Spectral Clustering

Shivank Goel, Email : sg2359@cornell.edu

INTRODUCTION

Spectral clustering is an unsupservised learning technique to cluster the the data points (which can represent nodes in a graph) into groups to detect communities, when the data points are not located as the close convex communities in their original space. This paper follows the idea from Ng et. al[1]. The approach uses the top eigen-vectors of a matrix, which is derived from the distance between nodes. This approach is motivated from the non-linear partitioning algorithms to divide the data points into two clusters. One can use *spectral graph partitioning* in which second eigen-vector of a graph's laplacian is used to define a semi-optimal cut. Since partitioning divides the graph into two almost equally sized parts, it can be applied recursively to find the k clusters. However, *spectral clustering* extends the partitioning approach to use k eigen-vectors simultaneously for obtaining the k clusters.

ALGORITHM

Given the set of points[1], $S = \{s_1, s_2, ..., s_n\} \in \mathbb{R}^l$, we obtain an affinity matrix $A \in \mathbb{R}^{n \times n}$, which is defined as $A_{ij} = exp(-\|s_i - s_j\|^2/2\sigma^2)$ if $i \neq j$ and $A_{ii} = 0$. The scaling parameter σ^2 controls how rapidly the affinity A_{ij} falls off with distance between s_i and s_j. Define the diagonal matrix D as, $D_{ii} = \sum_j A_{ij}$, whose i^{th} diagonal element is the sum of A's i^{th} row. Let $x_1, x_2, ..., x_k$ be the largest k eigen-vectors of the matrix[2], $L = D^{-1/2}AD^{-1/2}$. We form the matrix, $X = [x_1, x_2, ..., x_k] \in \mathbb{R}^{n \times k}$, by stacking eigen-vectors in columns of X, and re-normalize each row of X to get $Y_{ij} = X_{ij}/(\sum_j X_{ij}^2)^{-1/2}$. Now we can treat each row of Y as a point in $\mathbb{R}^k$ dimension, and can use clustering algorithms like *K-means* on top of it.

IMPLEMENTATION AND CODE

1. The following code creates the affinity matrix as per Ng. et al[1], for a given value of scaling parameter σ^2.

```
import numpy as np
from numpy.linalg import eigh
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from scipy.io import loadmat

def similarity(a,b,sigma):
    return(np.exp(-(np.sum(np.square(a-b)))/(2*sigma*sigma)))

def getaffinitymatrix(data,sigma=0.1):
        n,l = data.shape
        A = np.zeros((n,n))
        for i in range(n):
                for j in range(n):
                        A[i,j] = similarity(data[i,:],data[j,:],sigma)
        return A
```

2. The following code takes an affinity matrix and uses the spectral clustering algorithm to convert each data point into a low dimensional feature representations (given by k). These features can then be used for the $K - means$ clustering algorithm to detect the clusters.

[1]We represent nodes as their feature vectors in $\mathbb{R}^l$ space, and call them as points.

[2]Although, symmetric normalized laplacian is more popular, $L = I - D^{-1/2}AD^{-1/2}$, but eigenvectors would be same in both cases.

```
def renormalize(Y):
        for i in range(Y.shape[0]):
                Y[i,:] /= np.sqrt(np.sum(np.square(Y[i,:])))
        return Y

def spectralcluster(A,k=1):
        Dinv = np.diag(1/np.sqrt(A.sum(axis=1)))
        L =  Dinv@A@Dinv
        e,v = eigh(L)
        Y = v[:,A.shape[0]-k:]
        Y = renormalize(Y)
        return Y
```

3. To run the $K - means$ algorithm, I am using a pre-defined function in the *Scikit-Learn* package.

```
def give_k_means(X,numc=2):
        from sklearn.cluster import KMeans
        kmeans = KMeans(n_clusters=numc, random_state=0).fit(X)
        return kmeans.labels_
```

TOY DATASET

I will first display the results on a toy dataset which consists two moon-shaped clusters of points, which are not linearly separable. Running the $K - means$ on the co-ordinates of these points give us a linear clustering (clustering which can be linearly separated) and does not capture the moon structure. However once we use spectral clustering, we can easily capture the non-linearity as shown in the Figure 1, followed by the code for the same.

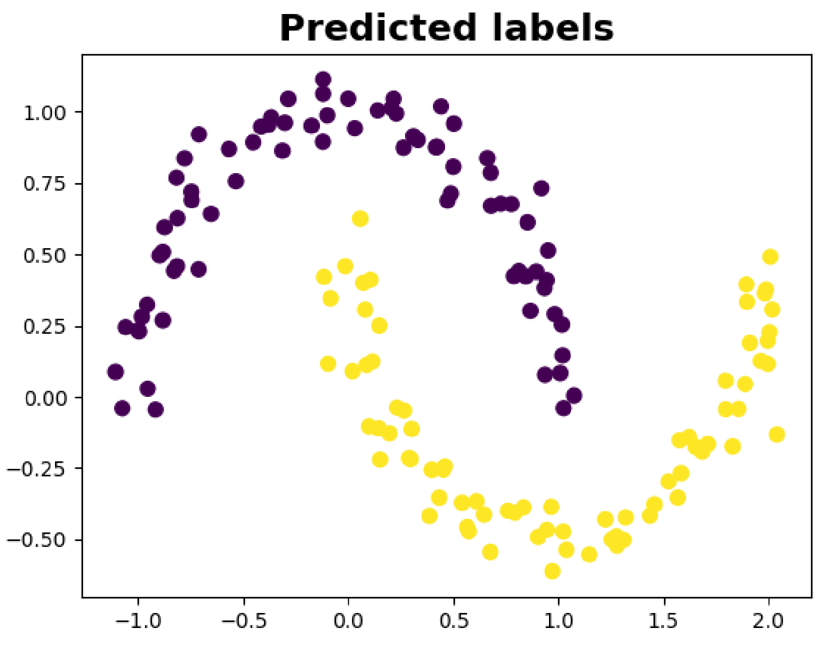

Figure 1. Clustering of the moons dataset

```
X_mn, y_mn = make_moons(150, noise=.07, random_state=21)
A = getaffinitymatrix(X_mn)
y_pred = (give_k_means(spectralcluster(A,2))> 0).reshape(-1,)
plt.title('Data with ground truth labels', fontsize=18, fontweight='demi')
plt.scatter(X_mn[:, 0], X_mn[:, 1], c=y_pred,s=50, cmap='viridis')
plt.show()
```

YALE DATASET

For demonstrating the usefulness of the spectral clustering, I will be using the Yale faces dataset.[3] It consists of 165 gray-scale images in GIF format of 15 individuals. There are 11 images per subject for different facial expressions, and configurations, such as, center-light, with glasses, happy, left-light, with no glasses, normal, right-light, sad, sleepy, surprised, and wink. The following code creates the clusters of photos of same person.

```
from scipy.io import loadmat
from collections import Counter
from sklearn.utils import shuffle

x = loadmat('data/yale.mat')
xtrain = (x['fea'])/255
ytrain = (x['gnd']).reshape(-1)
xtrain, ytrain = shuffle(xtrain, ytrain, random_state=0)

def giveaccuracy(y1,y2):
        return np.sum(y1==y2)/len(y1)

maxacc = 0
bestk = 0
bestsigma = 0
sigmas = [x for x in range(5,100)]
Ks = [5,10,15,20]
for sigma in sigmas:
        for k in Ks:
                A = getaffinitymatrix(xtrain,sigma)
                feats = spectralcluster(A,k)
                y_kmeans = give_k_means(feats,15)
                y_pred = y_kmeans
                for i in range(10):
                        cnt = Counter(ytrain[y_kmeans==i])
                        if cnt.most_common() != []:
                                y_pred[y_kmeans == i] = cnt.most_common(1)[0][0]
                acc = giveaccuracy(y_pred, ytrain)
                if acc > maxacc:
                        maxacc = acc
                        bestk = k
                        bestsigma = sigma

print('Accuracy %4f, Best K %4f Best Sigma %4f'% (maxacc, bestk,bestsigma))
```

[3]The dataset is available at http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html

OBSERVATIONS

I tried various different settings for the scaling parameter, σ^2, and the number of largest eigen-vectors obtained from the spectral clustering algorithm, i.e., k. The best reported accuracy on this dataset was 40 percent, using this algorithm, for $k = 15$, and, $\sigma = 24$. I note that, after clustering the pictures into 15 groups, I assign labels to the clustered groups using the *mode* of original labels of images in that cluster. I am using original labels, since I merely want to check the effectiveness of the clustering algorithm. Though, for practical purposes when the original labels are not known, we can compare the cluster (using some distance metric such as the euclidean distance), from the labelled images, to assign label to the clusters. Since the dataset contains 15 clusters for different people, a random guess would provide an accuracy of 6.67%. Hence, 40 percent clustering accuracy is a reasonable performance for the algorithm, since the dataset is not perfectly tailored for clustering.

I also compared the clustering accuracy obtained using *Spectral Clustering* algorithm, with the case when we directly feed the original data points into the $K - means$ algorithm. Figure 2, shows the accuracy obtained by using $K - means$ for different number of columns (features) of the original data points. We can clearly see, spectral clustering provides a better accuracy, by projecting this data into a space, where the data points are more closely and convexly clustered.

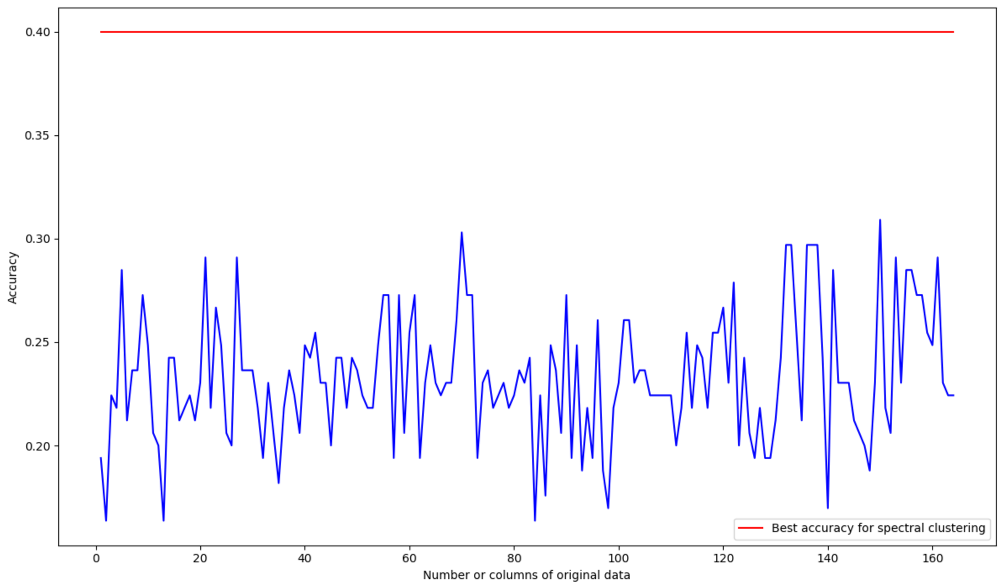

Figure 2. Clustering accuracy by using $K - means$ on original data

Code for obtaining the above curve.

```
accs = []
for l in range(1,165):
        y_kmeans = give_k_means(xtrain[:,:l],15)
        y_pred = y_kmeans
        for i in range(10):
                cnt = Counter(ytrain[y_kmeans==i])
                if cnt.most_common() != []:
                        y_pred[y_kmeans == i] = cnt.most_common(1)[0][0]
        accs.append(giveaccuracy(y_pred, ytrain))
```

```
plt.plot(range(1,165),accs,'b-')
plt.plot(range(1,165),[0.40 for i in range(1,165)],'r-',label = 'Best accuracy..')
plt.xlabel('Number or columns of original data')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.show()
```

REFERENCES

1. A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, pp. 849–856, 2002.