常见的距离算法和相似度计算方法

机器学习研究组订阅 2020-08-01

作者｜奋发的菜鸟酱@知乎

来源｜https://zhuanlan.zhihu.com/p/138107999

本文整理了常见的距离算法和相似度（系数）算法，并比较了欧氏距离和余弦距离间的不同之处。

1、常见的距离算法

1.1 欧几里得距离（Euclidean Distance）

$$d = \sqrt{\sum_{n}^{i=1}(x_i - y_j)^2}$$

在数学中，欧几里得距离或欧几里得度量是欧几里得空间中两点间"普通"（即直线）距离。使用这个距离，欧氏空间成为度量空间。相关联的范数称为欧几里得范数。

Euclidean Distance是一个通常采用的距离定义，它是在m维空间中两个点之间的真实距离。

代码：

```
>>> pdist = nn.PairwiseDistance(p=2)
>>> input1 = torch.randn(100, 128)
>>> input2 = torch.randn(100, 128)
>>> output = pdist(input1, input2)
```

1.2 Earth Mover's Distance (EMD距离)

和欧式距离一样，它们都是一种距离度量的定义、可以用来测量某两个分布之间的距离。EMD主要应用在图像处理和语音信号处理领域。

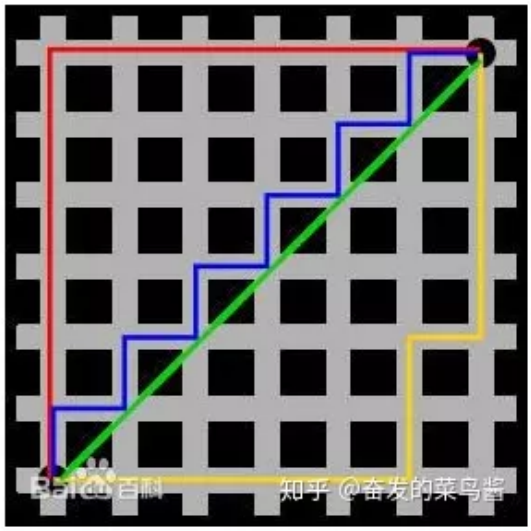
EMD问题通俗解释: Earth Move翻译过来是搬土，指把P位置的m个坑的土，用最小的代价搬到Q位置的n个坑中，dij是pi到qj两个坑的距离，fij是从pi搬到qj的土量，则WORK工作量就是要最小化的目标。线性规划求解出fij后，再用fij对WORK作个归一化，就得到了EMD。EMD 实际上是线性规划中运输问题的最优解。

EMD具体定义可参考：http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

```
C代码包：emd.h, emd.c, emd.i

OpenCV：实现了EMD api,
    pip install --upgrade setuptools
    pip install numpy Matplotlib
    pip install opencv-python

import numpy as np
import cv

#p、q是两个矩阵，第一列表示权值，后面三列表示直方图或数量
p=np.asarray([[0.4,100,40,22],
        [0.3,211,20,2],
        [0.2,32,190,150],
        [0.1,2,100,100]],np.float32)
q=np.array([[0.5,0,0,0],
        [0.3,50,100,80],
        [0.2,255,255,255]],np.float32)
pp=cv.fromarray(p)
qq=cv.fromarray(q)
emd=cv.CalcEMD2(pp,qq,cv.CV_DIST_L2)
```

1.3 曼哈顿距离（Manhattan Distance）：表示两个点在标准坐标系上的绝对轴距之和。也就是和象棋中的"車"一样横平竖直的走过的距离。曼哈顿距离是超凸度量。

$$d = \sum_{i=1}^{n} |x_i - y_i|$$



红色：曼哈顿距离，绿色：欧式距离，蓝色和黄色代表等价的曼哈顿距离

1.4 杰卡德距离（Jaccard Distance）：用来衡量两个集合差异性的一种指标，它是杰卡德相似系数的补集，被定义为1减去Jaccard相似系数。适用于集合相似性度量，字符串相似性度量 。

$$d(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

1.5 马氏距离（Mahalanobis distance）：表示点与一个分布之间的距离。它是一种有效的计算两个未知样本集的相似度的方法。与欧氏距离不同的是，它考虑到各种特性之间的联系（例如：一条关于身高的信息会带来一条关于体重的信息，因为两者是有关联的），并且是尺度无关的(scale-invariant)，即独立于测量尺度。修正了欧式距离中各个维度尺度不一致且相关的问题。

单个数据点的马氏距离：

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

数据点x,y之间的马氏距离：

$$D_M(x,y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}$$

其中 Σ 是多维随机变量的协方差矩阵。如果协方差矩阵是单位向量，也就是各维度独立同分布，马氏距离就变成了欧式距离。

```
import numpy as np
def mashi_distance(x,y):
    print x
    print y
    #马氏距离要求样本数要大于维数，否则无法求协方差矩阵
    #此处进行转置，表示10个样本，每个样本2维
    X=np.vstack([x,y])
```

```
    print X
    XT=X.T

    print XT

    #方法一：根据公式求解
    S=np.cov(X)   #两个维度之间协方差矩阵
    SI = np.linalg.inv(S) #协方差矩阵的逆矩阵
    #马氏距离计算两个样本之间的距离，此处共有4个样本，两两组合，共有6个距离。
    n=XT.shape[0]
    d1=[]
    for i in range(0,n):
        for j in range(i+1,n):
            delta=XT[i]-XT[j]
            d=np.sqrt(np.dot(np.dot(delta,SI),delta.T))
            print d
            d1.append(d)
```

1、切比雪夫距离（Chebyshev Distance）

2、明可夫斯基距离（Minkowski Distance）

3、海明距离（Hamming distance）

4、马哈拉诺比斯距离（Mahalanobis Distance）

2、常见的相似度（系数）算法

2.1 余弦相似度（Cosine Similarity）

$$cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

余弦相似度是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。

性质：给出的相似性范围从-1到1：-1意味着两个向量指向的方向正好截然相反，1表示它们的指向是完全相同的，0通常表示它们之间是独立的，而在这之间的值则表示中间的相似性或相异性。

代码：

```
>>> input1 = torch.randn(100, 128)
```

```
>>> input2 = torch.randn(100, 128)
>>> cos = nn.CosineSimilarity(dim=1, eps=1e-6)
>>> output = cos(input1, input2)
```

2.2 皮尔森相关系数（Pearson Correlation Coefficient）：用于度量两个变量X和Y之间的相关（线性相关），其值介于-1与1之间。

$$r = \frac{\sum_{i=1}^{n}(x_i - \hat{x})(y_i - \hat{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \hat{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \hat{y})^2}}$$

分子是两个集合的交集大小，分母是两个集合大小的几何平均值。是余弦相似性的一种形式。

皮尔逊相关系数具有平移不变性和尺度不变性，计算出了两个向量（维度）的相关性。在各个领域都应用广泛，例如，在推荐系统根据为某一用户查找喜好相似的用户,进而提供推荐，优点是可以不受每个用户评分标准不同和观看影片数量不一样的影响。

代码：

```
import numpy as np
x=np.random.random(8)
y=np.random.random(8)

#方法一：根据公式求解
x_=x-np.mean(x)
y_=y-np.mean(y)
d1=np.dot(x_,y_)/(np.linalg.norm(x_)*np.linalg.norm(y_))

#方法二：根据numpy库求解
X=np.vstack([x,y])
d2=np.corrcoef(X)[0][1]
```

2.3 KL散度（Kullback-Leibler Divergence）：即相对熵；是衡量两个分布(P、Q)之间的距离；越小越相似。

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot log\frac{p(x_i)}{q(x_i)}$$

表示的就是概率 q 与概率 p 之间的差异，很显然，散度越小，说明 概率 q 与概率 p 之间越接近，那么估计的概率分布于真实的概率分布也就越接近。

代码：

```
>>> torch.nn.functional.kl_div(input, target, size_average=None,
reduce=None, reduction='mean')
>>> F.kl_div(q.log(),p,reduction='sum')
#函数中的 p q 位置相反(也就是想要计算D(p||q)，要写成kl_div（q.log（），p）的形式)，而且q要先取 log
#reduction 是选择对各部分结果做什么操作，
```

2.4 Jaccard相似系数（Jaccard Coefficient）：主要用于计算符号度量或布尔值度量的样本间的相似度。两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的杰卡德相似系数，用符号J(A,B)表示。杰卡德系数值越大，样本的相似度越高。

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A|+|B|+|A\cap B|}$$

应用：假设样本A和样本B是两个n维向量，而且所有维度的取值都是0或1。例如，A（0,1,1,0）和B（1,0,1,1）。我们将样本看成一个集合，1表示集合包含该元素，0表示集合不包含该元素。p：样本A与B都是1的维度的个数；q：样本A是1而B是0的维度的个数；r：样本A是0而B是1的维度的个数；s：样本A与B都是0的维度的个数

