机器学习中有哪些距离度量方式

原创 fahai AI搞事情 2020-03-21

本文涉及到的距离度量方法：

1. 欧氏距离
2. 曼哈顿距离
3. 闵氏距离
4. 切比雪夫距离
5. 标准化欧氏距离
6. 马氏距离
7. 汉明距离
8. 编辑距离
9. DTW距离
10. 杰卡德相似系数
11. 余弦距离
12. 皮尔逊相关系数
13. 斯皮尔曼相关系数
14. 肯德尔相关性系数
15. 布雷柯蒂斯距离
16. 卡方检验
17. **交叉熵**
18. **相对熵**

I 欧式距离

欧式距离欧氏距离是最常见也是最常用的一种距离计算方式，也叫欧几里得距离、L_2距离。函数形式如下：

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$$

x,y表示两个n维向量，$d(x,y)$为两个n维向量的欧式距离。

python实现

```
import numpy as np
x = np.random.random(10)
y = np.random.random(10)
# n维向量
#方法一：根据公式求解
d1 = np.sqrt(np.sum(np.square(x - y)))
#方法二：根据np.linalg.norm求解
d2 = np.linalg.norm(x-y)
#方法三：根据scipy库求解
from scipy.spatial.distance import pdist
X = np.vstack([x,y])           #将x,y两个一维数组合并成一个2D数组 ； [[x1,x2,x3...],[y1,y2
d3 = pdist(X)                  #d2=np.sqrt(x1-y1)
# m * n维矩阵欧氏距离计算， 一行代表一个样本，一列代表一个特征，计算对应样本间欧氏距离
from sklearn.metrics import pairwise_distances
from scipy.spatial import distance_matrix
from scipy.spatial.distance import cdist
d1 = pairwise_distances(x.reshape(-1, 10), y.reshape(-1, 10)) # 运行时间次之 占cpu多
d2 = distance_matrix(x.reshape(-1, 10), y.reshape(-1, 10))
d3 = cdist(x.reshape(-1, 10), y.reshape(-1, 10)) # 运行时间最短 占cpu少，建议使用
```

II 曼哈顿距离

曼哈顿距离也称为城市街区距离、L_1距离，顾名思义，假设在曼哈顿街区从P点到Q点，我们不能直接穿过高楼大厦走直线的距离，而是表示走过的街道的距离。在二维坐标上的表示，即两个点在标准坐标系上的绝对轴距之和。具体公式表示：

$$d(x,y)=\sum_{k=1}^{n}|x_k-y_k|$$

x,y表示两个n维向量，$d(x,y)$为两个n维向量的曼哈顿距离。

python实现

```
import numpy as np
x = np.array([1,2,3])
y = np.array([4,5,6])
d1 = np.sum(np.abs(x-y))
d2 = np.linalg.norm(x-y, ord=1)
from scipy.spatial.distance import pdist
```

```
X=np.vstack([x,y])
d3=pdist(X,'cityblock')
```

III 闵氏距离

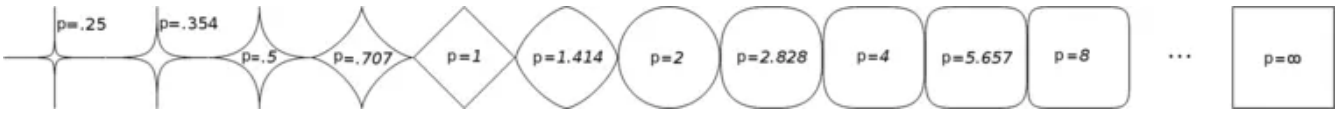
闵氏距离，全名闵可夫斯基距离，它不是一种距离，而是一组距离的定义，是对多个距离度量公式的概括性的表述。函数表达：

$$d(x,y)=(\sum_{k=1}^{n}|x_k-y_k|^p)^{1/p}$$

p取1或2时的闵氏距离是最为常用的：

p = 2即为欧氏距离。
p = 1时则为曼哈顿距离。
当p取无穷时的极限情况下，可以得到切比雪夫距离。

P为不同值时，等距离组成的形状：



python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
#方法一：根据公式求解,p=2
d1=np.sqrt(np.sum(np.square(x-y)))
d2 = np.linalg.norm(x-y, ord=2)
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d3=pdist(X,'minkowski',p=2)
```

IV 切比雪夫距离

切比雪夫距离，也叫L_∞度量，在国际象棋中，国王可以直行、横行、斜行，所以国王走一步可以移动到相邻的8个方格中的任意一个。国王从格子(x1，y1)走到格子(x2，y2)最少需要多少步，这个距离就叫切比雪夫距离。表示为：

$$d(x,y)=\max\left(|x_k-y_k|\right)$$

python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
d1=np.max(np.abs(x-y))
d2 = np.linalg.norm(x-y,ord=np.inf)
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d3=pdist(X,'chebyshev')
```

闵氏距离，包括曼哈顿距离、欧氏距离和切比雪夫距离都存在明显的缺点:

- 将各个分量的**量纲**(scale)，也就是“单位”当作相同的看待，量纲不同时，通常需要对数据做正规化；
- 没有考虑各个分量的**分布**（期望，方差等)可能是不同的；
- 各个维度必须是**互相独立**的，也就是“正交”的。

绿色表示两点之间欧氏距离，红色、蓝色和黄色代表等价的曼哈顿距离。

V 标准化欧氏距离

标准化欧氏距离是针对简单欧氏距离的缺点而作的一种改进方案。标准欧氏距离的思路：既然数据各维分量的分布不一样，那么我先将各个分量都“标准化”到均值、方差相等。通过统计学知识，假设样本集X的均值(mean)为m，标准差(standard deviation)为s，那么X的“标准化变量”表示为：

$$X^*=\frac{X-m}{s}$$

那么标准化欧式距离公式为：

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(\frac{x_k-y_k}{s_k})^2}$$

如果将方差的倒数看成是一个权重，也可称之为加权欧式距离。

python实现

```
import numpy as np
from scipy.spatial.distance import pdist
```

```
x=np.random.random(10)
y=np.random.random(10)
X=np.vstack([x,y])
sk=np.var(X,axis=0,ddof=1)
d1=np.sqrt(((x - y) ** 2 /sk).sum())
d2 = np.linalg.norm((x - y) /np.sqrt(sk), ord=2)
d3 = pdist(X, 'seuclidean',[0.5,1])
```

VI 马氏距离

闵氏距离比较直观，但是它与数据的分布无关，具有一定的局限性，标准化欧氏距离涉及到数据分布，但没有考虑到数据的相关性，而马氏距离在计算两个样本之间的距离时，考虑到了样本所在分布造成的影响，主要是因为：

1）不同维度的方差不同，进而不同维度对距离的重要性不同。

2）不同维度可能存在相关性，影响距离的度量。

定义：假设有M个样本向量$X_1,\ldots,X_m$，协方差矩阵记为S，均值记为向量μ，则其中样本向量X到μ的马氏距离表示为：

$$D(X)=\sqrt{(X-\mu)^T S^{-1}(X-\mu)}$$

其中，样本向量X_i与X_j之间的马氏距离定义为：

$$D(X)=\sqrt{(X_i-X_j)^T S^{-1}(X_i-X_j)}$$

若协方差矩阵S是单位矩阵（各个样本向量之间独立同分布）,则公式就成了：

$$D(X)=\sqrt{(X_i-X_j)^T(X_i-X_j)}$$

即：欧氏距离

若协方差矩阵是对角矩阵，公式变成了标准化欧氏距离。

python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
#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])
XT=X.T
#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10个样本，两两组合，共有45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))
        d1.append(d)
#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
d2=pdist(XT,'mahalanobis')
```

VII 汉明距离

两个等长字符串s1与s2之间的汉明距离定义为将其中一个变为另外一个所需要作的最小替换次数。

例如：

- 1011101 与 1001001 之间的汉明距离是 2。
- 2143896 与 2233796 之间的汉明距离是 3。
- "toned" 与 "roses" 之间的汉明距离是 3。

还可以用简单的匹配系数来表示两点之间的相似度，即：匹配字符数/总字符数。

python实现

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.random.random(10)>0.5
y=np.random.random(10)>0.5
x=np.asarray(x,np.int32)
y=np.asarray(y,np.int32)
#方法一：根据公式求解
d1=np.mean(x!=y)
#方法二：根据scipy库求解
X=np.vstack([x,y])
d2=pdist(X,'hamming')
```

VIII 编辑距离

汉明距离可以度量两个相同长度的字符串之间的相似度，如果比较的两个字符串长度不同，则不仅要进行替换，还要进行插入与删除的操作，这种情况下下，通常使用更加复杂的编辑距离进行相似度判断，即通过字符编辑（插入、删除或替换），将字符串A转换成字符串B所需要的最少操作数。

python实现

```
import numpy as np
# 动态规划
def edit_distance(str1, str2):
    len1 = len(str1)
    len2 = len(str2)
    dp = np.zeros((len1 + 1, len2 + 1))
    for i in range(len1 + 1):
        dp[i][0] = i
    for j in range(len2 + 1):
        dp[0][j] = j
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            delta = 0 if str1[i - 1] == str2[j - 1] else 1
            dp[i][j] = min(dp[i - 1][j - 1] + delta, min(dp[i - 1][j] + 1, dp[i][j - 1
    return dp[len1][len2]
```

IX DTW距离

DTW（Dynamic Time Warping，动态时间归整）是一种衡量两个长度不等的时间序列间相似度的方法，主要应用在语音识别领域，识别两段语音是否表示同一个单词。
大部分情况下，需要比较的两个序列在整体上具有非常相似的形状，但是这些形状并不是对一一对应的。所以我们在比较他们的相似度之前，需要将其中一个（或者两个）序列在时间轴下warping扭曲，以达到更好的对齐。而DTW就是实现这种warping扭曲的一种有效方法。DTW通过把时间序列进行延伸和缩短，来计算两个时间序列性之间的相似性。

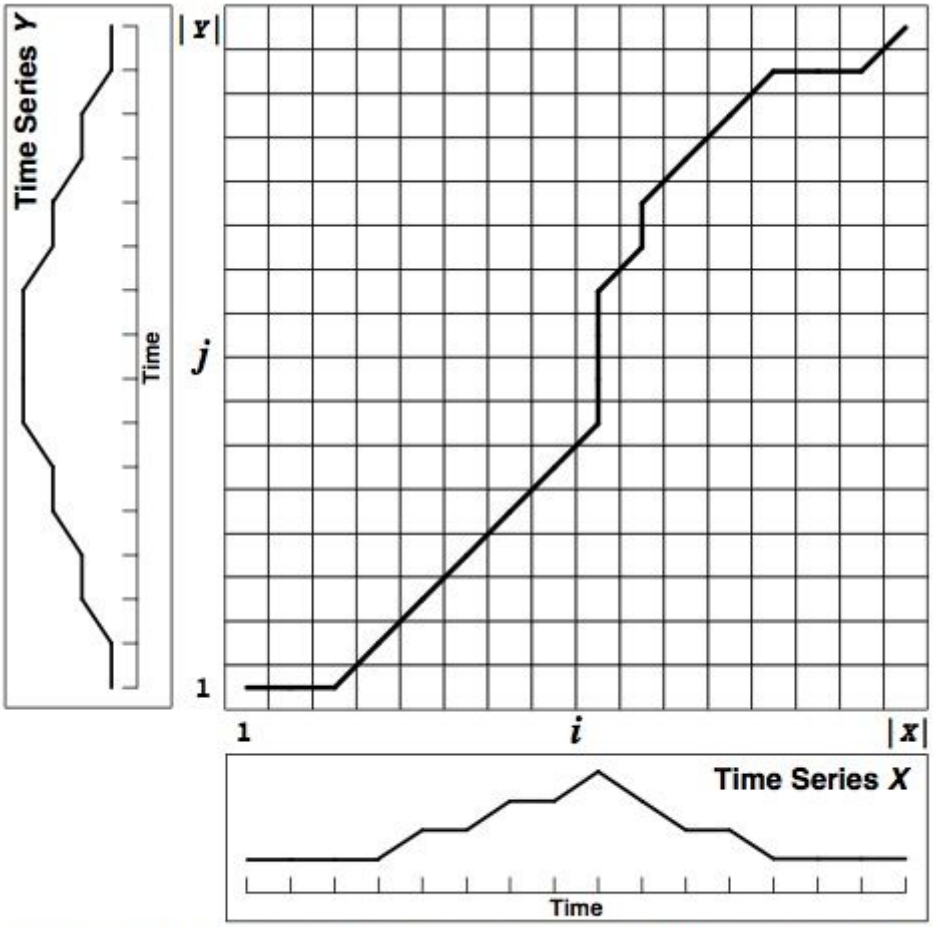


Figure 2. A cost matrix with the minimum-distance warp path traced through it.

python实现

```
def cal_dtw_distance(X, Y):  # dtw距离计算
    sign_len_N, num_features = X.shape                # 获取T的行数features，和列数N
    sign_len_M, num_features = Y.shape[1]            # 获取R的列数
    eudist_matrix = np.zeros((sign_len_N, sign_len_M))
    for i in range(num_features):                                    # 生成原始距离矩阵
        eudist_matrix += pow(np.transpose([X[i, :]])-Y[i, :], 2)
    eudist_matrix = np.sqrt(eudist_matrix)
    # 动态规划
    dtw_distance_matrix = np.zeros(np.shape(eudist_matrix))
    dtw_distance_matrix[0, 0] = eudist_matrix[0, 0]
    for n in range(1, sign_len_N):
        dtw_distance_matrix[n, 0] = eudist_matrix[n, 0] + dtw_distance_matrix[n-1, 0]
    for m in range(1, sign_len_M):
        dtw_distance_matrix[0, m] = eudist_matrix[0, m] + dtw_distance_matrix[0, m-1]
    # 三个方向最小
    for n in range(1, sign_len_N):
        for m in range(1, sign_len_M):
```

```
            dtw_distance_matrix[n, m] = eudist_matrix[n, m] + \
                min([dtw_distance_matrix[n-1, m], dtw_distance_matrix[n-1, m-1], dtw_d
    n = sign_len_N-1
    m = sign_len_M-1
    k = 1
    warping_path = [[sign_len_N-1, sign_len_M-1]]
    while n+m != 0:  # 匹配路径过程
        if n == 0:
            m = m-1
        elif m == 0:
            n = n-1
        else:
            number = np.argmin([dtw_distance_matrix[n-1, m], dtw_distance_matrix[n-1, m
            if number == 0:
                n = n-1
            elif number == 1:
                n = n-1
                m = m-1
            elif number == 2:
                m = m-1
        k = k+1
        warping_path.append([n, m])
    warping_path = np.array(warping_path)
    dtw_distance = dtw_distance_matrix[-1, -1]  # 序列距离
    return dtw_distance, warping_path
```

X 杰卡德相似系数

杰卡德相似性系数主要用于计算符号度量或布尔值度量的样本间的相似度，等于样本集交集个数和样本集并集个数的比值。

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$$

python实现

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.random.random(10)>0.5
```

```
y=np.random.random(10)>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')
```

XI 余弦距离

余弦距离，也称为余弦相似度，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方法。余弦值越接近1，就表明夹角越接近0度，也就表示两个向量越相似。

余弦距离计算公式:

$$cos(\theta) = \frac{\sum_{k=1}^{n} x_k y_k}{\sqrt{\sum_{k=1}^{n} x_k^2}\sqrt{\sum_{k=1}^{n} y_k^2}}$$

python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
#方法一：根据公式求解
d1=np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))
#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d2=1-pdist(X,'cosine')
```

XII 皮尔逊相关系数

皮尔逊相关系数（Pearson correlation），也叫相关系数，相比于余弦弦相似度只与向量方向有关，受向量的平移影响，皮尔逊相关系数具有平移不变性和尺度不变性，在夹角余弦公式中，如果将 x 平移到 x+1, 余弦值就会改变。皮尔逊相关系数值在-1.0到1.0之间，接近0的表示无相关性，接近1或者-1被称为具有强相关性，负数表示负相关，不过皮尔逊相关系数只能衡量两个随机变量间的线性相关性。

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{E((X-EX)(Y-EY))}{\sqrt{D(X)}\sqrt{D(Y)}}$$

如果将夹角余弦公式写成：

$$CosSim(x,y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\langle x,y\rangle}{||x||\ ||y||}$$

则皮尔逊相关系数则可表示为：

$$Corr(x,y) = \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum (x_i-\bar{x})^2}\sqrt{\sum (y_i-\bar{y})^2}} = \frac{\langle x-\bar{x},\ y-\bar{y}\rangle}{||x-\bar{x}||\ ||y-\bar{y}||} = CosSim(x-\bar{x}, y-\bar{y}) \quad (1)$$

因此，皮尔逊相关系数可以看作中心化后变量的余弦距离。

python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
#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]
#方法三：利用pandas库求解
import pandas as pd
X1 = pd.Series(x)
Y1 = pd.Series(y)
d3 = X1.corr(Y1, method="pearson")
d4 = X1.cov(Y1) / (X1.std() * Y1.std())
```

XIII 斯皮尔曼相关系数

斯皮尔曼相关系数又称斯皮尔曼秩相关系数，是利用两变量的秩次大小作线性相关分析，是根据原始数据的排序位置进行求解，对原始变量的分布不作要求，属于非参数统计方法，适用范围要广些。

$$\rho_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

计算过程：先对两个变量（X, Y）的数据进行排序，然后记下排序以后的位置（X′, Y′），（X′, Y′）的值就称为秩次，秩次的差值就是上面公式中的d_i，n是变量中数据的个数。

python实现

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
X1 = pd.Series(x)
Y1 = pd.Series(y)
n=x1.count()
x1.index=np.arange(n)
y1.index=np.arange(n)
#分部计算
d=(x1.sort_values().index-y1.sort_values().index)**2
dd=d.to_series().sum()
d1=1-n*dd/(n*(n**2-1))
d2 = X1.corr(y1,method='spearman')
```

XIV 肯德尔相关性系数

肯德尔相关性系数又称肯德尔秩相关系数，它所计算的对象是分类变量，因此它需要的数据集必须是分类变量。

$$KROCC = \frac{2(N_c - N_d)}{N(N-1)}$$

Nc表示主客观评价值中一致的值的个数，Nd则表示了主观评估值和客观评估值不一样的个数。

适用案例：评委对选手的评分（优、中、差等），我们想看两个（或者多个）评委对几位选手

的评价标准是否一致；或者医院的尿糖化验报告，想检验各个医院对尿糖的化验结果是否一致。

python实现

```
import pandas as pd
import numpy as np
#原始数据
x= pd.Series([3,1,4,2,5,3])
y= pd.Series([1,2,3,2,1,1])
d = x.corr(y,method="kendall")
```

XV 布雷柯蒂斯距离

布雷柯蒂斯距离（Bray Curtis Distance）主要用于生态学和环境科学，计算坐标之间的距离。该距离取值在[0,1]之间。它也可以用来计算样本之间的差异。

$$d(x,y)=\frac{\sum_{k=1}^{n}|x_k-y_k|}{\sum_{k=1}^{n}x_k+\sum_{k=1}^{n}y_k}$$

python实现

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.array([11,0,7,8,0])
y=np.array([24,37,5,18,1])
#方法一：根据公式求解
up=np.sum(np.abs(y-x))
down=np.sum(x)+np.sum(y)
d1=(up/down)
#方法二：根据scipy库求解
X=np.vstack([x,y])
d2=pdist(X,'braycurtis')
```

XVI 卡方检验

卡方检验应用统计样本的实际观测值与理论推断值之间的偏离程度，常用来检验某一种观测分布是不是符合某一类典型的理论分布（如二项分布，正态分布等)。如果卡方值越大，二者偏差程度越大；反之，二者偏差越小；若两个值完全相等时，卡方值就为0，表明理论值完全符合。

python实现

```
# -*- coding: utf-8 -*-
'''
卡方公式(o-e)^2 / e
期望值和收集到数据不能低于5，o(observed)观察到的数据，e（expected）表示期望的数据
(o-e)平方，最后除以期望的数据e
'''
import numpy as np
from scipy.stats import chisquare
list_observe=np.array([30,14,34,45,57,20])
list_expect=np.array([20,20,30,40,60,30])
#方法一:根据公式求解（最后根据c1的值去查表判断）
c1=np.sum(np.square(list_observe-list_expect)/list_expect)
#方法二：使用scipy库来求解
c2,p=chisquare(f_obs=list_observe, f_exp=list_expect)
'''
返回NAN，无穷小
'''
if p>0.05 or p=="nan":
   print("H0 win,there is no difference")
else:
   print("H1 win,there is difference")
```

XVII 交叉熵

如果一个随机变量X 服从 p(x)分布，q(x)用于近似p(x)的概率分布，那么随机变量和模型q之间的交叉熵定义为：

$$H(X,q) = -\sum_{x} p(x)log(q(x))$$

交叉熵在CNN分类中经常用到，用来作为预测值和真实标签值的距离度量。经过卷积操作后，最后一层出来的特征经过softmax函数后会变成一个概率向量，我们可以看作为是概率分布q, 而真实标签我们可以看作是概率分布p, 因此真实分布p和预测分布q的交叉熵就是我们要求的loss损失值。

python实现

```
import numpy as np
```

```
import tensorflow as tf
fea=np.asarray([6.5,4.2,7.4,3.5],np.float32)
label=np.array([1,0,0,0])
#方法一：根据公式求解
def softmax(x):
    return np.exp(x)/np.sum(np.exp(x),axis=0)
loss1=-np.sum(label*np.log(softmax(fea)))
#方法二：调用tensorflow深度学习框架求解
sess=tf.Session()
logits=tf.Variable(fea)
labels=tf.Variable(label)
sess.run(tf.global_variables_initializer())
loss2=sess.run(tf.losses.softmax_cross_entropy(labels,logits))
sess.close()
```

XVIII 相对熵

又称**KL散度**（Kullback–Leibler divergence，简称KLD），**信息散度**（information divergence），**信息增益**（information gain），相对熵是交叉熵与信息熵的差值。

python实现

```
import numpy as np
import scipy.stats
p=np.asarray([0.65,0.25,0.07,0.03])
q=np.array([0.6,0.25,0.1,0.05])
#方法一：根据公式求解
kl1=np.sum(p*np.log(p/q))
#方法二：调用scipy包求解
kl2=scipy.stats.entropy(p, q)
```

参考：

> 距离度量以及python实现(一、二、三、四):
> https://www.cnblogs.com/denny402/p/7027954.html