Lab2_report

姓名：徐宇鸣

学号：PB17111636

P2:无监督学习问题

数据预处理：

在这里为数据做了标准化

(这里做了一大堆套娃的原因是我是找了一个现成的标准化函数，没有自己去对返回的data进行处理，因此做了一些套娃)

```
def standardization(x):
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0)
    return (x - mu) / sigma

def standaridzation_wrap(traindata):
  traindata_std = []
  for vector in traindata:
    traindata_std.append(list(standardization(np.array(vector,'float64'))))
  return np.array(traindata_std)
```

PCA算法：

算法分析

PCA算法我们要做的只有:

1. 将矩阵设置为行代表属性/变量，列代表样本：对输入进行专置
2. 为该矩阵计算协方差矩阵
3. 计算特征值和特征向量
4. 将特征向量根据特征值进行从大到小排序
5. 根据threshold决定取哪几个特征向量来组成矩阵
6. 进行矩阵乘法获得结果

算法代码

```
def PCA(data,threshold=0.9):
  cov = np.cov(data)
  eigVal,eigVec = np.linalg.eig(np.mat(cov))
  eigVal_sum = np.sum(eigVal)
  eigVal_rank = np.argsort(eigVal)[::-1]
  m = 0
```

```
  lower_sum = 0
  while 1 :
    upper_sum = lower_sum + eigVal[eigVal_rank[m]]
    lower = lower_sum/eigVal_sum
    upper = upper_sum/eigVal_sum
    if threshold > lower and threshold <= upper:
      break
    #print("lower:",lower)
    #print("upper:",upper)
    m+=1
    lower_sum = upper_sum
  #print(m)
  eigVec_firstm = eigVec.T[eigVal_rank[0:m+1]]
  #print(eigVec_firstm)
  #print(eigVec_firstm)
  data_pca = eigVec_firstm * data
  return data_pca
```

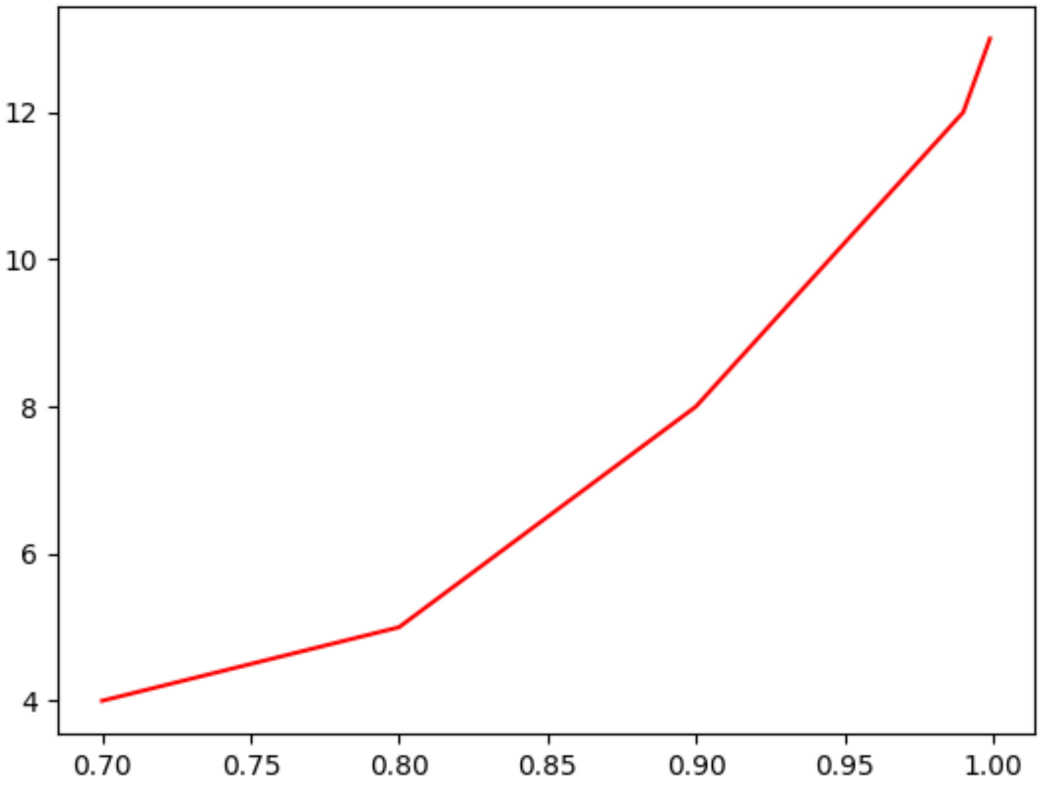
算法结果

选取的阈值为:

```
Thresholds = [0.7,0.8,0.9,0.99,0.999]
```

得到的结果

threshold	0.7	0.8	0.9	0.99	0.999
dim	4	5	8	12	13



可以看到，当threshold从0.8到0.9，再从0.9到0.99，每一次都能筛选到3，4个维度，这说明这些属性的影响占比相对较低

Kmeans算法：

算法分析

Kmeans算法的流程：

1. 随机选取k个点作为样本中心点
2. 对于每个样本点，计算与这k个点的距离，选取距离最小的点作为他的所属簇
3. 对获得的k个簇重新计算中心，如果最大的新中心与原中心的误差不超过0.0001，就认为收敛了，否则就进入新一轮的迭代

算法代码

```
def Kmeans(k,data):
  data_mat = np.matrix(data)
  tot = np.shape(data)[0]
  col = np.shape(data)[1]
  currentP_index = random_points(k,len(data))
  currentP = data_mat[currentP_index]
  max_dis = 1
  #print(currentP)
  while max_dis >0.0001:
    #if dis <= 0.0001, regard them as one point
```

```
    #sort
    group = {}
    grouplabel = []
    for i in range(k):
      group[i] = []
    for example in data_mat:
      dis_list = []
      for center in currentP:
        dis = cal_dis(example,center)
        dis_list.append(dis)
      dis_rank = np.argsort(dis_list)
      group[dis_rank[0]].append(example)
      grouplabel.append(dis_rank[0])
    lastP = currentP
    currentP = []
    #calculate the difference
    for key in group:
      total = np.matrix(np.zeros(col))
      for example in group[key]:
        total += example
      if group[key] != []:
        total /= len(group[key])
      currentP.append(total)
    max_dis = 0
    for lastC,curC in zip(group.keys(),currentP):
      dis = cal_dis(lastP[lastC],curC)
      if dis>max_dis :
        max_dis = dis
  S = 0
  #calculate S
  s = []
  for i in range(tot):
    a_s = []
    b_s = {}
    for m in range(k):
      if m == grouplabel[i]:
        continue
      b_s[m] = []
    for j in range(tot):
      if i == j:
        continue
      if grouplabel[i] == grouplabel[j]:
        a_s.append(cal_dis(data_mat[i],data_mat[j]))
      elif grouplabel[i] != grouplabel[j]:
        b_s[grouplabel[j]].append(cal_dis(data_mat[i],data_mat[j]))
    a_i = np.mean(np.array(a_s))
    for m in range(k):
      if m == grouplabel[i]:
        continue
```

