Kmeans++算法

/\*将测试向量读入，之后将行和列转置之后用myVector来保存，clusters用来划分不同的聚类比如，如果要选取7个向量，那么就需要k=7个聚类中心\*/

Vector<vector<double>> myVector

Vector<vector<double>> clusters[k]

Vector<double> means

#define CHANGE 0.5 //定义一个准则值，当变化小于0.5时不需要再进行迭代计算聚类中心

means  **<-** getInitMeans(myVector) //初始化各个聚类中心

For i **<-** 0 to myVector.size //根据初始的聚类中心将每一个向量划分到每一个clusters;

Label **<-** clusterOfTuple(means,myVector[i]) //确定当前向量属于哪一个聚类中心对应的簇

clusters[lable].push\_back(myVector[i]) //将相应的向量放入对应的族中

Var oldVar = -1

Var newVar = getVar(clusters,means)

For i  **<-** (newVar-oldVar) to CHANGE

For j  **<-** 0 to k

means[j]  **<-**  getMeans(clusters[j]) //更新每个簇的中心点,就是根据每个族的向量取平均值

oldVar = newVar;

newVar = getVar(clusters,means) //计算新的准则值,就是聚簇中的每一个向量与聚类中心的欧几里距离

For j **<-**  0 to k //清空每个簇

clusters[j].clear()

For j **<-** 0 to myVector.size() //根据新的质心获得新的簇

Label **<-** clusterOfTuple(means,myVector[i])

clusters[lable].push\_back(myVector[i])

/\*当newVar-oldVar **<-** CHANGE 时每一个聚簇里面包含的向量大致不再变化，我们就跳出迭代，产生出需要的K个聚类中心\*/

//获取初始的聚类中心

**Void getInitMeans (myVector)**

temp = 0

Means[temp++] = myVector[0] //myVector第一个向量为第一个聚类中心

For i **<-** 1 to k

For j **<-** 0 to myVector.size() //遍历所有向量，选取与means中存在的向量差距最大的向量作为新的聚类中心

Means[temp++] **<-** GetMaxDistance(myVector[j],means)

//获取每个聚簇中的向量的聚类中心之间值之和

**Double getVar(clusters,means)**

Var= 0

For i 0 **<-** k

t = clusters[i]

For j **<-** 0 to t.size

Var+=getDistXY(means[i],t[j])//此距离为欧几里得距离

Return var