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[eigVal_rank[0:m+1]]
  #print(eigVec_firstm)
  #print(eigVec_firstm)
  data_pca = eigVec_firstm * data
  return data_pca
```

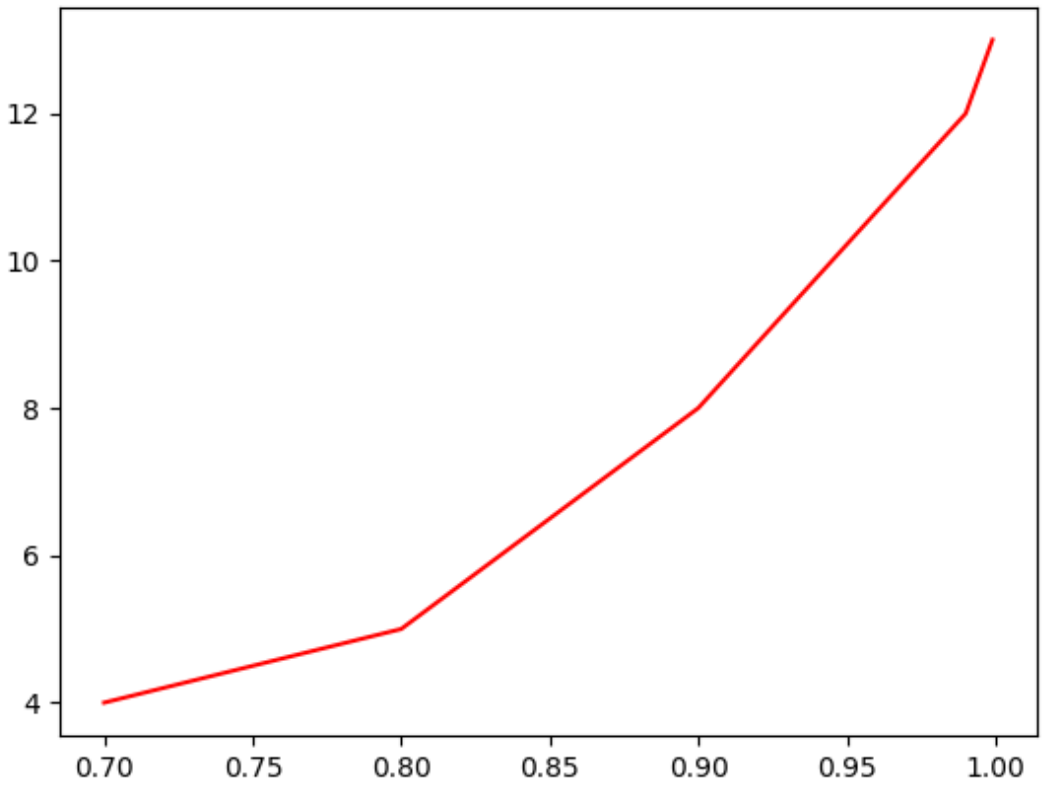
算法结果

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
  for C in group.keys():
    dis_C_min = 100000
    for otherC in group.keys():
      if C is otherC:
        continue
      dis_C = cal_dis(currentP[C],currentP[otherC])
      if dis_C_min > dis_C :
        dis_C_min = dis_C
        nearC = otherC
    for Vec in group[C]:
      a = 0
      for otherVec in group[C]:
        if otherVec is Vec:
          continue
        a += cal_dis(Vec,otherVec)
      a /= len(group[C]) - 1
      b = 0
      for otherVec in group[nearC]:
        b += cal_dis(Vec,otherVec)
```

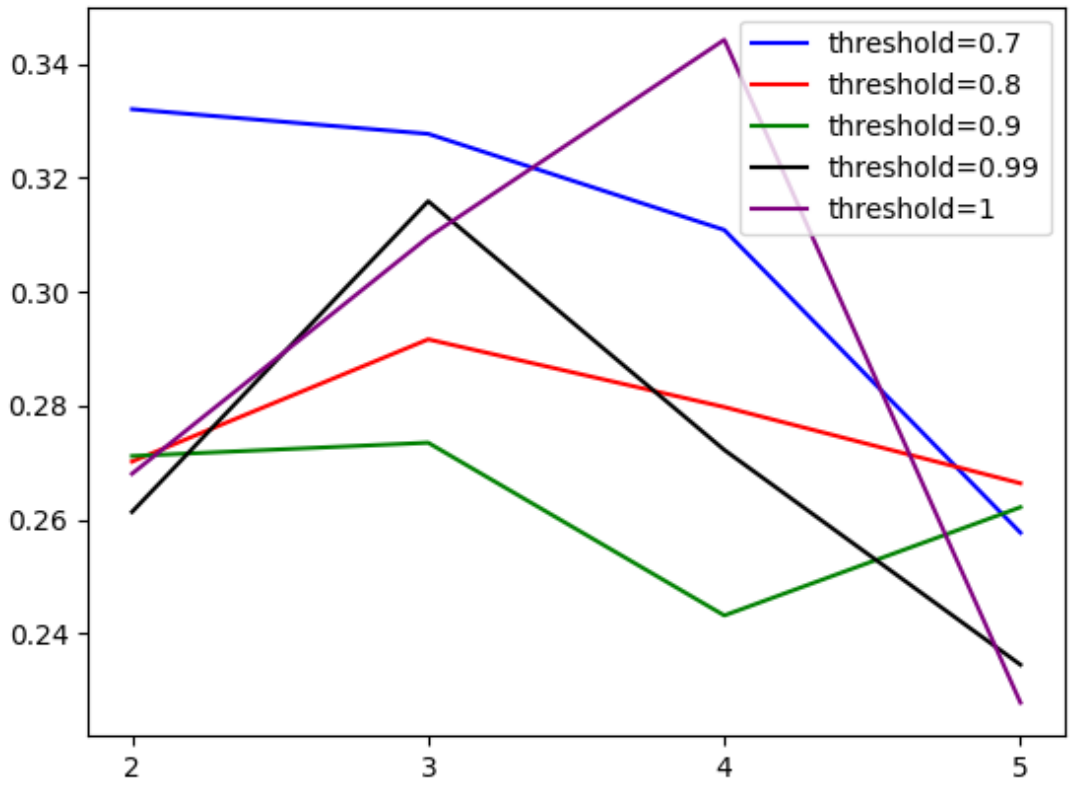
```
            b /= len(group[nearC]) - 1
            S += (b-a)/(max(a,b))
        S /= tot
        return (grouplabel,S)
```

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	0.332010668007553	0.2701819758157376	0.27114957199138906	0.26134135189474395	0.2680411489143324
3	0.32768527409527737	0.2916266882314143	0.2734895101044082	0.31586331944148943	0.30955142992972723
4	0.31082456668431163	0.279747765356576	0.24317829812240205	0.272221243296691	0.34419009337878215
5	0.2577042684644359	0.2663635392614887	0.2621713869985771	0.23453211921894798	0.22787357468477382



可以看到

对于threshold=0.7，也就是降维到4维的情况下，k=2时聚类效果最好，此时分类更接近于2类

对于threshold=0.8，也就是降维到5维到情况下，k=3时聚类效果最好，此时分类更接近于3类

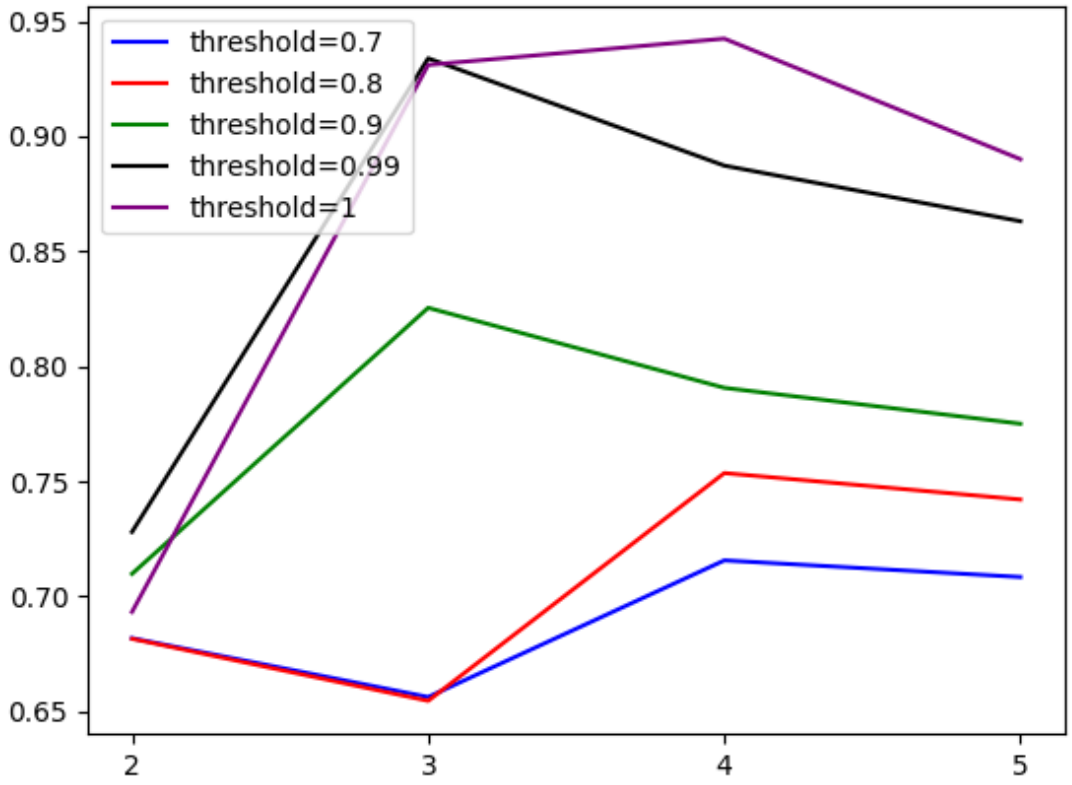
对于threshold=0.9、0.99时，也就是8维以及12维的情况，也是k=3时聚类效果最好，也是分为3类

而对于threshold=1，不进行降维的情况，反而是k=4的时候聚类效果最好，此时分类更接近于4类

但是从12维到13维其实增加的属性在特征值占比上并不高，原因个人认为是：注意到因为样本点的选取具有随机性，因此结果只能做一个参考，我们的聚类结果会收到初始随机选择样本中心点的影响

兰德系数：

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	0.681774900019044	0.6815209801307688	0.7097695677013902	0.7279883196851393	0.6932012949914302
3	0.6561924712753127	0.6545419920015235	0.8254300768107662	0.9339173490763664	0.9310607503332698
4	0.7156097251317209	0.7535707484288707	0.7906430521170571	0.8872595696057893	0.942487145305656
5	0.7083730083158764	0.7420808734844156	0.7750269789881292	0.8630737002475719	0.8900526883768172



可以看到

对于threshold=0.7以及0.8，也就是降维至4维以及5维的情况下，与真实聚合情况相比，结果好的是k=4的情况，但是总体也都是不佳（最高也只有0.75）

对于threshold=0.9以及threshold=0.99的情况下，也就是降维至8维以及12维的情绪下，与真实聚合情况相比，结果好的是k=3的情况，这相对比较符合我们的真实聚合情况

而对于threshold=1，也就是不进行降维的情况下，k=3以及k=4的效果都不错，但是k=4的效果比k=3的效果来的好，我想主要原因出在kmeans算法的样本点随机选取的问题

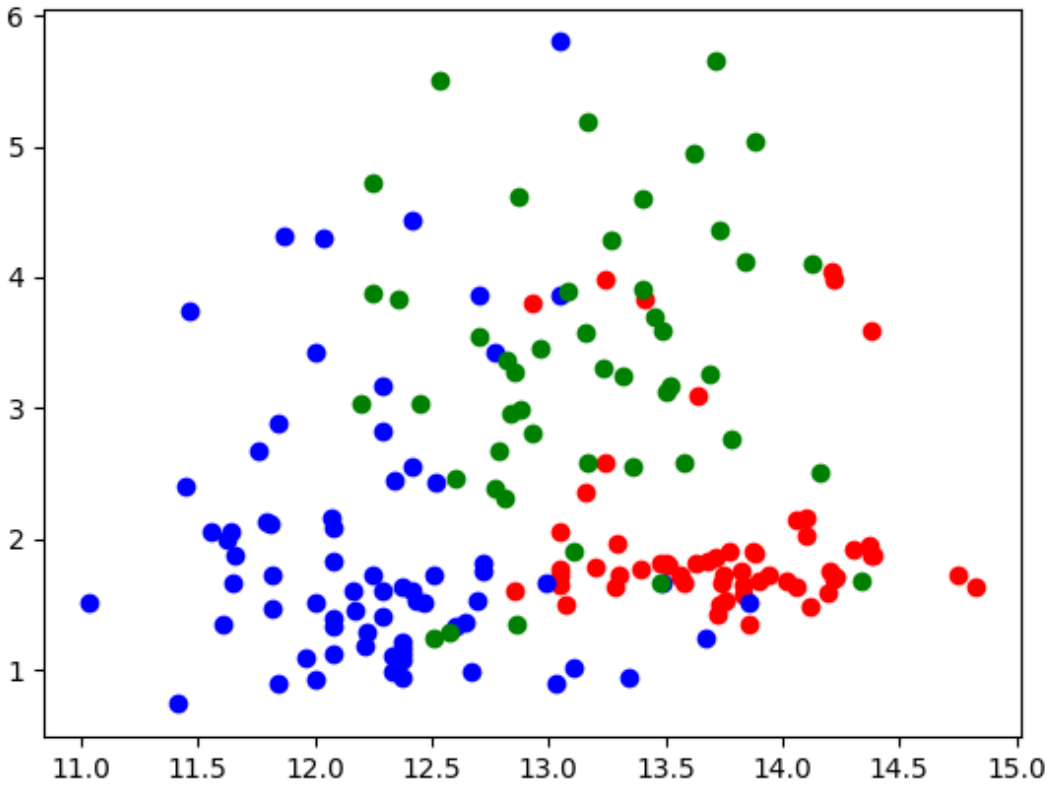
算法结果总结

可以看到，总体结果上，当出现降维的情况时如果降维太多会出现聚类结果不是k=3最好的情况，而且由于kmeans算法的初始点选择的随机性，会导致最后的结果也会稍稍偏离预期，这个误差我认为如果经过多次试验取平均值可以消除，不过这里没有做更多的实验

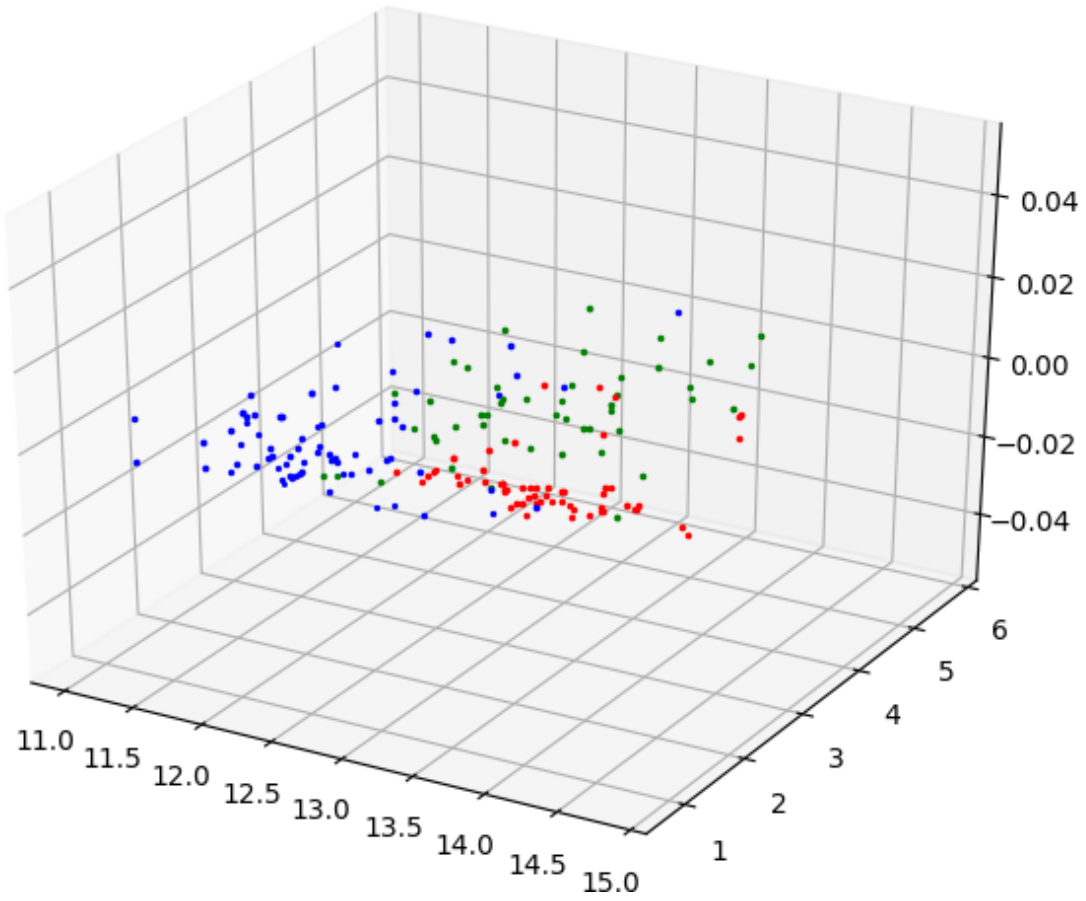
聚类结果可视化

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

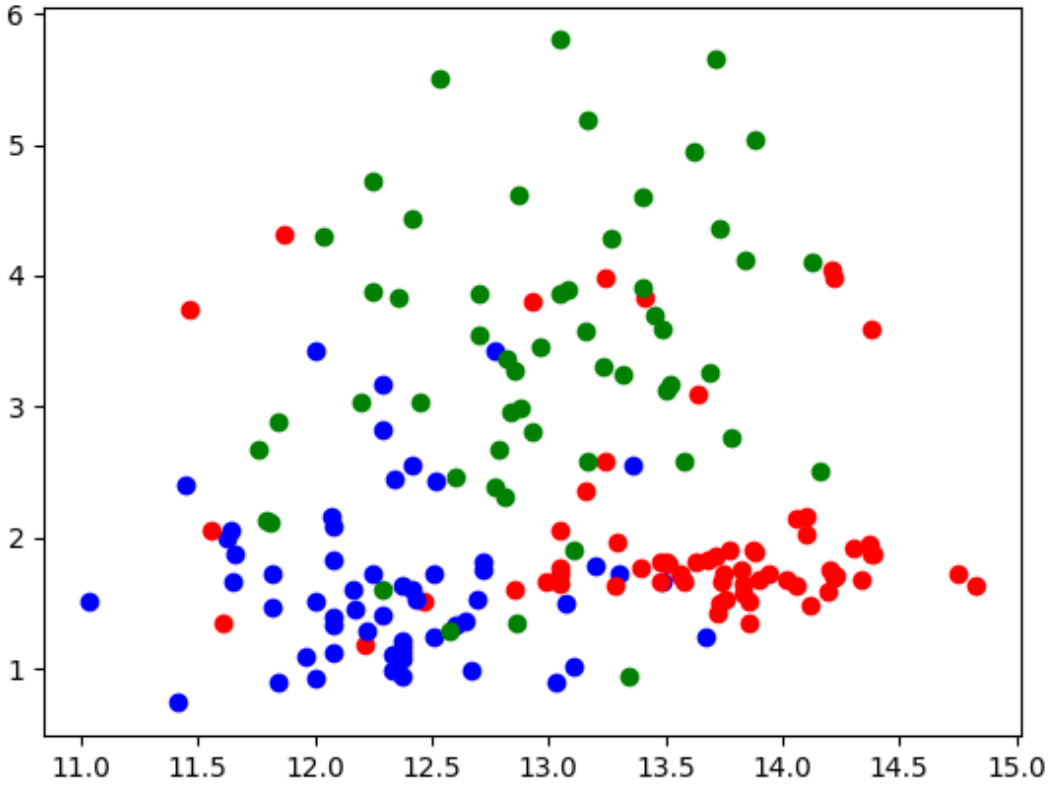
真实结果2d图，取了前2个维度（同时也是特征值最高的两个维度）



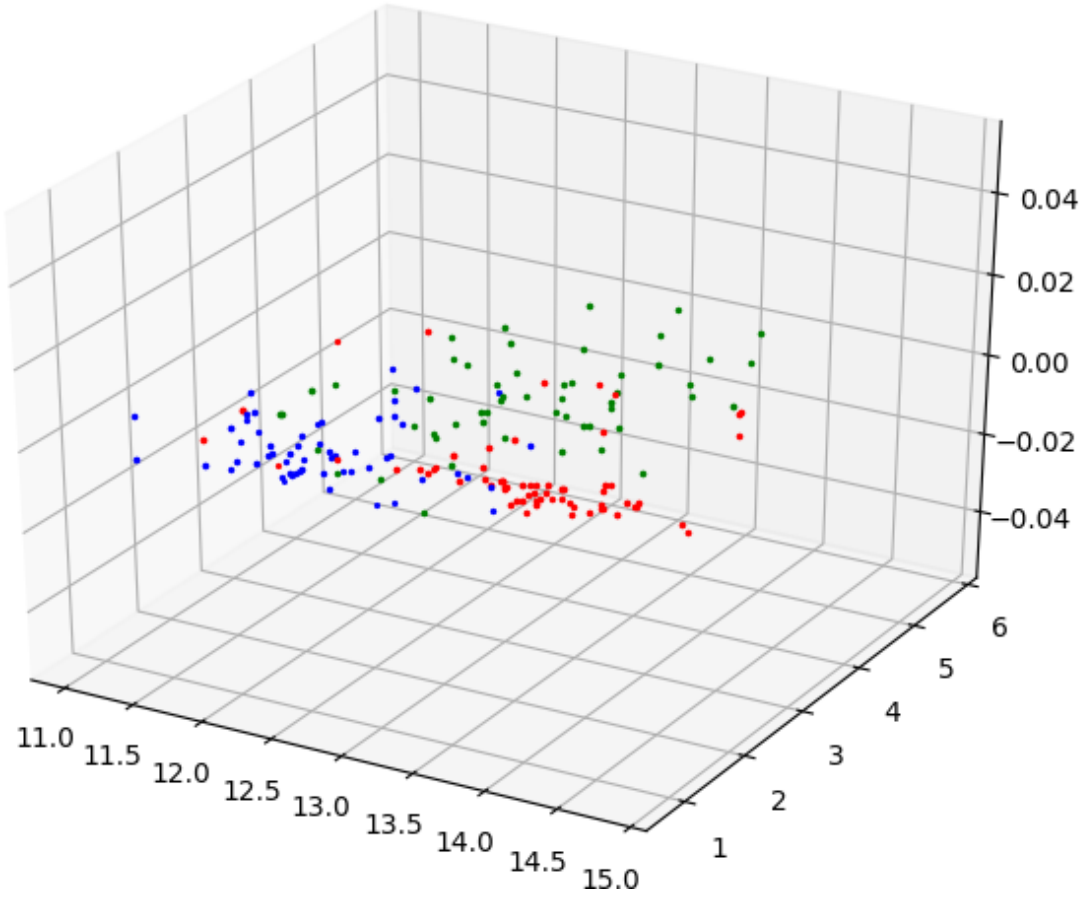
真实结果3d图，取了前3个维度（同时也是特征值最高的三个维度）



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）