PCA

Ishita Jaju
16BCE1059

1 About the algorithm

1.1 PCA

Principal Component Analysis (PCA) is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set. This is especially helpful in a dataset like mine, where there are many dimensions. It is also helpful in visualization of large datasets.

1.2 KPCA

KPCA or Kernel PCA is an extension of PCA using kernel method techniques. Using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space.

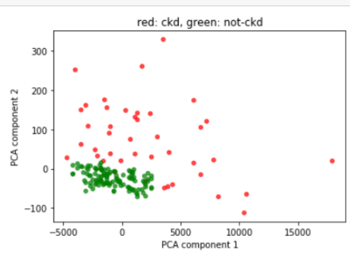
2 Result on my dataset

2.1 PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(ckd)
T = pca.transform(ckd)

T = pd.DataFrame(T)
```

```
label_color = ['red' if i=='ckd' else 'green' for i in targets]
T.columns = ['PCA component 1', 'PCA component 2']
T.plot.scatter(x='PCA component 1', y='PCA component 2', marker='o',
        alpha=0.7, # opacity
        color=label_color,
        title="red: ckd, green: not-ckd" )
plt.show()
```



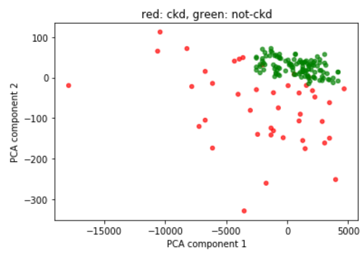
The dataset gets reduced to 2 dimensions from 14 dimensions and the classification process beomes much easier.

2.2 KPCA

```
from sklearn.decomposition import KernelPCA
transformer = KernelPCA(n_components=2, kernel='linear')
Tk = transformer.fit_transform(ckd)
Tk = pd.DataFrame(Tk)
Tk.shape
```

```
(158, 2)
```

```
label_color = ['red' if i=='ckd' else 'green' for i in targets]
Tk.columns = ['PCA component 1', 'PCA component 2']
Tk.plot.scatter(x='PCA component 1', y='PCA component 2', marker='o',
        alpha=0.7, # opacity
        color=label_color,
        title="red: ckd, green: not-ckd" )
plt.show()
```



3 Performance

3.1 Before dimensionality reduction

```
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.725
```

The accuracy comes out to be 72%.

3.2 After PCA

PCA ¶

```
X_train,X_test,y_train,y_test = train_test_split(T,targets)
```

```
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.85
```

The accuracy comes out to be 85%.

3.3 After KPCA

KPCA

```
X_train,X_test,y_train,y_test = train_test_split(Tk,targets)
```

```
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
accuracy_score(y_test,y_pred)
```

```
0.85
```

The accuracy comes out to be 85%.