In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
data =  pd.read_csv('car.data', sep=",")
```

In [3]:

```
data.head()
```

Out[3]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [4]:

```
data['vhigh'].describe()
```

Out[4]:

```
count      1727
unique        4
top        high
freq        432
Name: vhigh, dtype: object
```

In [5]:

```
data['vhigh.1'].describe()
```

Out[5]:

```
count      1727
unique        4
top        high
freq        432
Name: vhigh.1, dtype: object
```

In [6]:

```
data['2'].describe()
```

Out[6]:

```
count      1727
unique        4
top       5more
freq        432
Name: 2, dtype: object
```

In [7]:

```
data['2.1'].describe()
```

Out[7]:

```
count     1727
unique       3
top       more
freq       576
Name: 2.1, dtype: object
```

In [8]:

```
data['small'].describe()
```

Out[8]:

```
count     1727
unique       3
top        med
freq       576
Name: small, dtype: object
```

In [9]:

```
data['low'].describe()
```

Out[9]:

```
count     1727
unique       3
top       high
freq       576
Name: low, dtype: object
```

In [10]:

```
data['unacc'].describe()
```

Out[10]:

```
count      1727
unique        4
top       unacc
freq       1209
Name: unacc, dtype: object
```

In [11]:

```
cleanup_nums = {"vhigh":     {"vhigh": 4, "high": 3, "med": 2, "low": 1},
                "vhigh.1": {"vhigh": 4, "high": 3, "med": 2, "low": 1},
                "2": {"2": 2, "3":3, "4":4, "5more":5},
                "2.1": {"2":2, "4":4, "more":5},
                "small":{"small":1,"med":2,"big":3},
                "low":{"low":1, "med":2, "high":3},
                "unacc":{"unacc": 1, "acc": 2, "good": 3, "vgood": 4}}
```

In [12]:

```
df_num=data.replace(cleanup_nums)
```

In [13]:

```
df_num.head()
```

Out[13]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	4	4	2	2	1	2	1
1	4	4	2	2	1	3	1
2	4	4	2	2	2	1	1
3	4	4	2	2	2	2	1
4	4	4	2	2	2	3	1

In [14]:

```
df2 = df_num.rename({'vhigh': 'buying', 'vhigh.1': 'maint', '2': 'doors', '2.1':
'persons', 'small':'lug_boot', 'low': 'safety', 'unacc': 'class'}, axis='column
s')
```

In [15]:

```
df2.head()
```

Out[15]:

	buying	maint	doors	persons	lug_boot	safety	class
0	4	4	2	2	1	2	1
1	4	4	2	2	1	3	1
2	4	4	2	2	2	1	1
3	4	4	2	2	2	2	1
4	4	4	2	2	2	3	1

In [16]:

```
df2["buying"].describe()
```

Out[16]:

```
count    1727.000000
mean        2.499131
std         1.118098
min         1.000000
25%         1.500000
50%         2.000000
75%         3.000000
max         4.000000
Name: buying, dtype: float64
```

In [17]:

```
df2_x=df2.drop(['buying','persons'], axis=1)
```

In [18]:

```
df2_x.head()
```

Out[18]:

	maint	doors	lug_boot	safety	class
0	4	2	1	2	1
1	4	2	1	3	1
2	4	2	2	1	1
3	4	2	2	2	1
4	4	2	2	3	1

In [19]:

```
df2_y=df2[["buying"]]
```

In [20]:

```
df2_y.head()
```

Out[20]:

	buying
0	4
1	4
2	4
3	4
4	4

In [21]:

```
from sklearn import model_selection

x_train, x_test, y_train, y_test = model_selection.train_test_split(df2_x, df2_y
, test_size = 0.5, random_state = 610)
```

Decision Tree

In [22]:

```
from sklearn import tree
```

In [23]:

```
# Fit a decision tree classifier
dt_estimator = tree.DecisionTreeClassifier(max_depth=2)
dt_estimator.fit(x_train, y_train)
```

Out[23]:

```
DecisionTreeClassifier(max_depth=2)
```

In [24]:

```
y_score = dt_estimator.fit(x_train, y_train)
#y_score = dt_estimator.fit(x_train, y_train).decision_function(x_test)
```

In [25]:

```
y_pred = dt_estimator.predict(x_test)
```

In [26]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
```

In [27]:

```
report = """
The evaluation report of fully grown tree is:
Confusion Matrix:
{}
Accuracy: {}
""".format(confusion_matrix(y_test, y_pred),
           accuracy_score(y_test, y_pred))
print(report)
```

```
The evaluation report of fully grown tree is:
Confusion Matrix:
[[ 39   0  41 122]
 [ 27   0  61 152]
 [  0   0  38 168]
 [  0   0  41 175]]
Accuracy: 0.2916666666666667
```

In [28]:

```
x_test
```

Out[28]:

	maint	doors	lug_boot	safety	class
785	1	3	2	1	1
1212	1	2	3	2	3
1278	1	5	1	2	2
1234	1	3	1	3	3
1597	2	5	2	3	1
...	...	...	...	...	...
578	3	3	2	1	1
1683	1	4	1	2	2
92	4	5	2	1	1
472	4	3	2	3	1
944	4	5	1	1	1

864 rows × 5 columns

In [29]:

```
data_para = {'maint': [4], 'doors': [4], 'lug_boot': [3], 'safety':[3], 'class':
[3]}
```

In [30]:

```
x_test_para= pd.DataFrame(data_para)
```

In [31]:

```
x_test_para
```

Out[31]:

	maint	doors	lug_boot	safety	class
0	4	4	3	3	3

In [32]:

```
y_pred_para = dt_estimator.predict(x_test_para)
```

In [33]:

```
y_pred_para
```

Out[33]:

```
array([1])
```

The logistic regression model predicted the price of the car to be "low" (buying column value of 1) with the given parameters.

Logistic Regression

In [34]:

```
from sklearn import linear_model
from sklearn import metrics
```

In [35]:

```
ovr_estimator = linear_model.LogisticRegression(
    solver = 'lbfgs',
    multi_class = 'ovr')
ovr_estimator.fit(x_train, y_train)

ovr_predict = ovr_estimator.predict(x_test)

ovr_report = """
The evaluation report of OVR is:
Confusion Matrix:
{}
Accuracy: {}
""".format(metrics.confusion_matrix(y_test, ovr_predict),
           metrics.accuracy_score(y_test, ovr_predict))
print(ovr_report)
print('The classification report of OVR:\n {}'
      .format(metrics.classification_report(y_test, ovr_predict)))
```

```
The evaluation report of OVR is:
Confusion Matrix:
[[102   0  33  67]
 [120   0  38  82]
 [ 52   2  43 109]
 [ 49   2  57 108]]
Accuracy: 0.29282407407407407

The classification report of OVR:
               precision    recall  f1-score   support

           1       0.32      0.50      0.39       202
           2       0.00      0.00      0.00       240
           3       0.25      0.21      0.23       206
           4       0.30      0.50      0.37       216

    accuracy                           0.29       864
   macro avg       0.22      0.30      0.25       864
weighted avg       0.21      0.29      0.24       864


/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.
py:73: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  return f(**kwargs)
```

In [36]:

```
x_test
```

Out[36]:

	maint	doors	lug_boot	safety	class
785	1	3	2	1	1
1212	1	2	3	2	3
1278	1	5	1	2	2
1234	1	3	1	3	3
1597	2	5	2	3	1
...	...	...	...	...	...
578	3	3	2	1	1
1683	1	4	1	2	2
92	4	5	2	1	1
472	4	3	2	3	1
944	4	5	1	1	1

