In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('feature.csv')
```

In [4]:
```
data.head(2)
```

Out[4]:

	Mean	Standard_Deviation	Variance	label
0	-0.76913	5.5906	31.255	1
1	-0.77444	5.5941	31.293	1

In [ ]:

In [5]:
```
data.keys()
```

Out[5]:
```
Index(['Mean', 'Standard_Deviation', 'Variance', 'label'], dtype='object')
```

In [6]:
```
data_feat=data[['Mean', 'Standard_Deviation', 'Variance']]
```

In [7]:
```
data_feat.head(2)
```

Out[7]:

	Mean	Standard_Deviation	Variance
0	-0.76913	5.5906	31.255
1	-0.77444	5.5941	31.293

In [8]:
```
class_names=['Port 1', 'Port 2']
```

In [9]:
```
type(class_names)
```

Out[9]:
```
list
```

In [10]:
```
data_label=data[['label']]
```

In [11]:
```
data_label.head(2)
```

Out[11]:

	label
0	1
1	1

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
X=data_feat;
y=data_label
X_train, X_test, y_train, y_test = train_test_split(X, np.ravel(y), test_size=0.30, random_sta
```

In [14]:
```
from sklearn.svm import SVC
```

In [15]:
```
model=SVC()
```

In [16]:
```
model.fit(X_train, y_train)
```

Out[16]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [17]:
```python
predictions = model.predict(X_test)
```

In [18]:
```python
from sklearn.metrics import classification_report,confusion_matrix
```

In [19]:
```python
print(confusion_matrix(y_test,predictions))
```

```
[[144   0]
 [  0 156]]
```

In [20]:
```python
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          1       1.00      1.00      1.00       144
          2       1.00      1.00      1.00       156

avg / total       1.00      1.00      1.00       300
```

In [21]:
```python
from sklearn.grid_search import GridSearchCV
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: T
his module was deprecated in version 0.18 in favor of the model_selection module into which
all the refactored classes and functions are moved. Also note that the interface of the new
CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/anaconda3/lib/python3.6/site-packages/sklearn/grid_search.py:42: DeprecationWarning: This m
odule was deprecated in version 0.18 in favor of the model_selection module into which all t
he refactored classes and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [22]:
```python
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf
```

In [23]:
```python
grid= GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
```

In [24]:
```python
grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ............. C=0.1, gamma=1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ............. C=0.1, gamma=1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ............. C=0.1, gamma=1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........... C=0.1, gamma=0.1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........... C=0.1, gamma=0.1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........... C=0.1, gamma=0.1, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] .......... C=0.1, gamma=0.01, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] .......... C=0.1, gamma=0.01, kernel=rbf, score=1.000000 -   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] .......... C=0.1, gamma=0.01, kernel=rbf, score=1.000000 -   0.0s
```

In [25]:
```python
grid.best_params_
```

Out[25]:
```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

In [26]:
```python
grid.best_estimator_
```

Out[26]:
```
SVC(C=0.1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [27]:
```python
grid_predictions = grid.predict(X_test)
```

In [28]:
```python
grid_predictions
```

Out[28]:
```
array([2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,
       1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1,
       1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1,
       1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2,
       1, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2, 1, 1, 1, 2,
       2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1,
       2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2,
       2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1,
       1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2,
       1, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1,
       2, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1])
```

In [29]:
```python
print(confusion_matrix(y_test,grid_predictions))
```

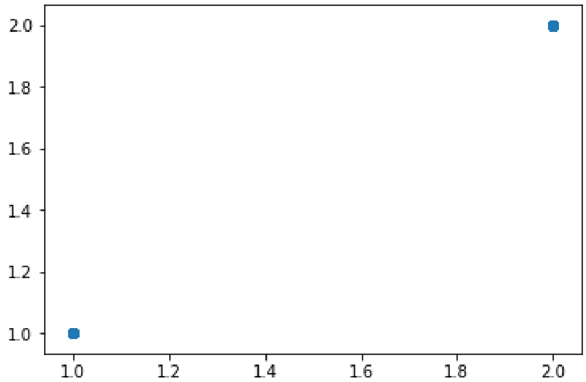
```
[[144   0]
 [  0 156]]
```

In [30]:
```python
print(classification_report(y_test, grid_predictions))
```

```
             precision    recall  f1-score   support

          1       1.00      1.00      1.00       144
          2       1.00      1.00      1.00       156

avg / total       1.00      1.00      1.00       300
```

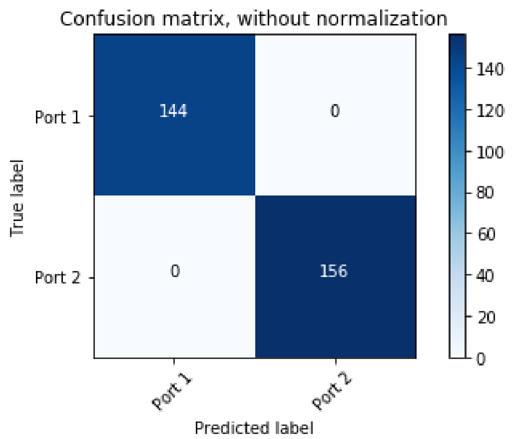
In [31]:
```python
plt.scatter(y_test, grid_predictions)
```

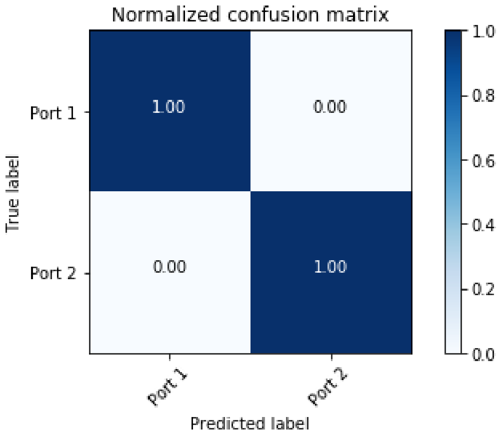
Out[31]:
```
<matplotlib.collections.PathCollection at 0x1a13ff3080>
```



In [58]:
```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, grid_predictions)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
beingsaved = plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

beingsaved.savefig('destination_path.png', format='png', dpi=1000)
# Plot normalized confusion matrix
beingsaved1 = plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
beingsaved1.savefig('destination_path1.png', format='png', dpi=1000)
plt.show()
```

```
Confusion matrix, without normalization
[[144   0]
 [  0 156]]
Normalized confusion matrix
[[1. 0.]
 [0. 1.]]
```





Normalized confusion matrix
1.00
0.00
0.00
1.00
Port 1
Port 2
True label
Port 1
Port 2
Predicted label
1.0
0.8
0.6
0.4
0.2
0.0

In [ ]:

In [ ]: