# 第2章 Theano基础

第1章我们介绍了NumPy，它是数据计算的基础，更是深度学习框架的基石。但如果直接使用NumPy计算大数据，其性能已成为一个瓶颈。

随着数据爆炸式增长，尤其是图像数据、音频数据等数据的快速增长，迫切需要突破这个瓶颈。需求就是强大动力！通过大咖们的不懈努力，在很多方面取得可喜进展，如硬件有GPU，软件有Theano、Keras、TensorFlow，算法有卷积神经网络、循环神经网络等等。

Theano是Python的一个库，为开源项目，在2008年，由Yoshua Bengio领导的加拿大蒙特利尔理工学院LISA实验室开发。对于解决大量数据的问题，使用Theano可能获得与手工用C实现差不多的性能。另外通过利用GPU，它能获得比CPU上的快很多数量级的性能。Theano开发者在2010年公布的测试报告中指出：在CPU上执行程序时，Theano程序性能是NumPy的1.8倍，而在GPU上是NumPy的11倍。这还是2010的测试结果，近些年无论是Theano还是GPU，性能都有显著提高。

这里我们把Theano作为基础来讲，除了其性能方面的跨越外，它还是“符合计算图”的开创者，当前很多优秀的开源工具，如TensorFlow、Keras等，都派生于或借鉴了Theano的底层设计。所以了解Theano的使用，将有助于我们更好学习TensorFlow、Keras等其他开源工具。

至于Theano是如何实现性能方面的跨越？如何用“符号计算图”来运算等，本章都将有所涉猎，但限于篇幅无法深入分析，只做一些基础性的介绍。涵盖的主要内容：

* 如何**安装Theano**
* **符号变量**是什么
* 如何设计**符号计算图**
* **函数**的功能
* **共享变量**的妙用

## 2.1安装

这里主要介绍Linux+Anaconda+theano环境的安装说明，Linux为CentOS或Ubuntu都可以，安装Python、NumPy、[SciPy](http://scipy.org/)等，建议使用Anaconda来安装，当然也可用pip进行安装。最好使用工具来安装，这样可以避免很多程序依赖的麻烦，而且日后的软件升级维护也很方便。

Theano支持CPU、GPU，如果使用GPU还需要安装其驱动程序如CUDA等，限于篇幅这里只介绍CPU的（TensorFlow将介绍基于GPU的安装），有关GPU的安装，大家可参考：http://www.deeplearning.net/software/theano/install.html

以下为安装主要步骤：

（1）安装anaconda

从anaconda官网：https://www.anaconda.com/download/下载linux环境最新的软件包，Python版本建议选择3系列的，2系列后续将不再维护。下载文件为一个sh程序包：如：

Anaconda3-4.3.1-Linux-x86\_64.sh，然后在下载目录下运行如下命令：

bash Anaconda3-4.3.1-Linux-x86\_64.sh

安装过程中按enter或y即可，安装完成后，程序提示是否把anaconda的binary加入到.bashrc配置文件中，加入后运行python、ipython时将自动使用新安装的Python环境。

安装完成后，你可用conda list命令查看已安装的库：

conda list

安装成功的话，应该能看到numpy、scipy、matplotlib、conda等库。

(2)安装theano

利用conda 来安装或更新程序

conda install theano

(3)测试

先启动python，然后导入theano模块，如果不报错，说明安装成功。

$ python

Python 3.6.0 |Anaconda custom (64-bit)| (default, Dec 23 2016, 12:22:00)

[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import theano

>>>

## 2.2符号变量

存储数据需要用到各种变量，Theano如何使用变量的呢？Theano用一种变量类型称为符号变量来表示变量，用TensorVariable表示，又称为张量（Tensor）,张量是Theano的核心元素（也是TensorFlow的核心元素），它是Theano表达式和运算操作的基本单位。张量可以是标量（scalar）、向量（vector）、矩阵（matrix）等的统称。具体来说，标量就是我们通常看到的0阶的张量，如12,a等，而向量和矩阵分别为1阶张量和2阶的张量。

如果通过这些概念，你还不很清楚，没有关系，可以结合以下一个实例，来直观感受一下。

首先定义三个标量：一个代表输入x、一个代表权重w、一个表示偏移量b，然后计算这些标量运算结果z=x\*w+b,Theano代码实现如下：

#导入需要的库或模块

import theano

from theano import tensor as T

#初始化张量

x=T.scalar(name='input',dtype='float32')

w=T.scalar(name='weight',dtype='float32')

b=T.scalar(name='bias',dtype='float32')

z=w\*x+b

#编译程序

net\_input=theano.function(inputs=[w,x,b],outputs=z)

#执行程序

print('net\_input: %2f'% net\_input(2.0,3.0,0.5))

打印结果

net\_input: 6.500000

通过以上实例我们不难看出，Theano本身是一个通用的符号计算框架,与非符号架构的框架不同，它先使用tensor variable初始化变量，然后将复杂的符号表达式编译成为函数模型，最后运行时传入实际数据进行计算。整个过程涉及三个步骤：定义符号变量，编译代码，执行代码。这节主要介绍第一步如何定义符号变量，其他步骤将在后续小节介绍。

如何定义符号变量？或定义符号变量有哪些方式？在Theano中定义符号变量有大致三种：使用内置的变量类型、自定义变量类型、转换其他的变量类型，具体如下：

**（1）使用内置的变量类型创建**

目前Theano支持7种内置的变量类型，分别是标量（scalar）、向量（vector）、行（row）、列(col)、矩阵(matrix)、tensor3、tensor4等。其中标量是0阶张量，向量为1阶张量，矩阵为二阶张量等，以下为创建内置变量的实例：

import theano

from theano import tensor as T

x=T.scalar(name='input',dtype='float32')

data=T.vector(name='data',dtype='float64')

其中，name指定变量名字，dtype指变量的数据类型。

**（2）自定义变量类型**

内置的变量类型只能处理4维及以下的变量，如果需要处理更高维的数据时，我们可以使用Theano的自定义变量类型，具体通过TensorType方法来实现：

import theano

from theano import tensor as T

mytype=T.TensorType('float64',broadcastable=(),name=None,sparse\_grad=False)

其中broadcastable是True或False的布尔类型元组，元组的大小等于变量的维度，如果为True，表示变量在对应维度上的数据可以进行广播，否则数据不能广播。

广播机制（broadcast）是一种重要机制，有了这种机制，就可以方便对不同维的张量进行运算，否则，就要手工把低维数据变成高维，利用广播机制系统自动利用复制等方法把低维数据补齐，numpy也有这种机制。以下我们通过图1-2所示的一个实例来说明广播机制原理：

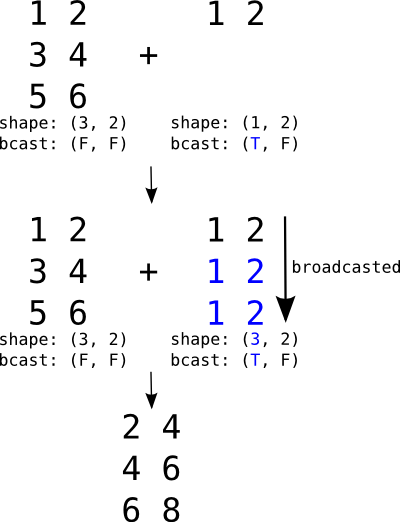


图2-1 广播机制

图2-1中矩阵与向量相加的具体代码如下：

import theano

import numpy as np

import theano.tensor as T

r = T.row()

r.broadcastable

# (True, False)

mtr = T.matrix()

mtr.broadcastable

# (False, False)

f\_row = theano.function([r, mtr], [r + mtr])

R = np.arange(1,3).reshape(1,2)

print(R)

#array([[1, 2]])

M = np.arange(1,7).reshape(3, 2)

print(M)

#array([[1, 2],

# [3, 4],

# [5, 6]])

f\_row(R, M)

#[array([[ 2., 4.],

# [ 4., 6.],

# [ 6., 8.]])]

**(3) 将Python类型变量或者Numpy类型变量转化为Theano共享变量**

共享变量是Theano实现变量更新的重要机制，后面我们会详细讲解。要创建一个共享变量，只要把一个Python对象或NumPy对象传递给shared函数即可,如下示例：

import theano

import numpy as np

import theano.tensor as T

data=np.array([[1,2],[3,4]])

shared\_data=theano.shared(data)

type(shared\_data)

## 2.3符号计算图模型

符号变量定义后，需要说明这些变量间的运算关系，如何描述变量间的运算关系？ Theano实际采用符号计算图模型来实现。首先创建表达式所需的变量，然后通过操作符（op）把这些变量结合在一起。如图2-1。

Theano处理符号表达式时通过把符号表达式转换为一个计算图（graph）来处理（TensorFlow也使用了这种方法，等到我们介绍TensorFlow时，大家可对比一下），符号计算图的节点有：variable、type、apply和op。

variable节点：即符号的变量节点，符号变量是符号表达式存放信息的数据结构，可以分为输入符号和输出符号。

type节点：当定义了一种具体的变量类型以及变量的数据类型时，Theano为其指定数据存储的限制条件。

apply节点：把某一种类型的符号操作符应用到具体的符号变量中，与variable不同，apply节点无须由用户指定，一个apply节点包括3个字段：op、inputs、outputs。

op节点：即操作符节点，定义了一种符号变量间的运算，如+、-、sum()、tanh（）等。

Theano是将符号表达式的计算表示成graphs。这些graphs是由Apply 和 Variable将节点连接而组成，它们分别与函数的应用和数据相连接。 操作由 Op 实例表示，而数据类型由 Type 实例表示。下面这段代码和图2-2，说明了这些代码所构建的结构。借助这个图或许有助于您进一步理解，如何将这些内容拟合在一起：

import theano

import numpy as np

import theano.tensor as T

x = T.dmatrix('x')

y = T.dmatrix('y')

z = x + y

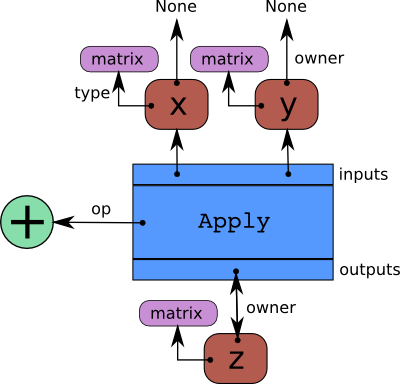


图2-2 符号计算图

图2-2中箭头表示指向python对象的引用。这里的蓝色盒子是一个 Apply 节点，红色盒子是 Variable 节点，绿色圆圈是Ops，紫色盒子是 Types。

在创建 Variables 之后，应用 Apply Ops得到更多的变量，这些变量仅仅是一个占位符，在function中作为输入。变量指向 Apply 节点的过程是用来表示函数通过它们的owner 域来生成它们 。这些Apply节点是通过它们的inputs和outputs域来得到它们的输入和输出变量。

x 和 y 的owner 域的指向都是None，是因为它们不是另一个计算的结果。如果它们中的一个是另一个计算的结果，那么owner域将会指向另一个蓝色盒。

## 2.4函数

上节我们介绍了如何把一个符号表达式转化为符号计算图，这节我们介绍函数的功能，函数是Theano的一个核心设计模块，它提供一个接口，把函数计算图编译为可调用的函数对象。前面介绍了如何定义自变量x(不需要赋值)，这节介绍如何编写函数方程。

**（1）函数定义的格式**

theano.function(inputs, outputs, mode=None, updates=None, givens=None, no\_default\_updates=False, accept\_inplace=False, name=None,rebuild\_strict=True, allow\_input\_downcast=None, profile=None, on\_unused\_input='raise')。

这里参数看起来很多，但常用的一般只用到三个，inputs表示自变量、outputs表示函数的因变量(也就是函数的返回值)，还有另外一个比较常用的是updates这个参数，这个一般用于神经网络共享变量参数更新，通常以字典或元组列表的形式指定；givens是一个字典或元组列表，记为[(var1,var2)],表示在每一次函数调用时，在符号计算图中，把符号变量var1节点替换为var2节点，该参数常用来指定训练数据集的batch大小。

我们看一个有多个自变量、同时又有多个因变量的函数定义例子：

import theano

x, y =theano.tensor.fscalars('x', 'y')

z1= x + y

z2=x\*y

#定义x、y为自变量，z1、z2为函数返回值（因变量）

f =theano.function([x,y],[z1,z2])

#返回当x=2，y=3的时候，函数f的因变量z1，z2的值

print(f(2,3))

打印结果

[array(5.0, dtype=float32), array(6.0, dtype=float32)]

在执行theano.function()时，Theano进行了编译优化，得到一个end-to-end的函数，传入数据调用f(2,3)时，执行的是优化后保存在图结构中的模型，而不是我们写的那行z=x+y，尽管二者结果一样。这样的好处是Theano可以对函数f进行优化，提升速度；坏处是不方便开发和调试，由于实际执行的代码不是我们写的代码，所以无法设置断点进行调试，也无法直接观察执行时中间变量的值。

**（2）自动求导**

Theano有了符号计算图2-2，自动计算导数就很容易了。tensor.grad()唯一需要做的就是从outputs逆向遍历到输入节点。对于每个Op，它都定义了怎么根据输入计算出偏导数。使用链式法则就可以计算出梯度了。利用Theano求导时非常方便，可以直接利用函数theano.grad()，比如求S函数的导数：

http://img.blog.csdn.net/20160213194717775?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

以下代码实现当x=3的时候，求s函数的导数：

import theano

x =theano.tensor.fscalar('x')#定义一个float类型的变量x

y= 1 / (1 + theano.tensor.exp(-x))#定义变量y

dx=theano.grad(y,x)#偏导数函数

f= theano.function([x],dx)#定义函数f，输入为x，输出为s函数的偏导数

print(f(3))#计算当x=3的时候，函数y的偏导数

打印结果

0.045176658779382706

**（3）更新共享变量参数**

在深度学习中通常需要迭代多次，每次迭代都需要更新参数。Theano如何更新参数呢？在theano.function函数中，有个非常重要的参数updates，updates是一个包含两个元素的列表或tuple，updates=[old\_w,new\_w]，当函数被调用的时候，这个会用new\_w替换old\_w,具体看下面这个例子。

import theano

w= theano.shared(1)#定义一个共享变量w，其初始值为1

x=theano.tensor.iscalar('x')

f=theano.function([x], w, updates=[[w, w+x]])#定义函数自变量为x，因变量为w，当函数执行完毕后，更新参数w=w+x

print(f(3))#函数输出为w

print(w.get\_value())#这个时候可以看到w=w+x为4

打印结果：1、4

在求梯度下降的时候，经常用到updates这个参数。比如updates=[w,w-α\*(dT/dw)]，其中dT/dw就是我们梯度下降的时候，代价函数对参数w的偏导数，α是学习速率。为便于大家的更全面的了解Theano函数一些使用，下面我们通过一个逻辑回归的完整实例来说明：

import numpy as np

import theano

import theano.tensor as T

rng = np.random

# 我们为了测试，自己生成10个样本，每个样本是3维的向量，然后用于训练

N = 10

feats = 3

D = (rng.randn(N, feats).astype(np.float32), rng.randint(size=N, low=0, high=2).astype(np.float32))

# 声明自变量x、以及每个样本对应的标签y(训练标签)

x = T.matrix("x")

y = T.vector("y")

#随机初始化参数w、b=0，为共享变量

w = theano.shared(rng.randn(feats), name="w")

b = theano.shared(0., name="b")

#构造代价函数

p\_1 = 1 / (1 + T.exp(-T.dot(x, w) - b)) # s激活函数

xent = -y \* T.log(p\_1) - (1-y) \* T.log(1-p\_1) # 交叉商代价函数

cost = xent.mean() + 0.01 \* (w \*\* 2).sum()# 代价函数的平均值+L2正则项以防过拟合，其中权重衰减系数为0.01

gw, gb = T.grad(cost, [w, b]) #对总代价函数求参数的偏导数

prediction = p\_1 > 0.5 # 大于0.5预测值为1，否则为0.

train = theano.function(inputs=[x,y],outputs=[prediction, xent],updates=((w, w - 0.1 \* gw), (b, b - 0.1 \* gb)))#训练所需函数

predict = theano.function(inputs=[x], outputs=prediction)#测试阶段函数

#训练

training\_steps = 1000

for i in range(training\_steps):

pred, err = train(D[0], D[1])

print (err.mean())#查看代价函数下降变化过程

## 2.5条件与循环

编写函数需要经常用到条件或循环，这节我们就简单介绍Theano如何实现条件判断或逻辑循环。

**（1）条件判断**

Theano是一种符号语言，条件判断不能直接使用Python的if语句。在Theano可以用ifelse 和 Switch来表示判定。这两个判决语句有何区别呢？

switch对每个输出变量进行操作，ifelse只对一个满足条件的变量操作,比如：

对语句：

switch(cond, ift, iff)

如果满足条件，则switch既执行ift也执行iff。对语句：

if cond then ift else iff

ifelse只执行ift或者只执行iff

以下通过一个示例进一步说明

from theano import tensor as T

from theano.ifelse import ifelse

import theano,time,numpy

a,b=T.scalars('a','b')

x,y=T.matrices('x','y')

z\_switch=T.switch(T.lt(a,b),T.mean(x),T.mean(y))#lt:a < b?

z\_lazy=ifelse(T.lt(a,b),T.mean(x),T.mean(y))

#optimizer:optimizer的类型结构（可以简化计算，增加计算的稳定性）

#linker:决定使用哪种方式进行编译(C/Python)

f\_switch = theano.function([a, b, x, y], z\_switch,mode=theano.Mode(linker='vm'))

f\_lazyifelse = theano.function([a, b, x, y], z\_lazy,mode=theano.Mode(linker='vm'))

val1 = 0.

val2 = 1.

big\_mat1 = numpy.ones((1000, 100))

big\_mat2 = numpy.ones((1000, 100))

n\_times = 10

tic = time.clock()

for i in range(n\_times):

f\_switch(val1, val2, big\_mat1, big\_mat2)

print('time spent evaluating both values %f sec' % (time.clock() - tic))

tic = time.clock()

for i in range(n\_times):

f\_lazyifelse(val1, val2, big\_mat1, big\_mat2)

print('time spent evaluating one value %f sec' % (time.clock() - tic))

打印结果

time spent evaluating both values 0.005268 sec

time spent evaluating one value 0.007501 sec

**（2）循环语句**

scan是theano中构建循环Graph的方法，scan是个灵活复杂的函数，任何用循环、递归或者跟序列有关的计算，都可以用scan完成。其格式如下：

theano.scan(fn, sequences=None, outputs\_info=None, non\_sequences=None, n\_steps=None, truncate\_gradient=-1, go\_backwards=False, mode=None, name=None, profile=False, allow\_gc=None, strict=False)

参数说明：

fn：函数类型，scan的一步执行。除了outputs\_info，fn可以返回sequences变量的更新updates。fn的输入变量顺序为sequences中的变量，outputs\_info的变量，non\_sequences中的变量。如果使用了taps，则按照taps给fn喂变量。taps的详细介绍会在后面的例子中给出。

sequences：scan进行迭代的变量，scan会在T.arange()生成的list上遍历，例如下面的polynomial 例子。

outputs\_info：初始化fn的输出变量，和输出的shape一致。如果初始化值设为None表示这个变量不需要初始值。

non\_sequences：fn函数用到的其他变量，迭代过程中不可改变（unchange）。

n\_steps：fn的迭代次数。

下面通过一个例子解释scan函数的具体使用方法。

代码实现思路是:先定义函数one\_step，它就是scan里的fn，其任务就是计算多项式的一项，scan函数返回的result里会保存多项式每一项的值，然后我们对result求和，就得到了多项式的值。

import theano

import theano.tensor as T

import numpy as np

# 定义单步的函数,实现a\*x^n

# 输入参数的顺序要与下面scan的输入参数对应

def one\_step(coef, power, x):

return coef \* x \*\* power

coefs = T.ivector() # 每步变化的值,系数组成的向量

powers = T.ivector() # 每步变化的值,指数组成的向量

x = T.iscalar() # 每步不变的值,自变量

# seq,out\_info,non\_seq与one\_step函数的参数顺序一一对应

# 返回的result是每一项的符号表达式组成的list

result, updates = theano.scan(fn = one\_step,

sequences = [coefs, powers],

outputs\_info = None,

non\_sequences = x)

# 每一项的值与输入的函数关系

f\_poly = theano.function([x, coefs, powers], result, allow\_input\_downcast=True)

coef\_val = np.array([2,3,4,6,5])

power\_val = np.array([0,1,2,3,4])

x\_val = 10

print("多项式各项的值: ",f\_poly(x\_val, coef\_val, power\_val))

#scan返回的result是每一项的值，并没有求和，如果我们只想要多项式的值，可以把f\_poly写成这样：

# 多项式每一项的和与输入的函数关系

f\_poly = theano.function([x, coefs, powers], result.sum(), allow\_input\_downcast=True)

print("多项式和的值：",f\_poly(x\_val, coef\_val, power\_val))

打印结果

多项式各项的值: [ 2 30 400 6000 50000]

多项式和的值： 56432

## 2.6共享变量

共享变量（shared variable）是实现机器学习算法参数更新的重要机制。shared函数会返回共享变量。这种变量的值在多个函数可直接共享。可以用符号变量的地方都可以用共享变量。但不同的是，共享变量有一个内部状态的值，这个值可以被多个函数共享。它可以存储在显存中，利用GPU提高性能。我们可以使用get\_value和set\_value方法来读取或者修改共享变量的值，使用共享变量实现累加操作。

import theano

import theano.tensor as T

from theano import shared

import numpy as np

#定义一个共享变量，并初始化为0

state = shared(0)

inc = T.iscalar('inc')

accumulator = theano.function([inc], state, updates=[(state, state+inc)])

# 打印state的初始值

print(state.get\_value())

accumulator(1) # 进行一次函数调用

# 函数返回后，state的值发生了变化

print(state.get\_value())

这里state是一个共享 变量，初始化为0，每次调用accumulator()，state都会加上inc。共享变量可以像普通张量一样用于符号表达式，另外，他还有自己的值，可以直接用.get\_value()和.set\_value()方法来访问和修改。

上述代码引入了函数中updates参数。updates参数是一个list，其中每个元素是一个元组(tuple)，这个tuple的第一个元素是一个共享变量，第二个元素是一个新的表达式。updatas中的共享变量会在函数返回后更新自己的值。updates的作用在于执行效率，updates多数时候可以用原地（in-place）算法快速实现，在GPU上，Theano可以更好地控制何时何地给共享变量分配空间，带来性能提升。最常见的神经网络权值更新，一般会用update实现。

## 2.7小结

Theano基于NumPy，但性能方面又高于NumPy。因Theano采用了张量(Tensor)这个核心元素，在计算方面采用符号计算模型,而且采用共享变量、自动求导、利用GPU等适合于大数据、深度学习的方法，其他很多开发项目也深受这些技术和框架影响。本章主要为后续介绍TensorFlow做个铺垫。

# 第3章线性代数

机器学习、深度学习的基础除了编程语言外，还有一个就是应用数学。它一般包括线性代数、概率与信息论、概率图、数值计算与最优化等。其中线性代数又是基础的基础。线性代数是数学的一个重要分支，广泛应用于科学和工程领域。大数据、人工智能的源数据在模型训练前，都需要转换为向量或矩阵，而这些运算正是线性代数的主要内容。

如在深度学习的图像处理中，如果1张图由28\*28像素点构成，那这28\*28就是一个矩阵。在深度学习的神经网络中，权重一般都是矩阵，我们经常把权重矩阵W与输入X相乘，输入X一般是向量，这就涉及矩阵与向量相乘的问题。诸如此类，向量或矩阵之间的运算在深度学习中非常普遍，也非常重要。

本章主要介绍如下内容：

* 标量、向量、矩阵和张量
* 矩阵和向量运算
* 特殊矩阵与向量
* 线性相关性及向量空间

## 3.1标量、向量、矩阵和张量

在机器学习、深度学习中，首先遇到的就是数据，如果按类别来划分，我们通常会遇到以下4种类型的数据。

#### 1.标量（scalar）

一个标量就是一个单独的数，一般用小写的变量名称表示，如a，x等。

#### 2.向量（vector）

向量就是一列数或一个一维数组，这些数是有序排列的。通过次序中的索引，我们可以确定向量中每个单独的数。通常我们赋予向量粗体的小写变量名称，如**x、y**等。一个向量一般有很多元素，这些元素如何表示？我们一般通过带脚标的斜体表示，如中的第一个元素，表示第二元素，依次类推。

当需要明确表示向量中的元素时，我们一般将元素排列成一个方括号包围的纵列：

**x=** （3.1）

我们可以把向量看作空间中的点，每个元素是不同的坐标轴上的坐标。

向量可以这样表示，那我们如何用编程语言如python来实现呢？如何表示一个向量？如何获取向量中每个元素呢？请看如下实例：

import numpy as np

a=np.array([1,2,4,3,8])

print(a.size)

print(a[0],a[1],a[2],a[-1])

打印结果如下：

5

1 2 4 8

这说明向量元素个数为5，向量中索引一般从0开始，如a[0]表示第一个元素1，a[1]

表示第二个元素2，a[2]表示第三个元素4，依次类推。这是从左到右的排列顺序，如果从右到左，我们可用负数来表示，如a[-1]表示第1个元素（注：从右到左），a[-2]表示第2个元素，依次类推。

#### 3.矩阵（matrix）

矩阵是二维数组，其中的每一个元素被两个索引而非一个所确定。我们通常会赋予矩阵粗体的大写变量名称，比如**A**。如果一个实数矩阵高度为m，宽度为n，那么我们说。

与向量类似，可以通过给定行和列的下标表示矩阵中元素，下标用逗号分隔，如表示**A**左上的元素，表示第一行第二列对应的元素，依次类推；这是表示单个元素，如果我们想表示1列或1行，该如何表示呢？我们可以引入冒号":"来表示，如第1行，可用A1,:表示，第2行，用A2,:表示，第1列用A:,1表示，第n列用A:,n表示。

如何用Python来表示或创建矩阵呢？如果希望获取其中某个元素，该如何实现呢？请看如下实例：

import numpy as np

A=np.array([[1,2,3],[4,5,6]])

print(A)

print(A.size) #显示矩阵元素总个数

print(A.shape) #显示矩阵现状，即行行和列数。

print(A[0,0],A[0,1],A[1,1])

print(A[1,:]) #打印矩阵第2行

打印结果：

[[1 2 3]

[4 5 6]]

6

(2, 3)

1 2 5

[4 5 6]

矩阵可以用嵌套向量生成，和向量一样，在Numpy中，矩阵元素的下标索引也是从0开始的。

#### 4.张量（tensor）

几何代数中定义的张量是向量和矩阵的推广，通俗一点理解的话，我们可以将标量视为零阶张量，向量视为一阶张量，那么矩阵就是二阶张量，三阶的就称为三阶张量，以此类推。在机器学习、深度学习中经常遇到多维矩阵，如一张彩色图片就是一个三阶张量，三个维度分别是图片的高度、宽度和色彩数据。

张量（tensor）也是深度学习框架TensorFlow的重要概念。TensorFlow由tensor（张量）+flow（流）构成。

同样我们可以用Python来生成张量及获取其中某个元素或部分元素，请看实例：

B=np.arange(16).reshape((2, 2, 4)) #生成一个3阶矩阵

print(B)

print(B.size) #显示矩阵元素总数

print(B.shape) #显示矩阵的维度

print(B[0,0,0],B[0,0,1],B[0,1,1])

print(B[0,1,:])

打印结果如下：

[[[ 0 1 2 3]

[ 4 5 6 7]]

[[ 8 9 10 11]

[12 13 14 15]]]

16

(2, 2, 4)

0 1 5

[4 5 6 7]

#### 5.转置(transpose)

转置以主对角线（左上到右下）为轴进行镜像操作，通俗一点来说就是行列互换。将矩阵A转置表示为，定义如下：

= （3.2）

如：

A=, =

向量可以看作只有一列的矩阵,把（3.1）式中向量**x**进行转置,得到下式。

用Numpy如何实现张量的转置？很简单，利用张量的T属性即可，示例如下：

C=np.array([[1,2,3],[4,5,6]])

D=C.T #利用张量的T属性(即转置属性)

print(C)

print(D)

打印结果如下：

[[1 2 3]

[4 5 6]]

[[1 4]

[2 5]

[3 6]]

## 3.2矩阵和向量运算

矩阵加法和乘法是矩阵运算中最常用的操作之一，两个矩阵相加，需要它们的形状相同，进行对应元素的相加，如：C=A+B,其中。矩阵也可以和向量相加，只要它们的列数相同，相加的结果是矩阵每行与向量相加，这种隐式地复制向量b到很多位置的方式称为广播(broadcasting)，以下我们通过一个代码实例来说明。

C=np.array([[1,2,3],[4,5,6]])

b=np.array([10,20,30])

D=C+b

print(D)

打印结果为：

[[11 22 33]

[14 25 36]]

两个矩阵相加，要求它们的形状相同，如果两个矩阵相乘，如A和B相乘，结果为矩阵C,矩阵A和B需要什么条件呢？条件比较简单，只要矩阵A的列数和矩阵B的行数相同即可。如果矩阵A的形状为mn，矩阵B的形状为np，那么矩阵C的形状就是mp，例如：

C=AB，则它们的具体乘法操作定义为：

即矩阵C的第i,j个元素为矩阵的A第i行与矩阵B的第j列的内积。

矩阵乘积有很多重要性质，如满足分配律A(B+C)=AB+AC 和结合律，A(BC)=(AB)C。大家思考一下是否满足交换律？

另外，转置也有很好性质，如：

两个矩阵可以相乘，矩阵也可和向量相乘，只要矩阵的列数等于向量的行数或元素个数。如：

Wx=b (3.3)

其中,b ,x

假设向量b=，则,, .......

## 3.3特殊矩阵与向量

上一节我们介绍了一般矩阵的运算，实际上在机器学习或深度学习中，我们还经常遇到一些特殊类型的矩阵，如可逆矩阵、对称矩阵、对角矩阵、单位矩阵、正交矩阵等等。这些特殊矩阵有特殊属性，下面我们逐一进行说明。

1.可逆矩阵

先简单介绍一下可逆矩阵，因后续需要用到。在（3.3）式中，假设矩阵W已知，向量b已知，如何求向量x？为求解向量x，我们需要引入一个称为逆矩阵的概念。而为了求逆矩阵，又牵涉到单位矩阵，何为单位矩阵？单位矩阵的结构很简单，就是所有沿主对角线上的元素都是1，而其他位置的元素都是0的方阵（行数等于列数的矩阵），一般记为，如：

任意向量和单位矩阵相乘，都不会改变。

矩阵A的逆记作，其定义为：A=

如果我们能找到（3.3）式中矩阵W的逆矩阵，，那么，我们只要在等式两边同时乘以，则可等到如下结果：

Wx=

x=

对此后续我们有更详细的讨论及代码实现。

2.对角矩阵

对角矩阵只有在主对角线上才有非零元素，其余都是0。从形式上来看，如果A为对角矩阵，当且仅当对所有ij，。对角矩阵可以是方阵（行数等于列数）也可以不是方阵，如下矩阵，就是一个对角矩阵。

对角矩阵有非常好的性质，这些性质使很多计算非常高效，在机器学习、深度学习中经常会遇到对角矩阵。

对于对角矩阵为方阵的情况，我们可以把对角矩阵简单记为：

diag()=diag() (3.4)

其中是由对角元素组成的向量，现在我们看一下对角矩阵在一些计算方面的奇妙之处。

假设现有向量，x=，则满足：

diag()*x*=*x*

=diag()

从上面两个式子可以看到对角矩阵的简洁高效。

（3）对称矩阵

对称矩阵，对于任意一个n阶方阵A,若A满足：A=成立，则称方阵A为对称矩阵。

（4）单位向量

任意给定的向量，若其范数为1，即，则称向量为单位向量。

（5）正交向量

假设现有向量，*x*=，则满足

*x =*0 (3.5)

成立，则称向量和向量*x*正交。这里表示向量的点积运算。

（6）正交矩阵

对于任意一个n阶方阵A，若矩阵的行向量之间互相正交，且行向量都是单位向量，即满足：

A（单位矩阵） （3.6）

则称矩阵A是一个正交矩阵。由（3.5）可以看出，若A是一个正交矩阵，则可推出=

## 3.4线性相关性及向量空间

前面我们介绍了向量、矩阵等概念，接下来我们将介绍向量组、线性组合、线性相关性、秩等重要概念。

由多个同维度的列向量构成的集合称为向量组，矩阵可以看成是由行向量或列向量构成的向量组。

**1.线性组合**

给定向量组X:其中,对任何一组实数,构成的表达式：

（3.7）

称为向量组X的一个线性组合，称为向量组的系数。

对于任意一个m维向量b，如果存在一组实数，使得：

=b

成立，则称向量b可以被向量组X:线性表示。

对于任意实数集{},由（3.7）式构成的所有向量集合，称为向量空间{，其中}

向量空间的概念有点抽象，我们举一个简单实例来说明这个概念，比如由三个向量构成的向量组{(1,0,0),(0,1,0),(0,0,1)} 和任何一组实数{}就构成了一个三维空间。

**2.线性相关**

对给定的一个向量组X:**，**如果存在不全为0的实数

=0

成立，则称向量组X为线性相关。反之，称向量组X为线性无关。

**3.向量组的秩**

假设在原向量组X：存在一个子向量组,不妨设为，r<n，满足：

（1）线性无关；

（2）向量组X的中任意r+1个向量构成的子向量组都是线性相关的。

那么，称向量组，r<n 是向量组X的一个最大线性无关组，最大线性无关向量组包含的向量个数r称为向量组X的秩。

秩是一个重要概念，运用非常广泛，实际上矩阵我们可以看成是一个向量组。如果把矩阵看成是由所有行向量构成的向量组，这样矩阵的行秩就等于行向量组的秩；如果把矩阵看成是由所有列向量构成的向量组，这样矩阵的列秩就等于列向量组的秩。矩阵的行秩与列秩相等，因此，把矩阵的行秩和列秩统称为矩阵的秩。