864 rows × 5 columns

In [37]:

```
y_pred
```

Out[37]:

```
array([4, 1, 3, 1, 4, 4, 4, 4, 4, 3, 4, 4, 3, 1, 1, 1, 4, 4, 4, 4,
4, 4,
       3, 4, 4, 4, 1, 4, 4, 1, 3, 1, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4,
3, 4,
       4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 1, 3, 3, 4,
1, 4,
       1, 4, 4, 1, 4, 4, 3, 4, 1, 4, 3, 4, 4, 1, 4, 3, 4, 4, 4, 4,
4, 3,
       4, 4, 4, 4, 4, 3, 4, 3, 4, 1, 4, 4, 4, 1, 4, 4, 1, 4, 4, 4,
4, 4,
       3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 3, 4, 4, 4, 4, 4, 1, 4,
4, 4,
       3, 3, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 1, 4, 4,
3, 4,
       4, 3, 4, 1, 4, 4, 3, 3, 4, 4, 3, 3, 4, 4, 4, 4, 3, 3, 4, 3,
4, 4,
       4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 3, 1, 4, 4, 1,
4, 4,
       4, 1, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 3, 3, 4, 4, 1, 4, 4,
4, 3,
       4, 3, 4, 4, 1, 1, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3,
4, 4,
       4, 3, 3, 4, 4, 4, 4, 3, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 1, 4,
4, 4,
       3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 3, 4, 3, 4, 4, 4,
4, 4,
       4, 4, 4, 3, 4, 4, 3, 4, 4, 3, 3, 4, 4, 4, 4, 1, 4, 4, 4, 3,
4, 4,
       4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 3,
4, 4,
       3, 4, 4, 4, 4, 3, 4, 4, 3, 4, 4, 3, 4, 4, 3, 3, 3, 4, 1, 4,
4, 4,
       4, 4, 4, 3, 1, 3, 1, 4, 4, 4, 1, 4, 4, 4, 3, 3, 4, 4, 4, 3,
4, 3,
       3, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 3,
4, 4,
       3, 4, 4, 4, 3, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
       4, 3, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 1, 4, 4, 3, 4, 4,
4, 1,
       4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 3, 3, 4, 4, 3, 4, 4,
4, 1,
       4, 4, 4, 4, 4, 3, 4, 1, 4, 4, 4, 3, 4, 1, 4, 4, 3, 4, 4, 4,
4, 3,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 3,
4, 4,
       4, 4, 4, 4, 4, 4, 4, 3, 4, 3, 3, 3, 4, 4, 3, 4, 4, 3, 4, 4,
1, 4,
       4, 3, 4, 4, 3, 3, 4, 4, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 3, 3,
4, 3,
       4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 3, 1, 4, 4, 3, 4, 4, 4, 4, 4,
3, 4,
       4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 4, 4, 4, 4,
4, 3,
       3, 4, 4, 1, 4, 4, 4, 4, 1, 4, 3, 3, 3, 4, 4, 1, 1, 4, 4, 4,
4, 4,
       4, 4, 3, 4, 3, 4, 4, 3, 3, 4, 3, 4, 4, 4, 4, 4, 4, 4, 3, 4,
3, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 3, 4, 4, 4, 4,
```

```
4, 4,
        1, 4, 4, 3, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3,
3, 4,
        4, 3, 3, 4, 1, 4, 4, 4, 4, 3, 4, 4, 3, 3, 1, 3, 3, 4, 3, 4,
3, 4,
        1, 4, 3, 3, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
        4, 3, 3, 3, 1, 4, 4, 4, 4, 3, 3, 3, 4, 4, 3, 4, 4, 4, 4, 3,
4, 3,
        4, 4, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 1, 1, 4, 4, 4, 4, 4,
3, 4,
        4, 4, 4, 1, 4, 4, 4, 4, 4, 3, 4, 3, 4, 1, 4, 4, 4, 4, 3, 4,
1, 4,
        3, 4, 1, 3, 3, 4, 4, 4, 4, 3, 4, 4, 3, 4, 3, 4, 3, 4, 4, 4,
4, 4,
        4, 4, 4, 4, 3, 1, 4, 4, 4, 4, 4, 3, 3, 4, 3, 4, 3, 4, 1, 3,
4, 4,
        4, 4, 3, 3, 4, 4, 1, 4, 4, 3, 4, 3, 4, 4, 4, 3, 3, 4, 4, 4,
1, 1,
        1, 4, 3, 4, 4, 4])
```

In [38]:

```
data_para_2 = {'maint': [4], 'doors': [4], 'lug_boot': [3], 'safety':[3], 'clas
s':[3]}
```

In [39]:

```
x_test_para_2= pd.DataFrame(data_para_2)
```

In [40]:

```
x_test_para_2
```

Out[40]:

	maint	doors	lug_boot	safety	class
0	4	4	3	3	3

In [41]:

```
y_pred_para_2 = ovr_estimator.predict(x_test_para)
```

In [42]:

```
y_pred_para_2
```

Out[42]:

```
array([1])
```

The logistic regression model predicted the price of the car to be "low" (buying column value of 1) with the given parameters.

Multi-class classification using SVM

In [43]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

In [44]:

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle

from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
from sklearn.metrics import roc_auc_score
```

In [45]:

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(df2_x, df2_y
, test_size = 0.5, random_state = 610)
```

In [46]:

```
# Learn to predict each class against the other
classifier = OneVsRestClassifier(svm.SVC(kernel='linear', probability=True,
                                 random_state=0, max_iter=1000))
#ovr_estimator = linear_model.LogisticRegression(
    #solver = 'lbfgs',
    #multi_class = 'ovr')
y_score = classifier.fit(X_train, Y_train).decision_function(X_test)
```

In [47]:

```
y_pred = classifier.predict(X_test)
```

In [48]:

```
y_pred
```

Out[48]:

```
array([2, 1, 1, 1, 4, 4, 4, 4, 2, 4, 2, 4, 1, 1, 1, 1, 2, 4, 4, 4,
1, 1,
       3, 1, 1, 4, 1, 2, 4, 1, 1, 1, 2, 2, 3, 1, 2, 4, 2, 1, 4, 4,
1, 4,
       1, 4, 1, 4, 1, 4, 4, 1, 1, 4, 4, 4, 1, 1, 1, 2, 1, 1, 3, 4,
1, 4,
       1, 1, 4, 1, 4, 4, 3, 1, 1, 4, 1, 1, 1, 1, 1, 1, 4, 2, 1, 1,
4, 1,
       1, 1, 4, 4, 1, 1, 1, 1, 4, 1, 1, 4, 1, 1, 1, 4, 1, 4, 4, 1,
2, 4,
       4, 2, 4, 2, 1, 2, 4, 4, 4, 1, 1, 2, 3, 4, 1, 4, 4, 4, 1, 1,
4, 2,
       3, 4, 4, 4, 1, 1, 2, 1, 4, 1, 4, 4, 1, 2, 3, 4, 1, 1, 4, 4,
1, 2,
       4, 3, 4, 1, 4, 4, 3, 4, 4, 4, 3, 1, 1, 4, 4, 4, 1, 1, 4, 1,
1, 2,
       4, 3, 4, 4, 1, 4, 4, 4, 4, 4, 1, 2, 1, 4, 4, 1, 1, 1, 4, 1,
4, 4,
       4, 1, 4, 4, 1, 4, 3, 4, 4, 4, 4, 1, 1, 3, 3, 4, 4, 1, 2, 4,
2, 3,
       1, 3, 4, 4, 1, 1, 1, 1, 2, 4, 2, 4, 4, 4, 4, 1, 1, 1, 3, 2,
2, 4,
       2, 4, 1, 2, 1, 4, 4, 1, 2, 4, 1, 2, 2, 4, 1, 1, 4, 4, 1, 4,
2, 4,
       3, 1, 4, 4, 4, 2, 1, 1, 4, 1, 4, 4, 1, 4, 1, 1, 3, 4, 4, 4,
4, 4,
       4, 4, 4, 2, 4, 1, 1, 4, 4, 3, 1, 1, 4, 4, 4, 1, 4, 4, 4, 3,
4, 4,
       2, 1, 4, 4, 2, 1, 4, 4, 4, 2, 1, 4, 4, 4, 2, 4, 1, 4, 4, 1,
1, 2,
       1, 4, 2, 4, 4, 3, 4, 2, 1, 4, 4, 4, 4, 1, 1, 1, 3, 1, 1, 4,
4, 4,
       4, 4, 4, 1, 1, 1, 1, 4, 2, 4, 1, 4, 1, 4, 4, 3, 1, 4, 2, 4,
4, 3,
       3, 1, 4, 3, 1, 1, 4, 2, 1, 4, 4, 2, 1, 3, 2, 4, 4, 1, 1, 1,
2, 1,
       1, 4, 1, 4, 1, 1, 2, 1, 4, 1, 1, 4, 2, 2, 4, 4, 2, 1, 4, 2,
4, 1,
       1, 3, 4, 4, 4, 4, 1, 2, 1, 4, 4, 4, 2, 2, 1, 4, 2, 3, 1, 1,
4, 1,
       4, 1, 4, 4, 4, 4, 1, 2, 2, 2, 4, 3, 4, 3, 1, 4, 4, 1, 1, 4,
4, 1,
       1, 4, 4, 4, 1, 1, 2, 1, 2, 4, 4, 3, 1, 1, 4, 4, 3, 4, 2, 4,
1, 1,
       1, 4, 1, 1, 4, 4, 1, 1, 1, 4, 4, 1, 4, 4, 4, 4, 1, 3, 4, 3,
4, 4,
       4, 2, 1, 2, 1, 4, 4, 3, 4, 1, 2, 3, 1, 1, 3, 4, 1, 4, 1, 4,
1, 4,
       4, 1, 4, 1, 1, 1, 4, 4, 4, 4, 1, 1, 1, 4, 2, 2, 4, 1, 3, 1,
4, 1,
       4, 4, 4, 1, 4, 2, 1, 4, 4, 4, 3, 1, 4, 4, 2, 2, 1, 4, 4, 4,
1, 4,
       4, 4, 1, 2, 1, 1, 4, 4, 4, 2, 1, 4, 1, 2, 3, 1, 1, 1, 1, 1,
4, 3,
       1, 1, 4, 1, 2, 2, 4, 4, 1, 4, 4, 1, 1, 4, 1, 1, 1, 2, 1, 4,
1, 4,
       2, 4, 1, 1, 1, 1, 4, 4, 1, 4, 3, 4, 2, 4, 4, 1, 2, 1, 1, 4,
3, 4,
       2, 4, 4, 4, 4, 1, 1, 4, 1, 4, 4, 1, 4, 2, 4, 1, 1, 1, 1, 4,
```

```
4, 4,
         1, 4, 4, 1, 1, 1, 4, 1, 1, 4, 1, 1, 4, 2, 1, 2, 4, 4, 1, 3,
1, 4,
         1, 1, 4, 1, 1, 4, 4, 4, 4, 3, 4, 1, 1, 3, 1, 4, 2, 4, 1, 4,
1, 4,
         1, 1, 3, 2, 4, 1, 1, 4, 4, 4, 1, 4, 4, 4, 1, 1, 2, 4, 4, 1,
2, 2,
         2, 1, 1, 1, 1, 4, 1, 4, 4, 1, 3, 1, 4, 4, 1, 2, 4, 4, 1, 1,
4, 4,
         4, 4, 4, 2, 4, 3, 1, 4, 1, 4, 4, 4, 2, 1, 1, 2, 4, 4, 4, 1,
3, 1,
         1, 4, 4, 1, 2, 4, 4, 2, 4, 1, 4, 3, 4, 1, 4, 1, 2, 1, 3, 1,
1, 4,
         1, 1, 1, 1, 3, 4, 4, 4, 1, 4, 4, 2, 4, 4, 1, 4, 3, 4, 1, 4,
4, 4,
         4, 1, 4, 4, 1, 1, 1, 1, 1, 4, 1, 1, 1, 4, 1, 1, 1, 4, 1, 3,
1, 4,
         4, 4, 1, 3, 2, 2, 1, 4, 4, 3, 4, 3, 4, 4, 1, 1, 1, 1, 4, 4,
1, 1,
         1, 1, 1, 1, 4, 1])
```

In [49]:

```
ovr_report = """
The evaluation report of SVM is:
Confusion Matrix:
{}
Accuracy: {}
""".format(metrics.confusion_matrix(y_test, y_pred),
           metrics.accuracy_score(y_test, y_pred))
print(ovr_report)
print('The classification report of SVM:\n {}'
      .format(metrics.classification_report(y_test, y_pred)))
```

```
The evaluation report of SVM is:
Confusion Matrix:
[[110  21   9  62]
 [111  31  16  82]
 [ 58  19  19 110]
 [ 50  30  16 120]]
Accuracy: 0.32407407407407407

The classification report of SVM:
               precision    recall  f1-score   support

           1       0.33      0.54      0.41       202
           2       0.31      0.13      0.18       240
           3       0.32      0.09      0.14       206
           4       0.32      0.56      0.41       216

    accuracy                           0.32       864
   macro avg       0.32      0.33      0.29       864
weighted avg       0.32      0.32      0.28       864
```

In [50]:

```
data_para_3 = {'maint': [4], 'doors': [4], 'lug_boot': [3], 'safety':[3], 'clas
s':[3]}
```

In [51]:

```
x_test_para_3= pd.DataFrame(data_para_2)
```

In [52]:

```
x_test_para_3
```

Out[52]:

	maint	doors	lug_boot	safety	class
0	4	4	3	3	3

In [53]:

```
y_pred_para_3 = classifier.predict(x_test_para)
```

In [54]:

```
y_pred_para_3
```

Out[54]:

```
array([1])
```

The logistic regression model predicted the price of the car to be "low" (buying column value of 1) with the given parameters.

