In [1]:
```
import collections
import itertools

import numpy as np
import pandas as pd
import scipy.spatial as sp
import scipy.cluster.hierarchy as hc
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import matplotlib.colors as colors
import matplotlib.cm as cmx
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classificatio
```

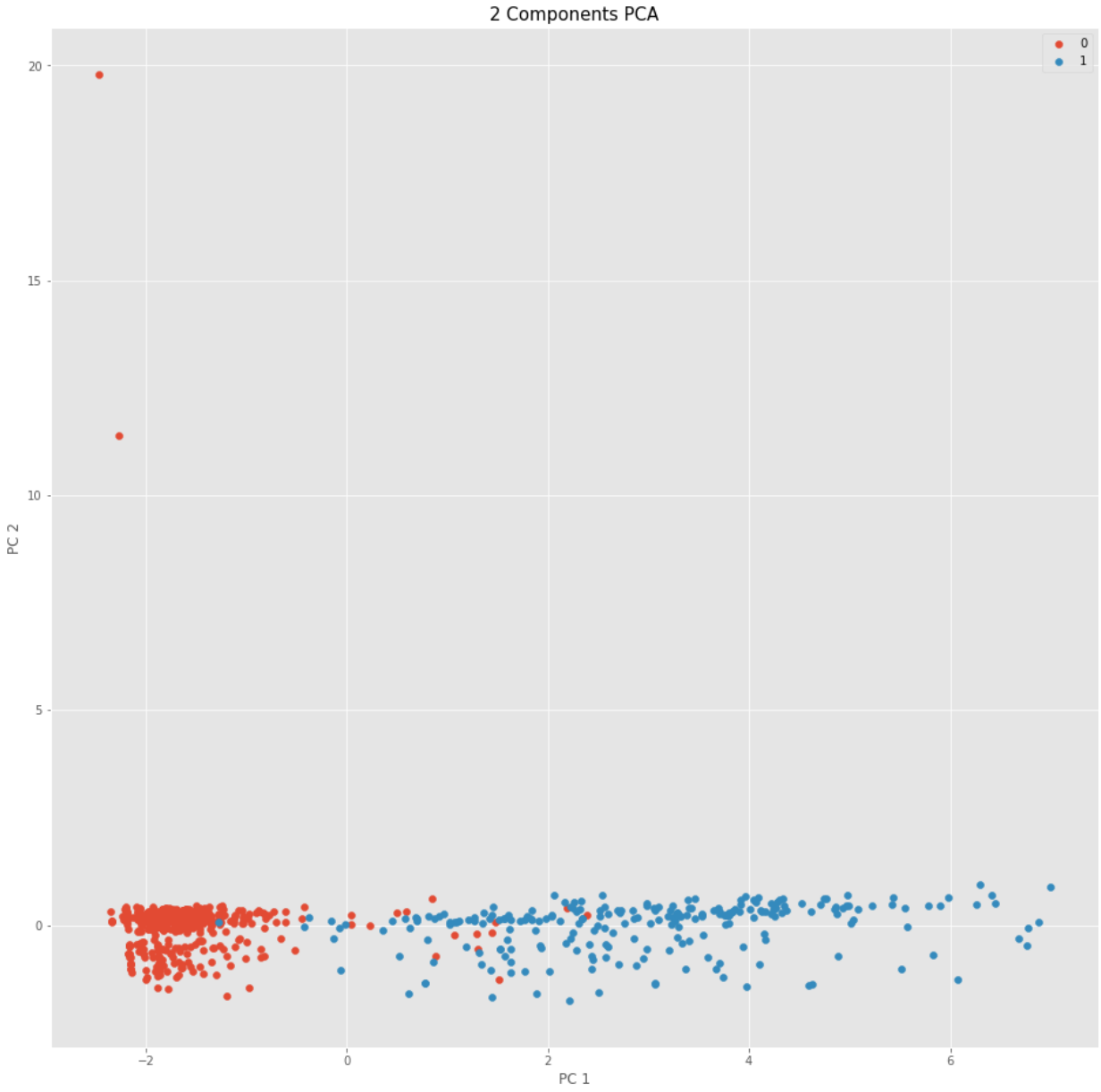
In [2]:
```
_colors(categorical):

 list(set(categorical))
rical_to_number = pd.factorize(categorical)[0]

 plt.get_cmap('brg') # https://matplotlib.org/tutorials/colors/colormaps.htm
 = colors.Normalize(vmin=0, vmax=len(uniq))
Map = cmx.ScalarMappable(norm=cnorm, cmap=cmap)
map = [scalarMap.to_rgba(i) for i in categorical_to_number]

 color_map


m_pca(X, y, perc, visualize=True):
ls = StandardScaler().fit_transform(X)

PCA(perc)
pal_comps = pca.fit_transform(std_vals)

pal_df = pd.DataFrame(data=principal_comps)

'Number of principal components explain {}% variance of data is {}'.format(p

ualize:
g = plt.figure(figsize=(16, 16))
 = fig.add_subplot(1, 1, 1)
.set_xlabel('PC 1', fontsize=12)
.set_ylabel('PC 2', fontsize=12)
.set_title('2 Components PCA', fontsize=15)

r group in set(y):
  idx = np.where(y == group)[0]
  ax.scatter(principal_df.loc[idx, 0], principal_df.loc[idx, 1], label=group

.legend()
.grid(True)

t.show()

t(pca.explained_variance_ratio_)
t(principal_df.values)

rn pca.explained_variance_ratio_, principal_df.values

 None

p(X, y, row_linkage, col_linkage):
s.clustermap(
 figsize=(16,16), z_score=0,
w_linkage=feature_linkage, col_linkage=sample_linkage,
l_colors=assign_colors(y)


ow()

ibsvm_format(url, classA, classB):
```

```
d.read_csv(url, sep=' ', header=None)
f.dropna(axis=1, how='all') # drop empty columns
f.dropna() #drop all rows that have any NaN values
 = df.iloc[:,0].values
[np.where(labels==classA)] = 0
[np.where(labels==classB)] = 1

f.iloc[:,1:].applymap(lambda x: x.split(':')[1])
ass'] = labels
f.astype(float)
ass'] = df['class'].astype(int)

 df

onfusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap
malize:
 = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

show(cm, interpolation='nearest', cmap=cmap)
tle(title)
lorbar()
arks = np.arange(len(classes))
icks(tick_marks, classes, rotation=45)
icks(tick_marks, classes)

'.2f' if normalize else 'd'
 = cm.max() / 2.
 j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
t.text(j, i, format(cm[i, j], fmt),
       horizontalalignment='center',
       color='white' if cm[i, j] > thresh else 'black')

abel('True label')
abel('Predicted label')
ght_layout()

(clf, model):
esult = clf.best_score_
'Best classification accuracy of {} models: {}'.format(model, best_result))
arameters = clf.best_params_
'Parameters of the best {} model: {}'.format(model, best_parameters))


fy(X, y, classifier, grid_params, name):
GridSearchCV(classifier, grid_params, scoring='accuracy',
             cv=5, verbose=0, n_jobs=-1) # for detailed log verbose=2
t(X, y)

(clf, name)
```

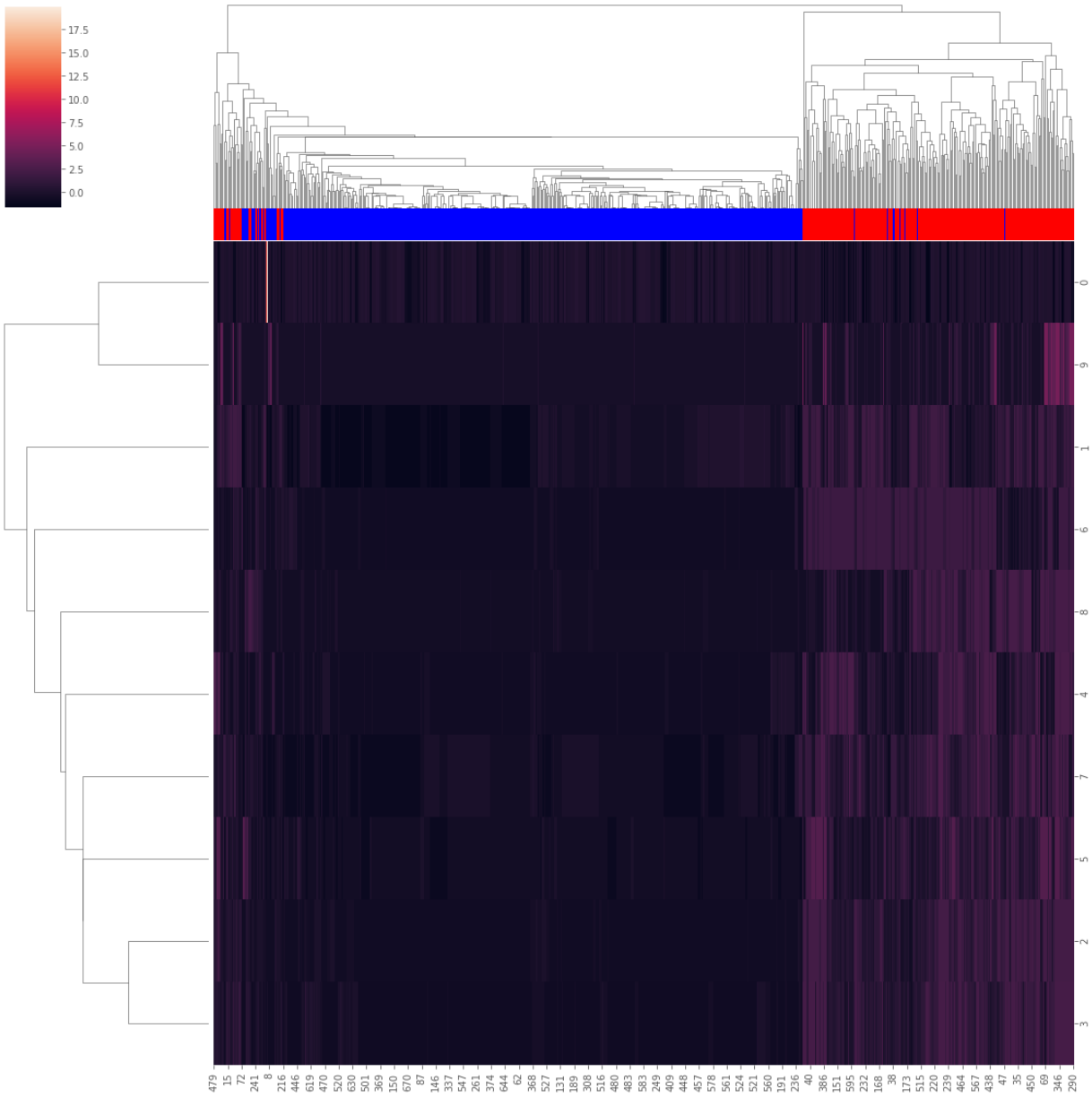
In [3]:
```
breast = 'https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/br
df = read_libsvm_format(breast, 2, 4)
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
```

In [4]:
```
perform_pca(X, y, 0.90)
```

```
Number of principal components explain 90.0% variance of data is 6
```



In [5]:
```
sample_dist = sp.distance.pdist(X, 'euclidean')
feature_dist = sp.distance.pdist(X.T, 'euclidean')

sample_linkage = hc.linkage(sample_dist, method='average')
feature_linkage = hc.linkage(feature_dist, method='average')

heatmap(X.T, y, sample_linkage, feature_linkage)
```



In [6]:
```
svm_param_grid = [
    {
        'C': [1, 10, 100, 1000],
        'class_weight': [None, 'balanced'], 'kernel': ['linear']
    },
    {
        'C': [1, 10, 100, 1000], 'degree': np.arange(1,11),
        'class_weight': [None, 'balanced'], 'kernel': ['poly']
    },
    {
        'C': [1, 10, 100, 1000],
        'class_weight': [None, 'balanced'],
        'kernel': ['rbf', 'sigmoid']
    },
]

svc = svm.SVC(gamma='scale')
classify(X, y, svc, svm_param_grid, 'SVM')
```

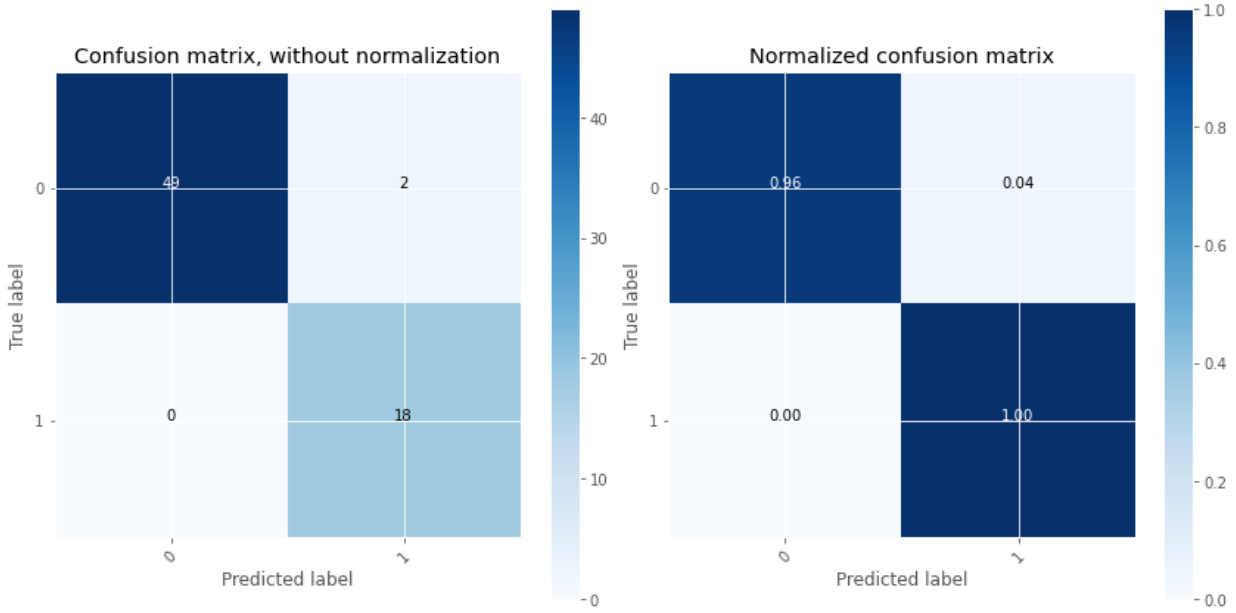
```
Best classification accuracy of SVM models: 0.9736904250751396
Parameters of the best SVM model: {'C': 1, 'class_weight': 'balanced', 'd
egree': 3, 'kernel': 'poly'}
```

In [7]:
```
lor_param_grid = [
    {
        'C':[1, 10, 100, 1000], 'penalty': ['l1'],
        'solver': ['liblinear', 'saga'],
        'class_weight': [None, 'balanced']
    },
    {
        'C':[1, 10, 100, 1000], 'penalty': ['l2'],
        'solver': ['newton-cg', 'lbfgs', 'sag'],
        'class_weight': [None, 'balanced']
    }
]

lor = LogisticRegression()

classify(X, y, lor, lor_param_grid, 'Logistic Regression')
```

```
Best classification accuracy of Logistic Regression models: 0.97077071704
59425
Parameters of the best Logistic Regression model: {'C': 100, 'class_weigh
t': 'balanced', 'penalty': 'l1', 'solver': 'saga'}

/Users/aryantimla/opt/anaconda3/lib/python3.8/site-packages/sklearn/linea
r_model/_sag.py:328: ConvergenceWarning: The max_iter was reached which m
eans the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [8]:
```
clf = svm.SVC(gamma='scale', C=1, class_weight=None, degree=2, kernel='poly
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, ra
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('Accuracy: {}\n'.format(accuracy_score(y_test, y_pred)))

print(classification_report(y_test, y_pred, target_names=[str(i) for i in s

cnf_matrix = confusion_matrix(y_test, y_pred)

plt.subplots(figsize=(12,6))
plt.subplot(1, 2, 1)
plot_confusion_matrix(cnf_matrix, classes=set(y_test),
                      title='Confusion matrix, without normalization')
plt.subplot(1, 2, 2)
plot_confusion_matrix(cnf_matrix, classes=set(y_test), normalize=True,
                      title='Normalized confusion matrix')

plt.tight_layout()
plt.show()
```

```
Accuracy: 0.9710144927536232

              precision    recall  f1-score   support

           0       1.00      0.96      0.98        51
           1       0.90      1.00      0.95        18

    accuracy                           0.97        69
   macro avg       0.95      0.98      0.96        69
weighted avg       0.97      0.97      0.97        69
```



In [ ]: