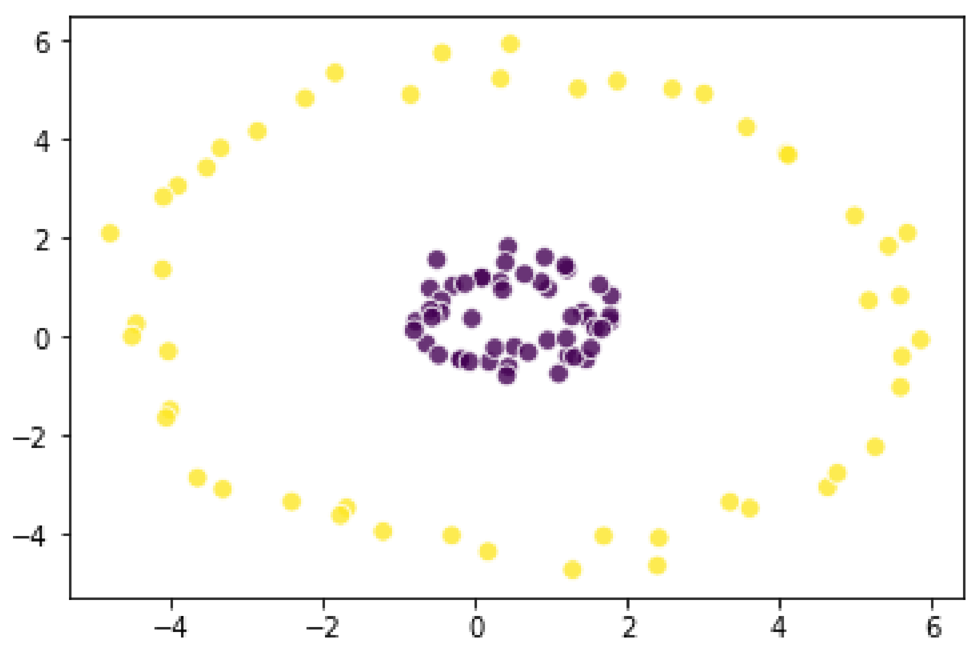
principal_component_analysis

January 19, 2019

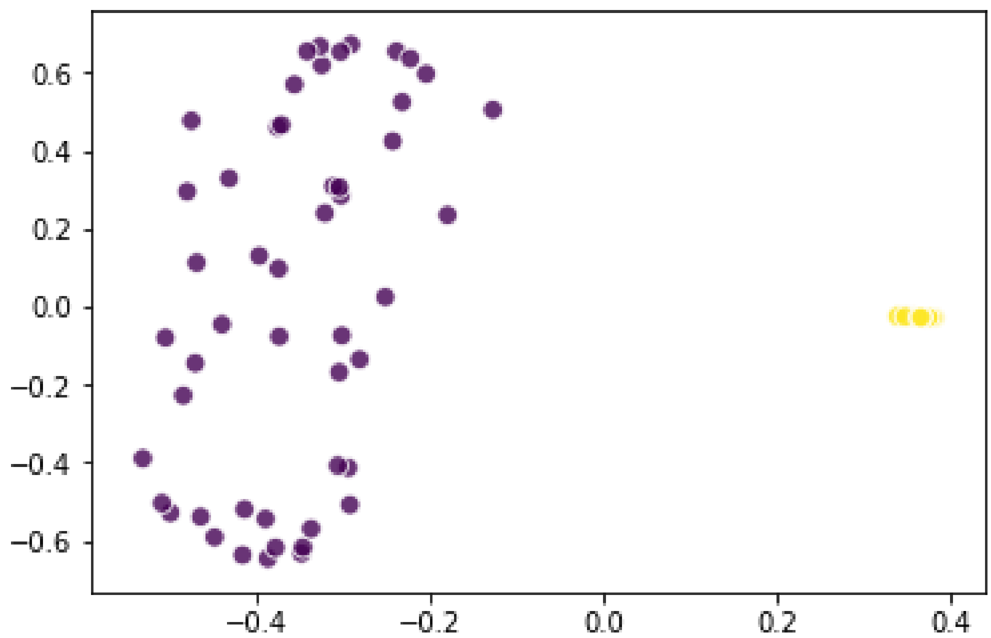
```
In [12]: #PCA - principal component analysis
         import numpy as np
         import matplotlib.pyplot as plt

         def circular_points (radius, N):
             return np.array([[np.cos(2*np.pi*t/N)*radius, np.sin(2*np.pi*t/N)*radius] for t in
         N_points = 50
         fake_circular_data = np.vstack([circular_points(1.0, N_points)
                                        ,circular_points(5.0, N_points)])
         fake_circular_data += np.random.rand(*fake_circular_data.shape)
         fate_circular_target = np.array([0]*N_points + [1]*N_points)
         plt.scatter(fake_circular_data[:,0], fake_circular_data[:,1],
                     c=fate_circular_target, alpha=0.8, s=60,
                     marker='o', edgecolors='white')
         plt.show()
```



In [16]:
```
#PCA - principal component analysis with radial basis function (kernel function)
from sklearn.decomposition import KernelPCA
kpca_2c = KernelPCA(n_components=2, kernel='rbf') # 2 componetns
X_kpca_2c = kpca_2c.fit_transform(fake_circular_data)
plt.scatter(X_kpca_2c[:,0], X_kpca_2c[:,1],
            c=fate_circular_target, alpha=0.8, s=60,
           marker='o', edgecolors='white')
plt.show()

# now it is possible to process data with linear techniques
```



In [ ]: