算法思想

PCA

```
def PCA(data, threshold):
  # 处理数据，使其中心化
  data -= np.mean(data, axis=0)
  # 计算协方差矩阵
  Cov = np.cov(data.T)
  # 作特征分解，得到特征向量和对应的特征值
  eigen_value, eigen_vector = np.linalg.eig(Cov)
  # 把得到的特征值与特征向量按照特征值从大到小的顺序排序，方便后续的选取主成分
  index = np.argsort(eigen_value)[::-1]
  eigen_value = eigen_value[index]
  eigen_vector = eigen_vector[:,index]
  # 根据Threshold选取满足条件的一定数量的特征向量
  total = np.sum(eigen_value)
  index = 1
  components = np.array([eigen_vector[:, 0]]).T
  selected = eigen_value[0] / total
  while(selected < threshold):
    print('test')
    components = eigen_vector[:, :index]
    selected += eigen_value[index] / total
    index += 1
  # 把中心化后的数据投影到特征平面
  print('Get', components.shape[1], 'principal components')
  projected = data @ components
  return projected
```

K-Means

```
def KMeans(k, data):
  # 随机选取k个数据点作为初始化均值
  means = data[np.random.choice(data.shape[0], size=k, replace=False), :]
  # 开始迭代
  means_old = -1
  while True:
    # 初始化
    Clusters = [np.array([]) for _ in range(k)]
    Prediction = [-1 for _ in range(data.shape[0])]
    # 找到与数据集中各个点最近的均值点，并把这些点分到对应的类中
    for pos, dot in enumerate(data):
      min_distance = float('inf')
      # 找最近均值点
      for index, mean_dot in enumerate(means):
        distance = np.sqrt(np.sum((dot - mean_dot) ** 2))
        if distance <= min_distance:
          min_distance = distance
```

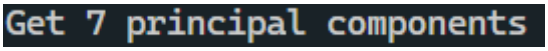
```
            to_be = index
        # 把数据点添加到和最近均值点对应的类中
        if Clusters[to_be].size == 0:
            Clusters[to_be] = np.array([dot])
        else:
            Clusters[to_be] = np.append(Clusters[to_be], [dot], axis=0)
        Prediction[pos] = to_be
    # 计算新的均值点
    means_old = means
    means = np.array([[]])
    for cluster in Clusters:
        temp = cluster.mean(axis=0)
        if means.size == 0:
            means = np.array([temp])
        else:
            means = np.append(means, [temp], axis=0)
    # 在迭代无大改变时终止迭代
    change = np.sum(abs(means - means_old))
    if (change < 0.001):
        # 计算轮廓系数
        S = []
        for index, i in enumerate(data):
            label = Prediction[index]
            cluster = Clusters[label]
            a_i = (np.sqrt(np.sum((i - cluster) ** 2, axis=1))).mean()
            min_distance = 100000000
            for index_j, j in enumerate(means):
                distance = np.sqrt(np.sum((i - j) ** 2))
                if distance < min_distance and index_j != label:
                    closest = index_j
            b_i = (np.sqrt(np.sum((i - Clusters[closest]) ** 2, axis=1))).mean()
            # get Silhouette coefficient
            sil_coef = (b_i - a_i) / max(a_i, b_i)
            S.append(sil_coef)
        # 返回对应结果，Prediction用于将数据保存至.csv文件时的类别属性指定
        return (Clusters, np.array(S).mean(), Prediction)
```

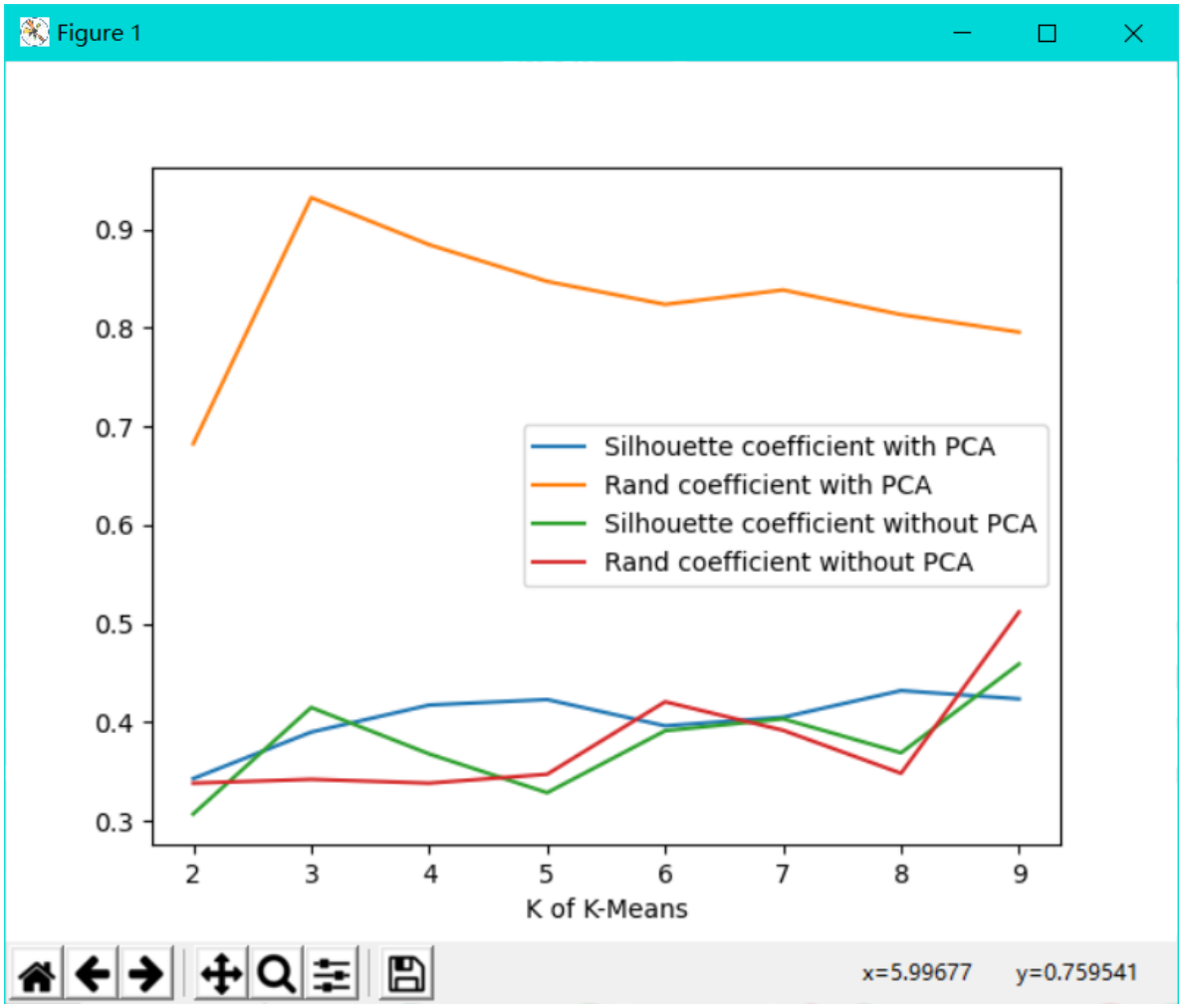
实验结果分析

不同Threshold降维结果分析

Threshold = 0.9

由下图知使用到了7个主成分，并且得到的最佳K值为3，与数据集一致



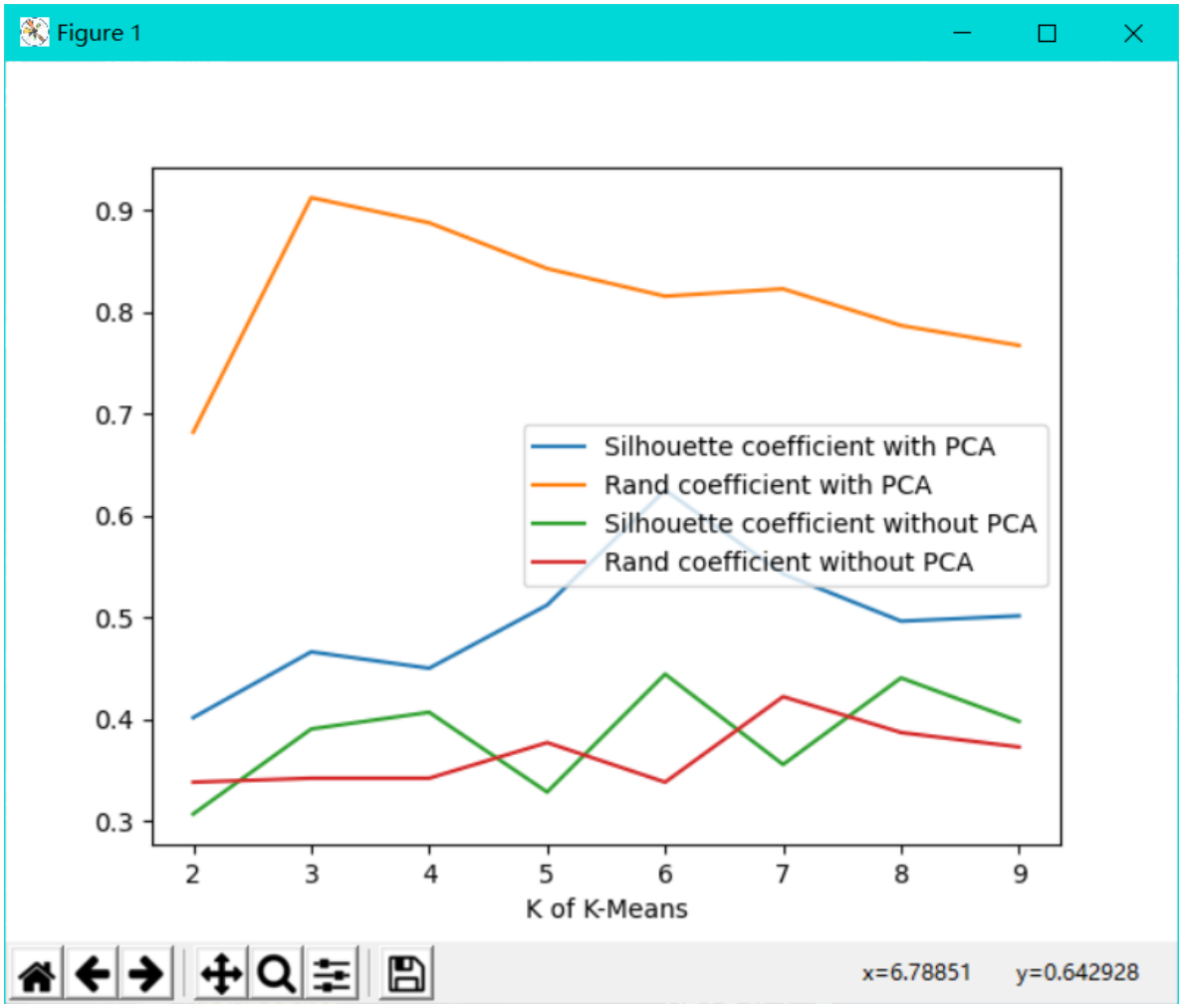



```
From the test above, the best choice is using 3 kinds. The corresponding Rand coefficient is  0.9273789119532787
And it uses PCA
```

Threshold = 0.8

由下图知使用到了4个主成分，并且得到的最佳K值为3，与数据集一致

```
Get 4 principal components
```

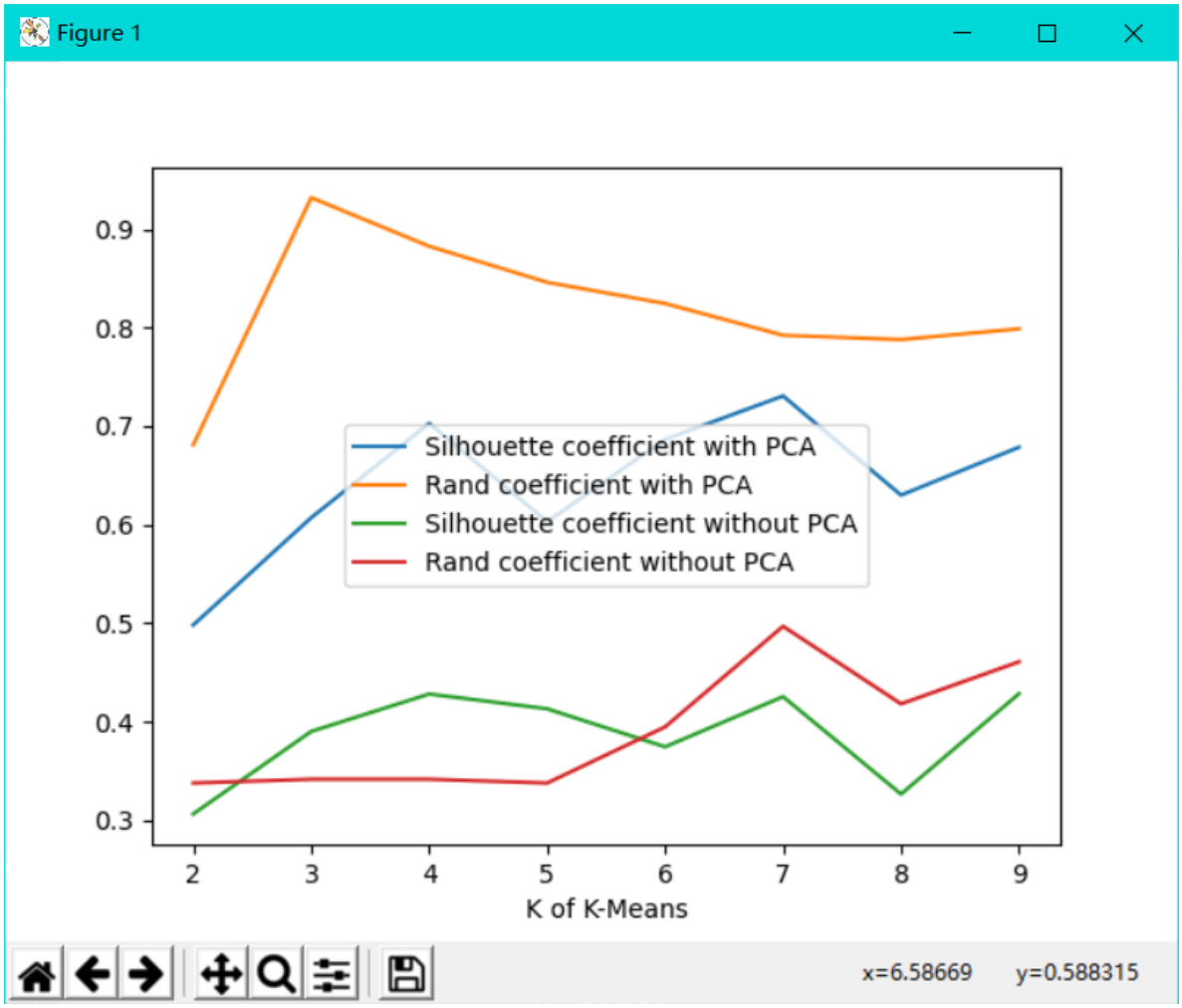



```
From the test above, the best choice is using 3 kinds. The corresponding Rand coefficient is  0.9250936329588015
And it uses PCA
```

Threshold = 0.6

由下图知使用到了2个主成分，并且得到的最佳K值为3，与数据集一致

```
Get 2 principal components
```

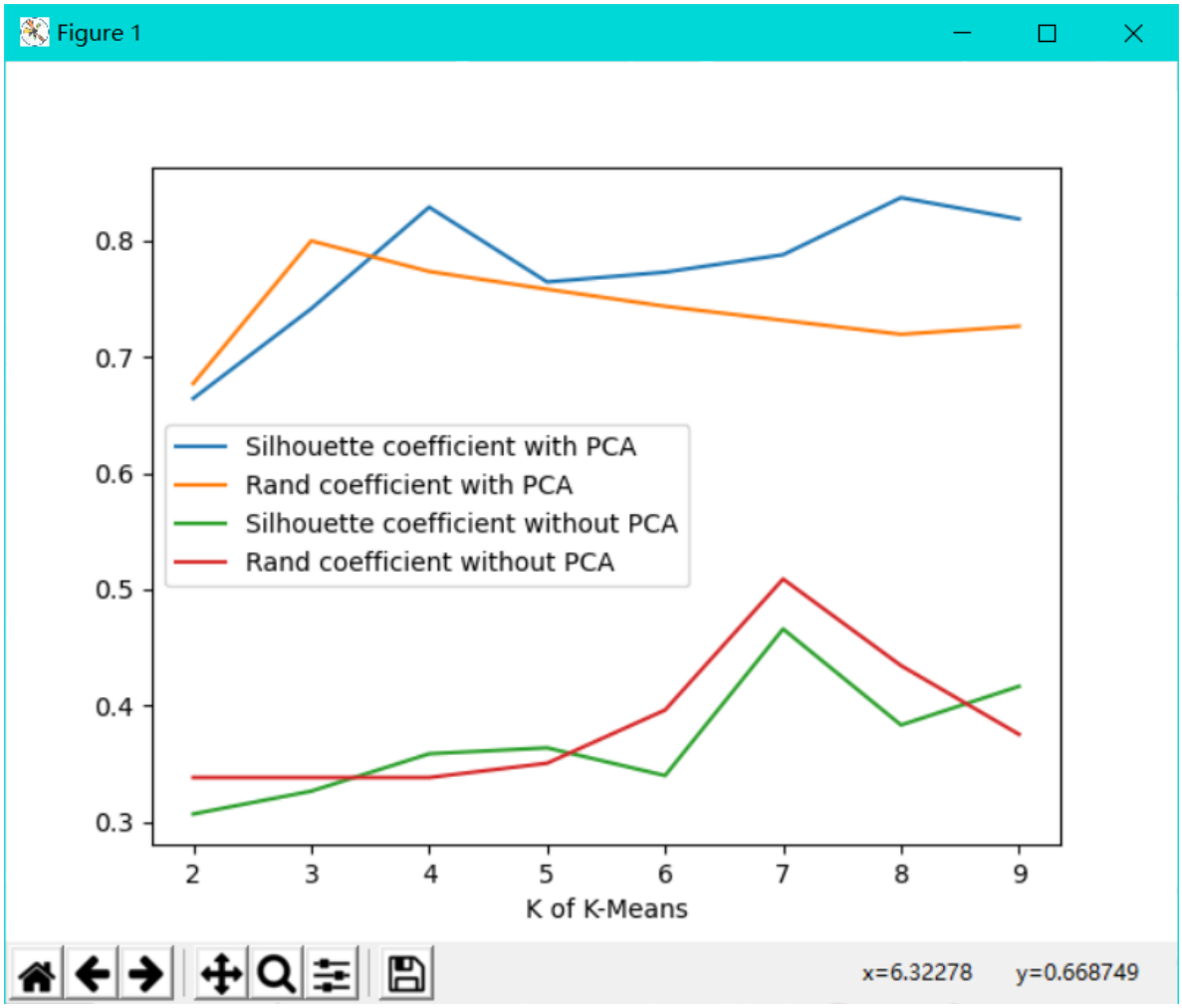



```
From the test above, the best choice is using 3 kinds. The corresponding Rand coefficient is  0.9318225099980956
And it uses PCA
```

Threshold = 0.3

由下图知使用到了2个主成分，并且得到的最佳K值为3，与数据集一致

```
Get 1 principal components
```



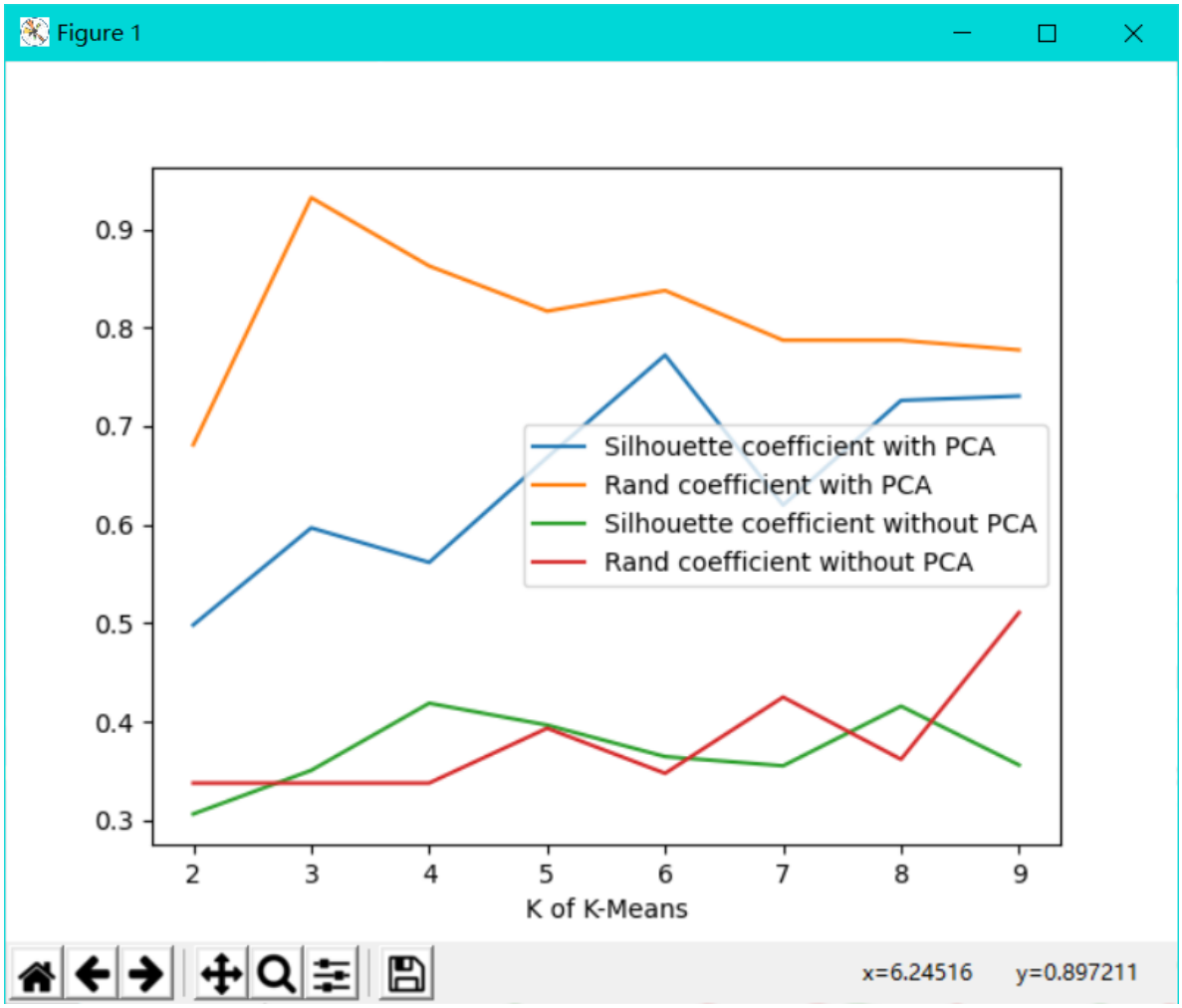

```
From the test above, the best choice is using 3 kinds. The corresponding Rand coefficient is  0.7998476480670349
And it uses PCA
```

总结

- 从上述各个Threshold的测试结果可以看出，随着Threshold的降低，轮廓系数不断提高
- 使用PCA相比不使用PCA得到得聚类结果更加精确，因此可见PCA对于聚类有积极作用
- 在取2个主成分（Threshold=0.6）时得到的结果最好，因此可见主成分有时会把一些干扰因素引入，而这些干扰因素不使用PCA的聚类效果差的原因

不同数量类别的K-Means聚类结果

采取PCA的Threshold=0.6下得到如下结果



- 从图中看出使用PCA对于K-Means聚类的结果有很大的提升
- 从图中可以看出K=3是最佳的输入K-Means方法的参数