In [ ]:

```
```

ROC Curve

In [55]:

```
X=df2_x.to_numpy()
Y=df2_y.to_numpy()

# Binarize the output
Y = label_binarize(Y, classes=[1, 2, 3,4])
n_classes = Y.shape[1]
```

In [56]:

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_s
ize = 0.5, random_state = 610)
```

In [57]:

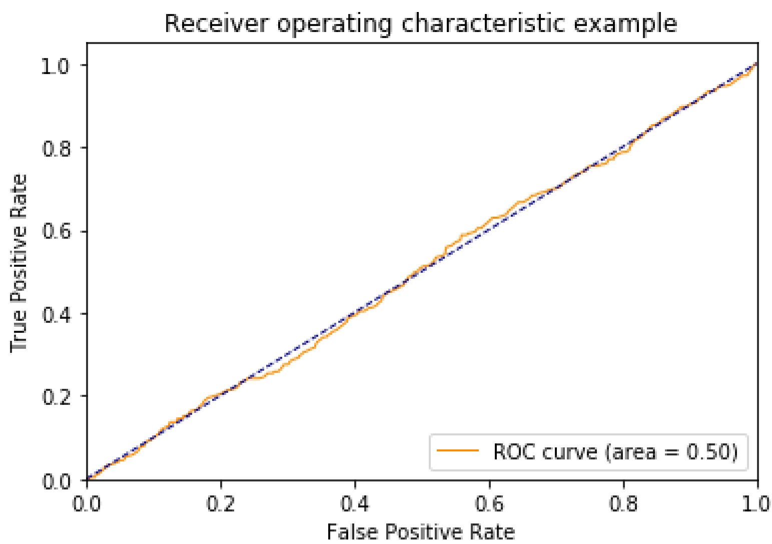
```
# Learn to predict each class against the other
classifier = OneVsRestClassifier(svm.SVC(kernel='linear', probability=True,
                                 random_state=0, max_iter=1000))
#ovr_estimator = linear_model.LogisticRegression(
    #solver = 'lbfgs',
    #multi_class = 'ovr')
y_score = classifier.fit(X_train, Y_train).decision_function(X_test)
```

In [58]:

```
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(Y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(Y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

In [59]:

```
plt.figure()
lw = 1
plt.plot(fpr[1], tpr[1], color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[1])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

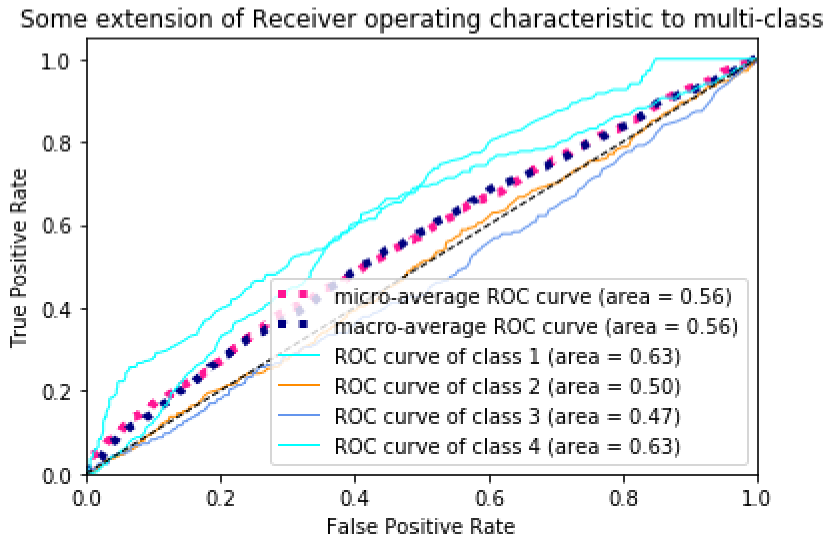


Plot 4 ROC curves in one plot

In [60]:

```
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i+1, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
DeprecationWarning: scipy.interp is deprecated and will be removed i
n SciPy 2.0.0, use numpy.interp instead
```



ROC curve of class 1 gives a value of 0.63

In [ ]: