ri | 0 | 9 |
| veil-type | 0 | 1 |
| veil-color | 0 | 4 |
| ring-number | 0 | 3 |
| ring-type | 0 | 5 |
| spore-print-color | 0 | 9 |
| population | 0 | 6 |
| habitat | 0 | 7 |
| Class | 0 | 2 |

Table 1. Table of attributes and the number of missing values and the number of factors for each attribute

1. Construction and tuning the classifier

Support Vector Machine (SVM) is used to analyze the data. In SVM, a hyperplane is selected to best separate the attributes by their class, either class p or class e. The distance between the hyperplane and the closet data points is referred to as the margin. The best hyperplane is that can separate the two classes with the largest margin.

But real data is messy and we cannot separate class data perfectly with a hyperplane. Thus, we will relax the constraint of maximizing the margin of the hyperplane. A tuning parameter C is introduced to define the amount of violation of the margin allowed. The larger the C value, the more complex the hyperplane, which permit more violations to classify all train data correctly. In this project, we will tune C parameters from selected values: 1, 10, 100 and 500.

The SVM algorithm is implemented using a kernel. The kernel used in training and predicting in this project can be one of the following:

* linear: \langle x, x'\rangle.
* polynomial: (\gamma \langle x, x'\rangle + r)^d. d is specified by keyword degree, r by coefficient. We will choose proper degree from 2, 3, 4, 5 and 6; we need choose c from 1, 10, 100, 500 and 1000.
* Radial Basis Function(rbf): \exp(-\gamma |x-x'|^2). \gamma is specified by keyword gamma, must be greater than 0.

For polynomial kernel SVM, we will tune degree parameter from a selection of 2, 3, 4, 5, and 6. As for radial kernel SVM, we tune gamma parameter by selecting from 1.0, 0.01, 0.001 and 0.0001. Gamma defines how influential a single training example can be.

We split data into random train and test subsets. The proportion of the test dataset is set to 0.33, so among 7312 observations, the train dataset contains 4899 observations the test dataset contains 2413 observations. The best model is chosen by fitting SVM on train dataset. The selection criteria is the averaged accuracy of 10-fold cross-validation. By running the following python code, we generate the performance of all kernels of our selection:

tuned\_parameters = {

'C': [1, 10, 100,500, 1000], 'kernel': ['linear','rbf'],

'C': [1, 10, 100,500, 1000], 'gamma': [1,0.1,0.01,0.001, 0.0001], 'kernel': ['rbf'],

'degree': [2,3,4,5,6] , 'C':[1,10,100,500,1000] , 'kernel':['poly']

}

model\_svm = RandomizedSearchCV(svm\_model, tuned\_parameters,cv=10,scoring='accuracy',n\_iter=20)

[mean: 0.50970, std: 0.00051, params: {'kernel': 'poly', 'C': 1000, 'degree': 6, 'gamma': 0.0001}, mean: 0.84038, std: 0.01768, params: {'kernel': 'poly', 'C': 10, 'degree': 6, 'gamma': 1}, mean: 0.90570, std: 0.00892, params: {'kernel': 'poly', 'C': 1, 'degree': 3, 'gamma': 0.01}, mean: 0.61829, std: 0.01599, params: {'kernel': 'poly', 'C': 10, 'degree': 5, 'gamma': 0.01}, mean: 0.90284, std: 0.01290, params: {'kernel': 'poly', 'C': 500, 'degree': 3, 'gamma': 0.1}, mean: 0.50970, std: 0.00051, params: {'kernel': 'poly', 'C': 500, 'degree': 5, 'gamma': 0.0001}, mean: 0.93529, std: 0.00727, params: {'kernel': 'poly', 'C': 500, 'degree': 3, 'gamma': 0.01}, mean: 0.53419, std: 0.00959, params: {'kernel': 'poly', 'C': 1, 'degree': 5, 'gamma': 0.01}, mean: 0.70565, std: 0.02371, params: {'kernel': 'poly', 'C': 500, 'degree': 3, 'gamma': 0.001}, mean: 0.50970, std: 0.00051, params: {'kernel': 'poly', 'C': 10, 'degree': 3, 'gamma': 0.0001}, mean: 0.93468, std: 0.00777, params: {'kernel': 'poly', 'C': 1, 'degree': 3, 'gamma': 0.1}, mean: 0.84303, std: 0.02106, params: {'kernel': 'poly', 'C': 1000, 'degree': 6, 'gamma': 0.1}, mean: 0.50970, std: 0.00051, params: {'kernel': 'poly', 'C': 1, 'degree': 2, 'gamma': 0.0001}, mean: 0.84487, std: 0.01993, params: {'kernel': 'poly', 'C': 500, 'degree': 6, 'gamma': 0.1}, **mean: 0.93611, std: 0.00755, params: {'kernel': 'poly', 'C': 1, 'degree': 2, 'gamma': 0.1}**, mean: 0.83935, std: 0.01906, params: {'kernel': 'poly', 'C': 1000, 'degree': 4, 'gamma': 1}, mean: 0.50970, std: 0.00051, params: {'kernel': 'poly', 'C': 10, 'degree': 5, 'gamma': 0.001}, mean: 0.53419, std: 0.00959, params: {'kernel': 'poly', 'C': 1, 'degree': 6, 'gamma': 0.01}, mean: 0.93468, std: 0.00777, params: {'kernel': 'poly', 'C': 1000, 'degree': 3, 'gamma': 0.01}, mean: 0.92815, std: 0.00814, params: {'kernel': 'poly', 'C': 1, 'degree': 4, 'gamma': 0.1}]

The best SVM is polynomial SVM with the C of 1, the degree of 2 and the gamma of 0.1. It achieves the highest accuracy of 10-fold cross-validation: 0.936. Thus, this polynomial SVM will be selected to predict test data set.

1. Testing results

Polynomial SVM with C:1, degree:2, and gamma:0.1 is implemented to the test dataset. The accuracy of prediction is 0.94. The confusion matrix is illustrated in Table 2. From Figure 1, the AUC of 0.94, much better than random guess with AUC of 0.5 and close to 1, indicates SVM performs well in classifying mushrooms.

|  |  |
| --- | --- |
| 1059 | 92 |
| 51 | 1211 |

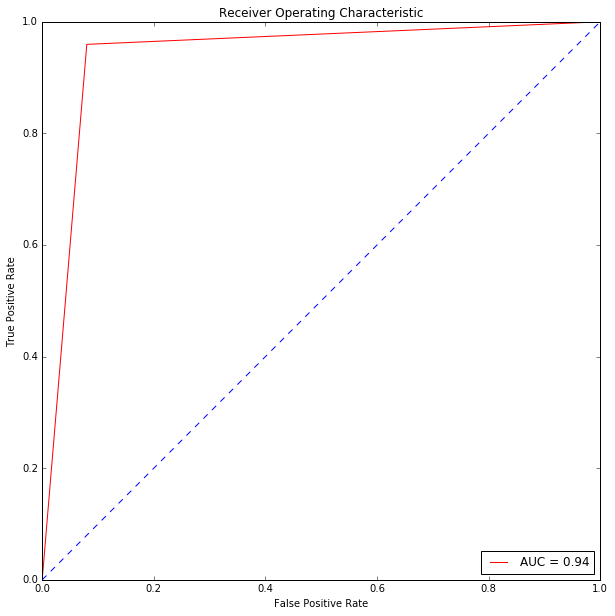
Table 2. Confusion matrix of predicted results using SVM 

Figure 1. AUC of predicted results using SVM

1. Discussion:

Decision Tree is also implemented to classify mushrooms into two groups using analytical software R. The dataset is partitioned into train and test data with the proportions of 66% and 33%, respectively. The first step is to reduce the dimension of dataset using Principle Component Analysis. From Figure 2, 12 principle components will be used because they already explain more than 90% of variation in the input variables. A tree is constructed to classify the target variable (Figure 3).

The accuracy of predicting using decision tree is 0.89. The confusion matrix is illustrated in Table 3. This accuracy is lower than the accuracy of predicting using SVM: 0.94. SVM performs better than decision tree in classifying mushrooms into groups.

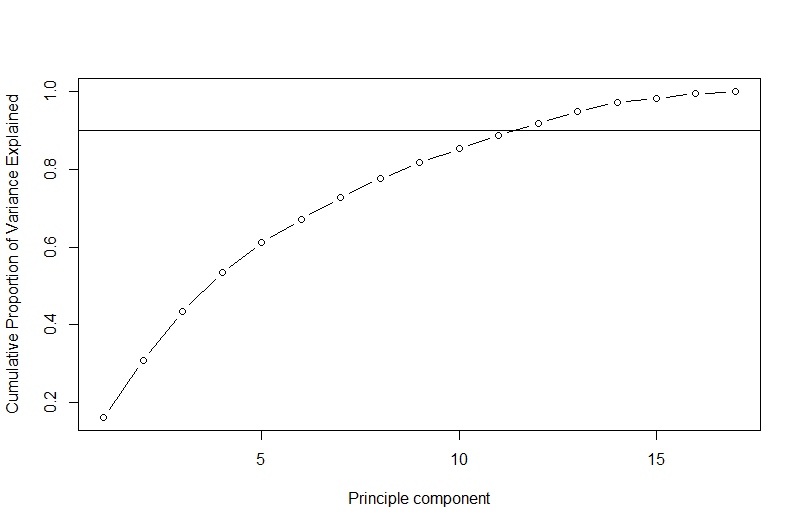


Figure 2: Cumulative proportion of variance explained versus the number of principle components

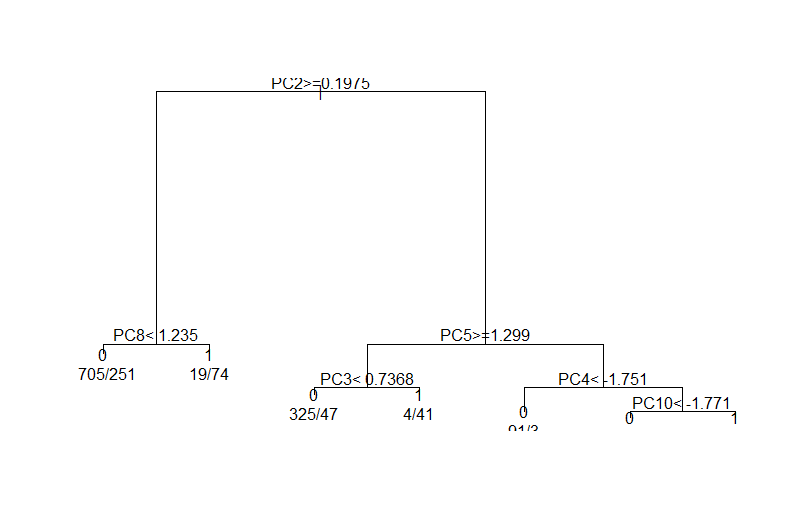


Figure 3: Decision Tree based on principle components

|  |  |
| --- | --- |
| 1089 | 103 |
| 167 | 1128 |

Table 3. Confusion matrix of decision tree

Appendix: Python Code for 1-3.

#---------------------------------Import library-------------------------------

import pandas as pd

import numpy as np

import sklearn as sk

from sklearn.preprocessing import LabelEncoder

from sklearn.cross\_validation import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn import metrics

from sklearn.grid\_search import RandomizedSearchCV

#-----------------------------------Read data----------------------------------

data = pd.read\_csv(‘~/mushroom\_training.csv', sep=',', header=0,

error\_bad\_lines=False, warn\_bad\_lines=True, low\_memory=False)

#-----------------------------------Explore data-------------------------------

print(data.head(2))

#check if there is any null values

print(data.isnull().sum())

#we have two claasification. Either the mushroom is poisonous or edible

print(data.shape)

#----------------------------------Prepare data--------------------------------

labelencoder=LabelEncoder()

for col in data.columns:

data[col] = labelencoder.fit\_transform(data[col])

print(data.head())

#Separating features and label

X = data.iloc[:,0:21] # input variable

y = data.iloc[:, -1] # target variable

#--------------------------------Split data-------------------------------------

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33)

print(X\_train.shape)

# Scale the data to be between -1 and 1

scaler = StandardScaler()

X\_train =scaler.fit\_transform(X\_train)

X\_test = scaler.fit\_transform(X\_test)

#--------------------------Tune model------------------------------------------

svm\_model= SVC()

tuned\_parameters = {

'C': [1, 10, 100,500, 1000], 'kernel': ['linear','rbf'],

'C': [1, 10, 100,500, 1000], 'gamma': [1,0.1,0.01,0.001, 0.0001], 'kernel': ['rbf'],

'degree': [2,3,4,5,6] , 'C':[1,10,100,500,1000] , 'kernel':['poly']

}

model\_svm = RandomizedSearchCV(svm\_model, tuned\_parameters,cv=10,scoring='accuracy',n\_iter=20)

model\_svm.fit(X\_train, y\_train)

print(model\_svm.best\_score\_)

print(model\_svm.grid\_scores\_)

print(model\_svm.best\_params\_)

#--------------------------Predict test dataset--------------------------------

# fit on test data set

y\_pred= model\_svm.predict(X\_test)

print(metrics.accuracy\_score(y\_pred,y\_test))

confusion\_matrix=metrics.confusion\_matrix(y\_test,y\_pred)

print(confusion\_matrix)

# calculate AUC

auc\_roc=metrics.classification\_report(y\_test, test\_predict)

print(auc\_roc)

from sklearn.metrics import roc\_curve, auc

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, test\_predict)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

roc\_auc

# plot AUC

import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))

plt.title('Receiver Operating Characteristic')

plt.plot(false\_positive\_rate,true\_positive\_rate, color='red',label = 'AUC = %0.2f' % roc\_auc)

plt.legend(loc = 'lower right')

plt.plot([0, 1], [0, 1],linestyle='--')

plt.axis('tight')

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')