那么样本A与B的杰卡德相似系数可以表示为： $J=\frac{p}{p+q+r}$

杰卡德相似度没有考虑向量中潜在数值的大小，而是简单的处理为0和1，不过，做了这样的处理之后，杰卡德方法的计算效率肯定是比较高的，毕竟只需要做集合操作。

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.random.random(8)>0.5
y=np.random.random(8)>0.5

x=np.asarray(x,np.int32)
y=np.asarray(y,np.int32)

#方法一：根据公式求解
up=np.double(np.bitwise_and((x != y),np.bitwise_or(x != 0, y !=
0)).sum())
down=np.double(np.bitwise_or(x != 0, y != 0).sum())
d1=(up/down)


#方法二：根据scipy库求解
```

```
X=np.vstack([x,y])
d2=pdist(X,'jaccard')
```

2.5 Tanimoto系数（广义Jaccard相似系数）

$$E_j(A,B)=\frac{A\cdot B}{||A||^2+||B||^2-A\cdot B}$$

其中A、B分别表示为两个向量，集合中每个元素表示为向量中的一个维度，在每个维度上，取值通常是[0, 1]之间的值（如果取值是二值向量0或1，那么Tanimoto系数就等同Jaccard距离），$A*B$ 表示向量乘积，$||A||^2=\sqrt{\sum_{i=1}^{n}A_i^2}$ 表示向量的模。

```
import numpy as np
def tanimoto_coefficient(p_vec, q_vec):
    """
    This method implements the cosine tanimoto coefficient metric
    :param p_vec: vector one
    :param q_vec: vector two
    :return: the tanimoto coefficient between vector one and two
    """
    pq = np.dot(p_vec, q_vec)
    p_square = np.linalg.norm(p_vec)
    q_square = np.linalg.norm(q_vec)
    return pq / (p_square + q_square - pq)
```

2.6 互信息(Mutual Information)：是信息论里一种有用的信息度量，它可以看成是一个随机变量 中包含的关于另一个随机变量的信息量，或者说是一个随机变量由于已知另一个随机变量而减少的不肯定性。衡量随机变量之间相互依赖程度的度量。

设两个随机变量（X,Y)的联合分布为P(x,y)，边缘分布分别为P(x),p(y)，互信息I(X;Y)是联合分布p(x,y)与边缘分布p(x)p(y)的相对熵，即：

$$I(X;Y)=\sum_{x\in X}\sum_{y\in Y}p(x,y)log\frac{p(x,y)}{p(x)p(y)}$$

```
#标准化互信息
from sklearn import metrics
if __name__ == '__main__':
    A = [1, 1, 1, 2, 3, 3]
    B = [1, 2, 3, 1, 2, 3]
```

```
    result_NMI=metrics.normalized_mutual_info_score(A, B)
    print("result_NMI:",result_NMI)
```

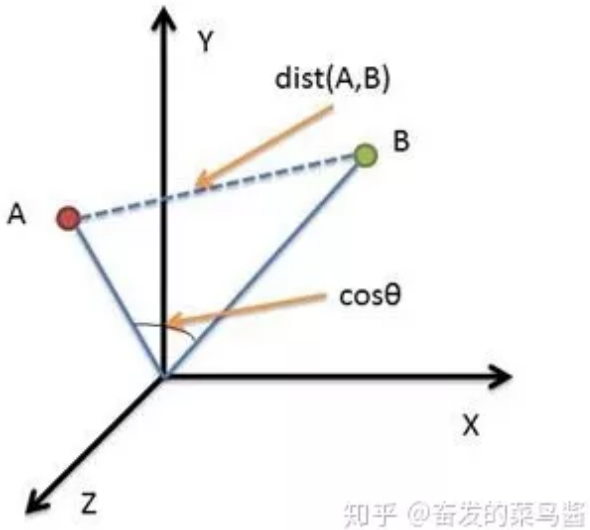
1、对数似然相似度/对数似然相似率

2、互信息/信息增益，相对熵/KL散度

3、信息检索——词频-逆文档频率（TF-IDF）

4、词对相似度——点间互信息

3、欧式距离vs余弦相似度



dist：欧式距离 cos：余弦相似度

欧氏距离衡量的是空间各点的绝对距离，跟各个点所在的位置坐标直接相关；而余弦距离衡量的是空间向量的夹角，更加体现在方向上的差异，而不是位置。如果保持A点位置不变，B点朝原方向远离坐标轴原点，那么这个时候余弦距离是保持不变的，而A、B两点的距离显然在发生改变，这就是欧氏距离和余弦距离之间的不同之处。

想要了解更多资讯，请扫描下方二维码，关注机器学习研究会