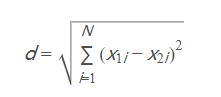
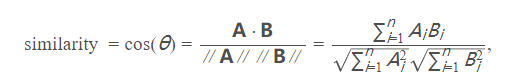
计算两点间距离

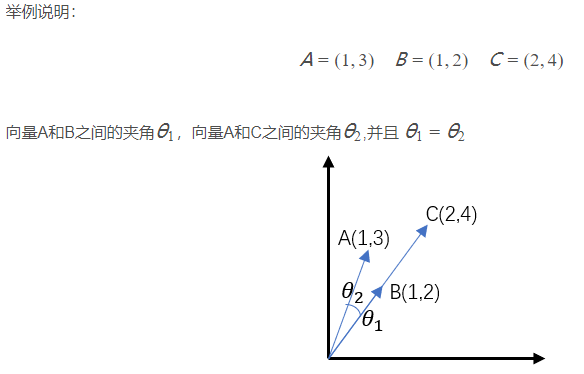
**一、欧氏距离：**



**D = np.sqrt(np.sum(np.square(A-B)))**

**二、余弦距离：**





**np.dot(A, B)/(np.linalg.norm(A) \* np.linalg.norm(B))**

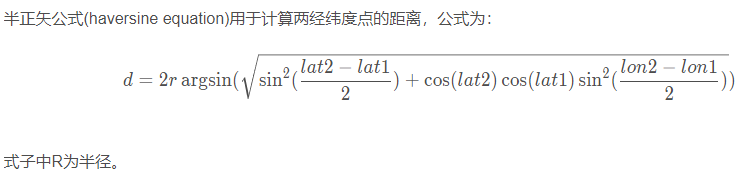
欧氏距离体现数值上的绝对差异，而余弦距离体现方向上的相对差异。

1）例如，统计两部剧的用户观看行为，用户A的观看向量为(0,1)，用户B为(1,0)；此时二者的余弦距很大，而欧氏距离很小；我们分析两个用户对于不同视频的偏好，更关注相对差异，显然应当使用余弦距离。

2）而当我们分析用户活跃度，以登陆次数(单位：次)和平均观看时长(单：分钟)作为特征时，余弦距离会认为(1,10)、(10,100)两个用户距离很近；但显然这两个用户活跃度是有着极大差异的，此时我们更关注数值绝对差异，应当使用欧氏距离。

**总结：在日常使用中需要注意区分，余弦距离虽然不是一个严格意义上的距离度量公式，但是形容两个特征向量之间的关系还是有很大用处的。比如人脸识别，推荐系统等。**

**三、haversine公式计算两经纬度点距离**



**def geo\_distance(origin, destination):**

**"""**

**Calculate the Haversine distance.**

**Parameters**

**----------**

**origin : tuple of float (lat, long)**

**destination : tuple of float (lat, long)**

**Returns**

**-------**

**distance\_in\_km : float**

**Examples**

**--------**

**>>> origin = (48.1372, 11.5756) # Munich**

**>>> destination = (52.5186, 13.4083) # Berlin**

**>>> round(distance(origin, destination), 1)**

**504.2**

**"""**

**lon1, lat1 = origin**

**lon2, lat2 = destination**

**radius = 6371 # km, Radius of the earth in km**

**dlat = math.radians(lat2 - lat1) #角度转换为弧度**

**dlon = math.radians(lon2 - lon1)**

**a = (math.sin(dlat / 2) \*\* 2 +**

**math.cos(math.radians(lat1)) \* math.cos(math.radians(lat2)) \***

**math.sin(dlon / 2) \*\* 2)**

**c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))**

**d = radius \* c**

**return d**

地球半径为6378km，R=6378\*1000m，则haversine公式可写为：

**def haversine(lon1, lat1, lon2, lat2):**

**lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])**

**dlon = lon2 - lon1**

**dlat = lat2 - lat1**

**a = np.sin(dlat/2.0)\*\*2 + np.cos(lat1) \* np.cos(lat2) \* np.sin(dlon/2.0)\*\*2**

**c = 2 \* np.arcsin(np.sqrt(a))**

**return c \* 6367 \* 1000**