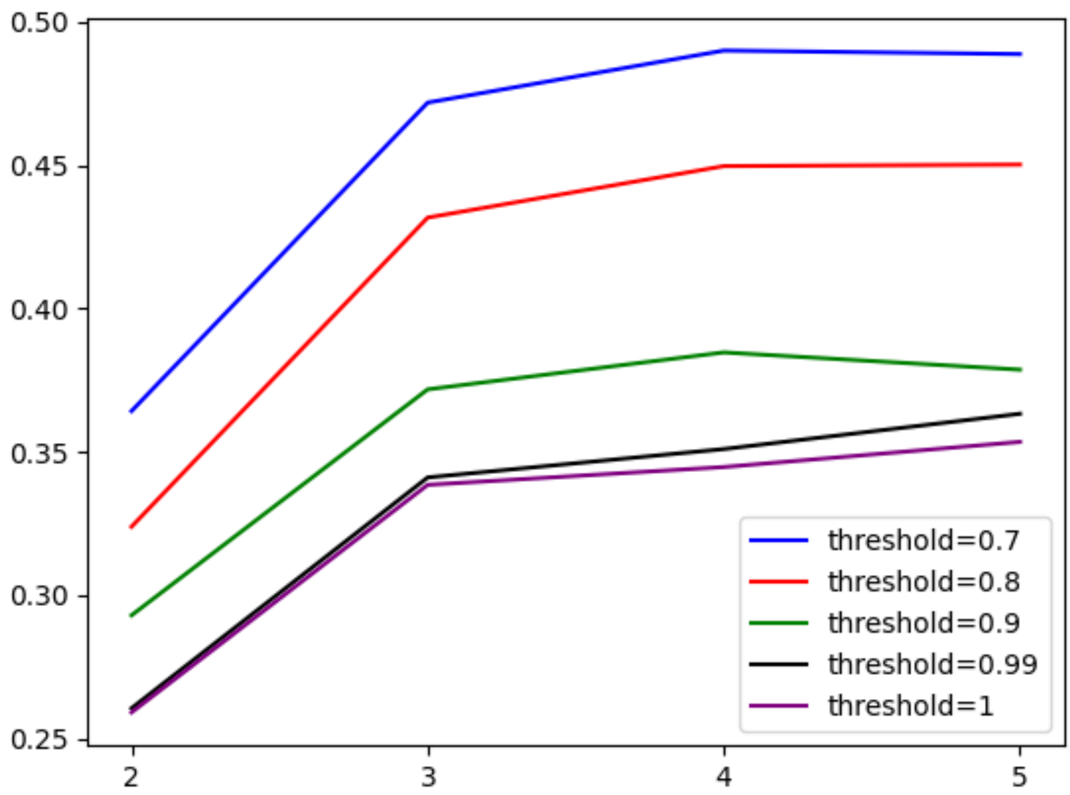
```
            b_s[m] = np.mean(np.array(b_s[m]))
        b_s_rank = sorted(b_s.items(),key=lambda b_s:b_s[1],reverse=False)
        #print(b_s_rank)
        b_i = b_s_rank[0][1]
        #print(a_i)
        #print(b_i)
        s.append((b_i - a_i) / max(a_i, b_i))
    S = np.mean(np.array(s))
    return (grouplabel,S)
```

算法结果

首先是聚类k=2，3，4，5之后的轮廓系数和兰德系数对比

轮廓系数：

K	threshold=0.7（4维）	threshold=0.8（5维）	threshold=0.9（8维）	thrshold=0.99（12维）	threshold=1（13维）
2	0.364	0.323	0.293	0.260	0.259
3	0.471	0.431	0.371	0.341	0.338
4	0.489	0.449	0.384	0.350	0.344
5	0.488	0.450	0.378	0.363	0.353



可以看到

随着维度的下降，轮廓系数是逐渐在升高的，而当k增大的时候，轮廓系数也是在增大的

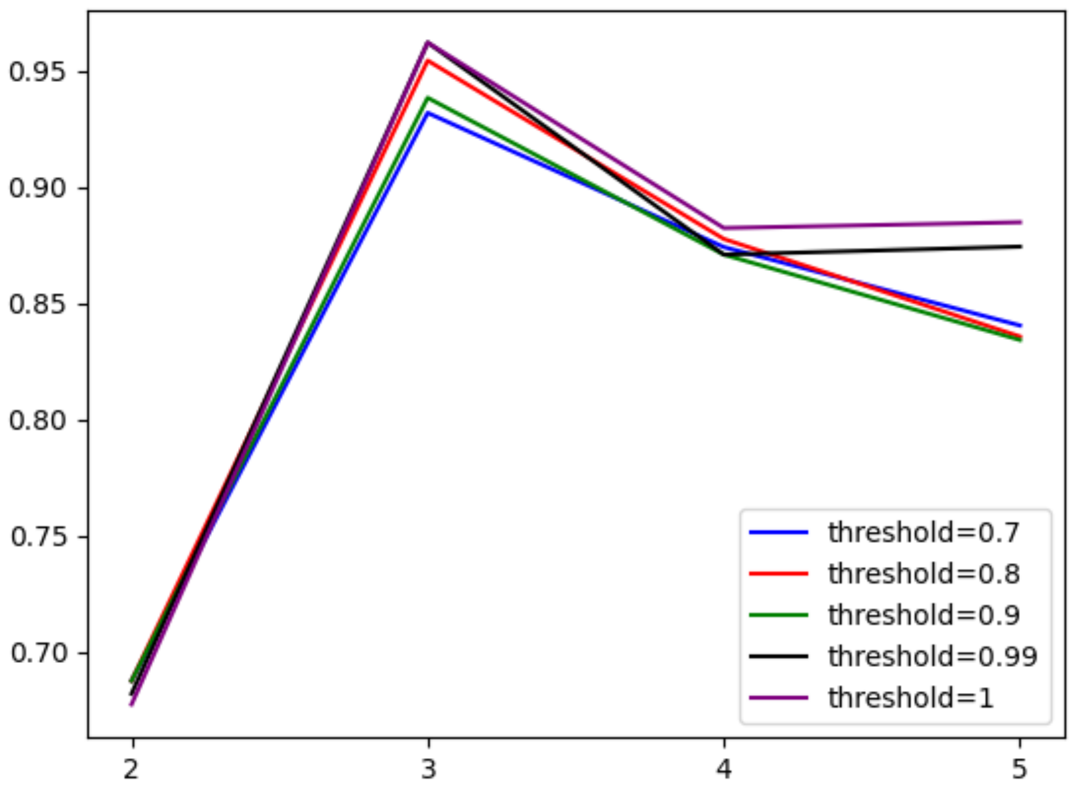
降维的效果是使得主成分的作用更加明显，因此聚类效果也越好

k越大说明簇越多，我们分的也就可以越细，因此聚类效果也能更好

兰德系数：

[[0.6876785374214436, 0.9318225099980956, 0.8741192153875452, 0.8404113502190059], [0.6876785374214436, 0.9542944201104552, 0.8776106138513299, 0.835650352313845], [0.6876785374214436, 0.9382974671491144, 0.8708817368120358, 0.8341903129562623], [0.682092299879388, 0.9620389767028502, 0.8708817368120358, 0.8743096553037517], [0.6775217418904336, 0.9620389767028502, 0.882308131784422, 0.8847838506951057]]

K	threshold=0.7（4维）	threshold=0.8（5维）	threshold=0.9（8维）	thrshold=0.99（12维）	threshold=1（13维）
2	0.687	0.587	0.687	0.682	0.677
3	0.931	0.954	0.938	0.962	0.962
4	0.874	0.877	0.870	0.870	0.882
5	0.840	0.835	0.834	0.874	0.884



可以看到

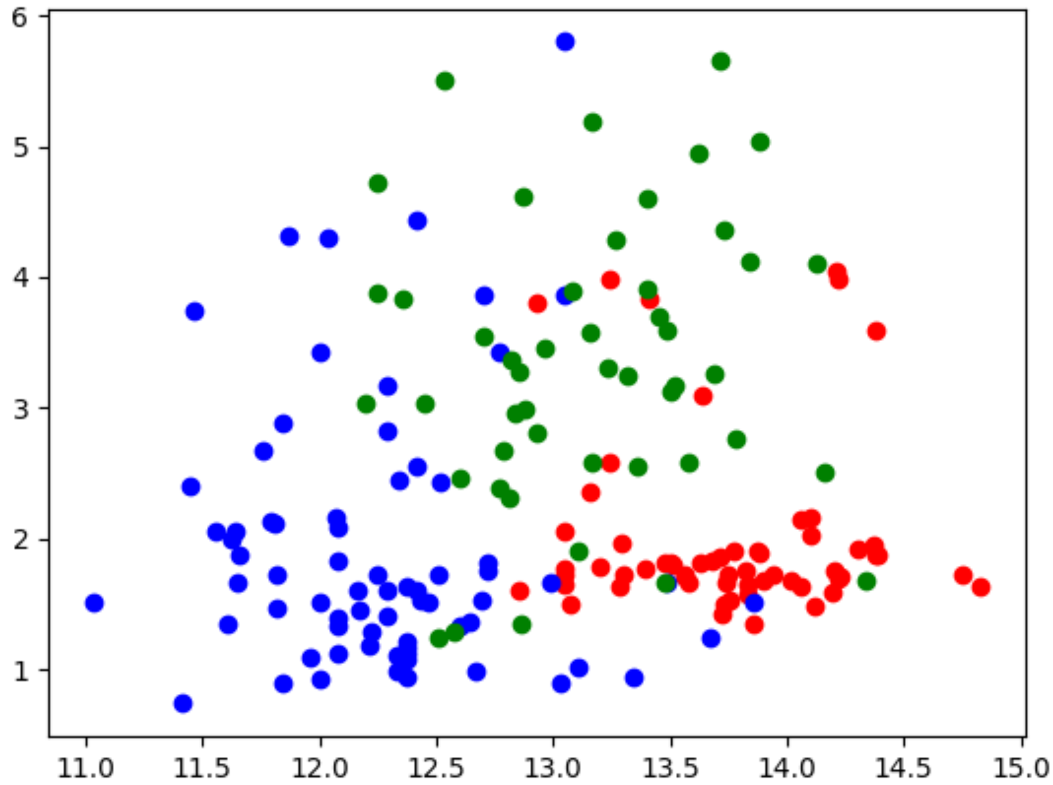
对于任意的threshold，兰德系数最高，也就是聚类最符合真实结果的点都在k=3的位置，这也符合我们的真实结果，除此之外，当threshlod增大时，同一k聚类的兰德系数也呈上升趋势，这是因为维度变大后计算距离的维度也增加，也就能分的更细的原因，但是总体来看 threshold=0.8就能取得很不错的效果

算法结果总结

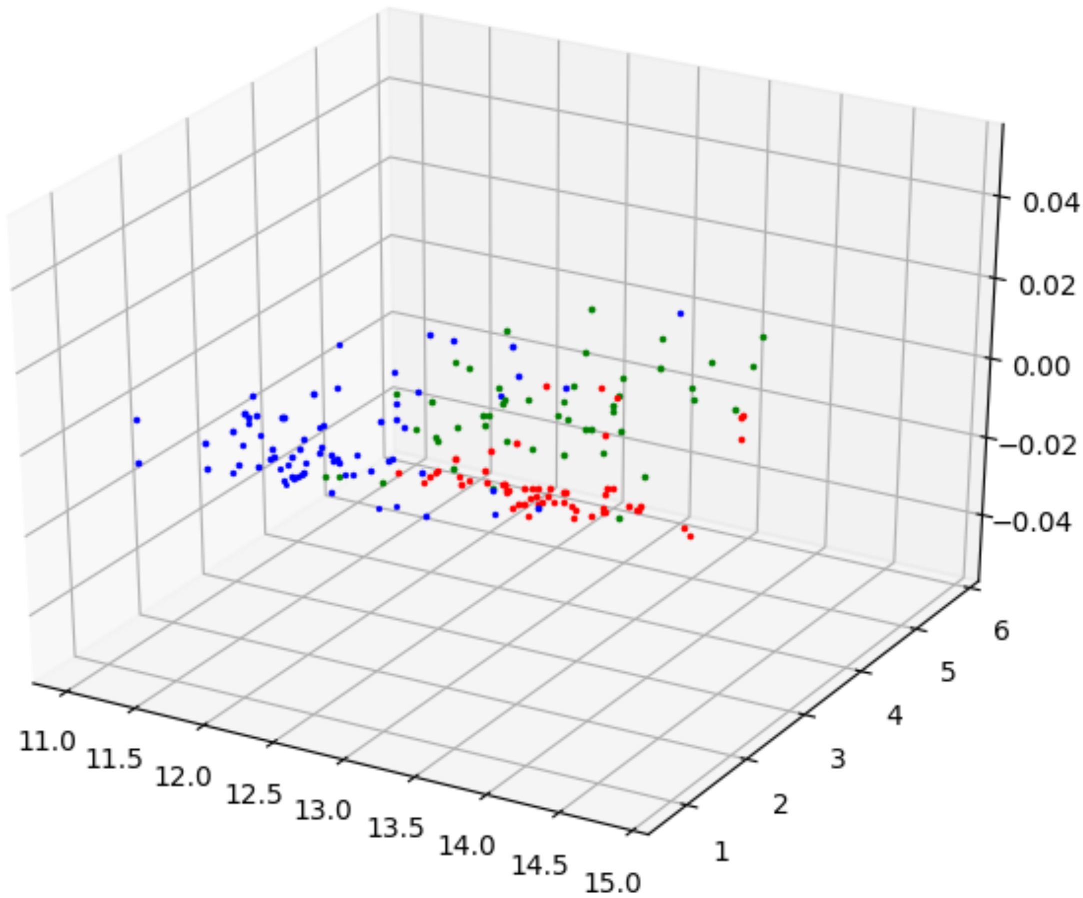
聚类结果可视化

在这里就仅放出真实结果的2d图以及3d图，还有threshold = 0.9，k=3的2d图和3d图，更多的图可以直接运行python程序然后在output文件夹查看，或者直接在output查看已经有的图

真实结果2d图，取了前2个维度（同时也是特征值最高的两个维度）

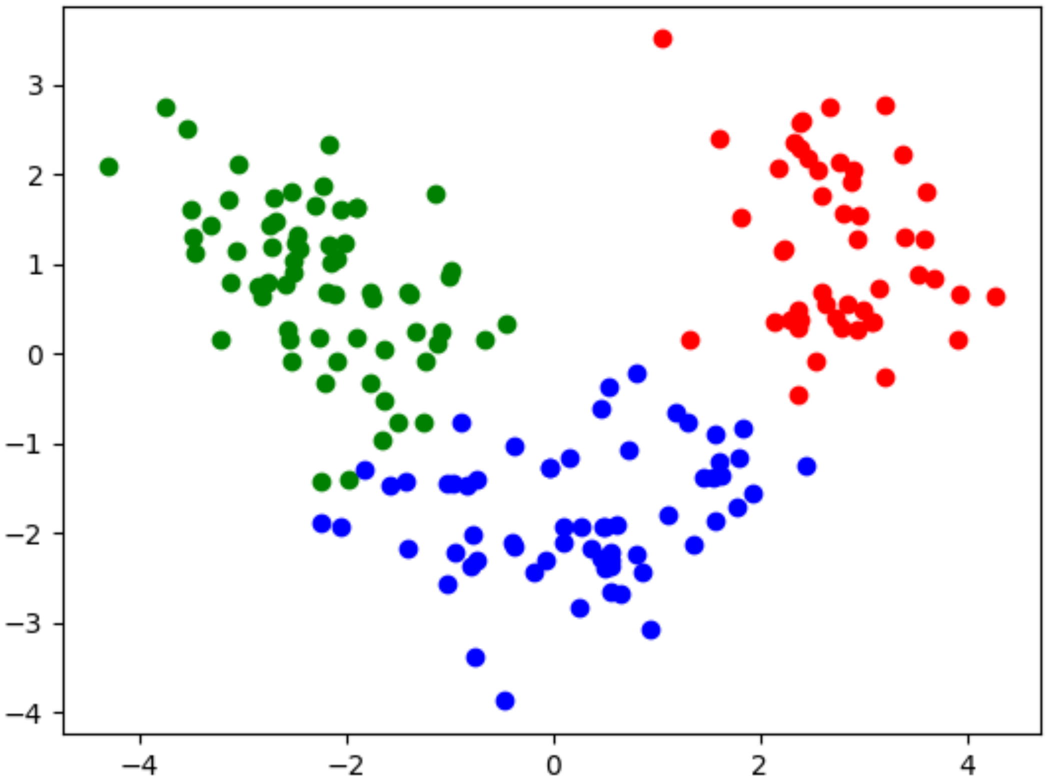


真实结果3d图，取了前3个维度（同时也是特征值最高的三个维度）

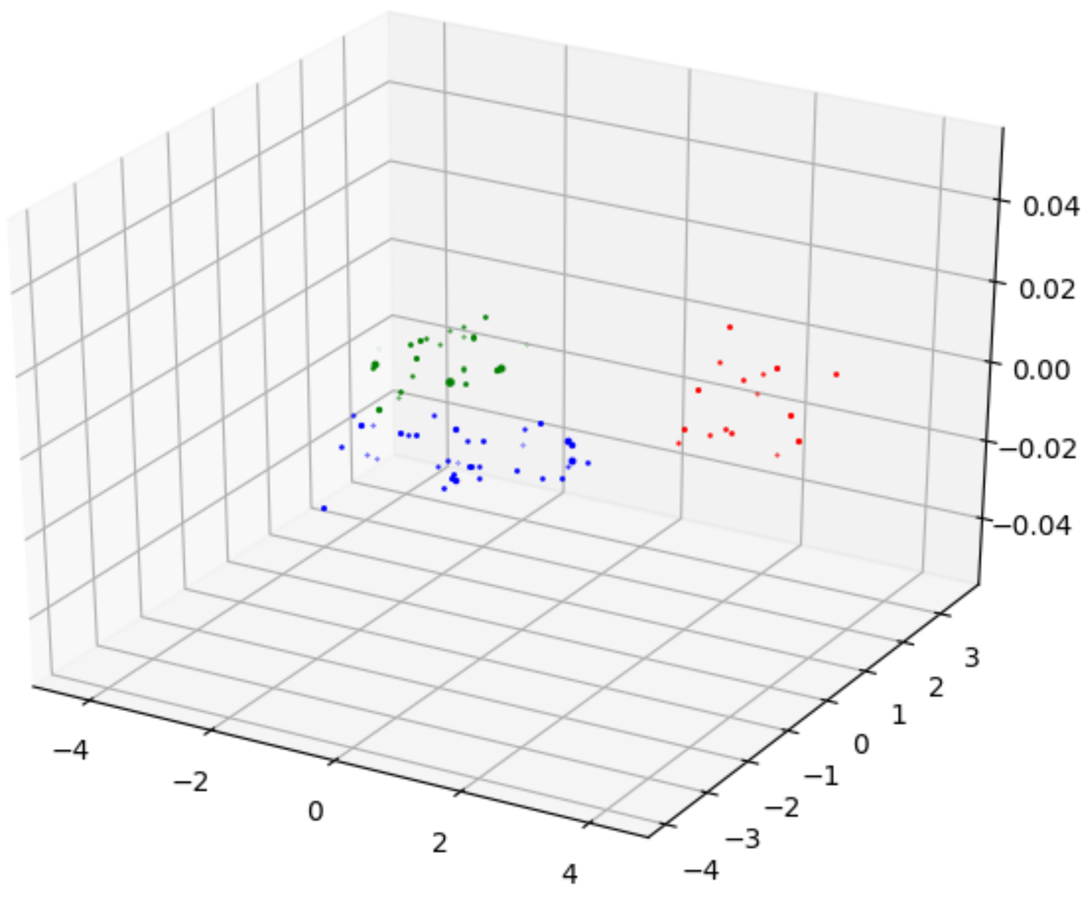


可以看到，在不进行降维的情况下，不管是2维还是3维的聚类效果都不是那么直观明显

k=3， 降维结果矩阵为8维/threshold=0.9，2d图：



3d图：



更进一步的话其实还可以标注出聚类效果的中心点（但是我懒了）

可以看到在进行了降维之后在二维、三维上，可视化后的点集聚类效果明显