## 3.5范数

数有大小，向量也有大小，向量的大小我们通过范数（Norm）来衡量。范数在机器学习、深度学习中运用非常广泛，特别在限制模型复杂度、提升模型的泛化能力方面效果不错。p范数的定义如下：

(3.8)

其中p

直观上来看，向量x的范数是度量从原点到点x的距离，范数是将向量映射到非负值的函数，如果从广义来说，任意一个满足以下三个条件的函数，都可称为范数：

（1）非负性：f(x),且当f(x)=0时，必有x=0；

（2）三角不等式性：f(x+y)f(x)+f(y);

（3）齐次性： f()=||f(x)。

当p=1时，即范数，也称为绝对值范数，大小等于向量的每个元素绝对值之和，即：

（3.9）

当p=2时，即范数，也称为欧几里得范数，其大小表示从原点到当前点的欧几里得距离，即：

(3.10)

当p为时，即范数，也称为最大范数，它的值等于向量中每个元素的绝对值的最大值，即：

= (3.11)

前面主要介绍了利用范数来度量向量的大小，矩阵的大小如何度量呢？我们可以用类似的方法。在深度学习中，常用Frobenius范数来描述，即：

(3.12)

它有点类似向量的范数。

两个向量的点积可以用范数来表示，即：

（3.13）

其中

以上说了向量一种度量方式，即通过范数来度量向量或矩阵的大小，并有具体公式，在实际编程中如何计算向量的范数呢？这里我们还是以Python为例进行说明。

import numpy as np

import numpy.linalg as LA #导入Numpy中线性代数库

x=np.arange(0,1,0.1) #自动生成一个[0,1)间的10个数，步长为0.1

print(x)

x1= LA.norm(x,1) #计算1范数

x2= LA.norm(x,2) #计算2范数

xa=LA.norm(x,np.inf) #计算无穷范数

print(x1)

print(x2)

print(xa)

打印结果如下：

[ 0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]

4.5

1.68819430161

0.9

由此看出利用Python求向量的范数还是很方便的。

## 3.6特征值分解

许多数学对象可以分解成多个组成部分。特征分解就是使用最广的矩阵分解之一，即将矩阵分解成一组特征向量和特征值。本节讨论的矩阵都是方阵。

我们先介绍特征值、特征向量的概念。

设A是一个n阶方阵，如果存在实数和n维的非零向量*x*，满足：

A*x*=*x* (3.14)

那么把数称为方阵A的特征值，向量*x*称为矩阵A对应特征值的特征向量。

假设矩阵A有n个线性无关的特征向量{},它们对应的特征值为{}

把这n个线性无关的特征向量组合成一个新方阵，每一列是一个特征向量。

=[]

用特征值构成一个n阶对角矩阵，对角线的元素都是特征值。

那么，A的特征分解可表示为：

A=diag() (3.15)

注意，并不是所有方阵都能进行特征值分解，一个n阶方阵A能进行特征值分解的充分必要条件是它含有n个线性无关的特征向量。

这里我们介绍了给定一个方阵，如何求该方阵的特征向量和特征值？如何用编程语言实现？这些问题有了Python的帮忙，实现起来都非常简单，具体请看如下示例：

import numpy as np

a = np.array([[1,2],[3,4]]) # 示例矩阵

A1 = np.linalg.eigvals(a) # 得到特征值

A2,V1 = np.linalg.eig(a) # 其中A2也是特征值，B为特征向量

print(A1)

print(A2)

print(V1)

打印结果：

[-0.37228132 5.37228132]

[-0.37228132 5.37228132]

[[-0.82456484 -0.41597356]

[ 0.56576746 -0.90937671]]

【说明】

在numpy.linalg模块中：

eigvals() 计算矩阵的特征值

eig() 返回包含特征值和对应特征向量的元组

## 3.7奇异值分解

上节我们介绍了方阵的一种分解方式，如果矩阵不是方阵，是否能分解？如果能，该如何分解？这节我们介绍一种一般矩阵的分解方法，称为奇异值分解，这种方法应用非常广泛，如降维、推荐系统、数据压缩等等。

矩阵非常重要，所以其分解方法也非常重要，方法也比较多，除了特征分解法，还有一种分解矩阵的方法，被称为奇异值分解（SVD）。将矩阵分解为奇异向量和奇异值。通过奇异分解，我们会得到一些类似于特征分解的信息。然而，奇异分解有更广泛的应用。

每个实数矩阵都有一个奇异值分解，但不一定都有特征分解。例如，非方阵的矩阵就没有特征分解，这时我们只能使用奇异值分解。

奇异分解与特征分解类似，只不过这回我们将矩阵A分解成三个矩阵的乘积：

(3.16)

假设A是一个mn矩阵，那么U是一个mm矩阵，D是一个mn矩阵，V是一个nn矩阵。这些矩阵每一个都拥有特殊的结构，其中U和V都是正交矩阵，D是对角矩阵（注意，D不一定是方阵）。对角矩阵D对角线上的元素被称为矩阵A的奇异值。矩阵U的列向量被称为左奇异向量，矩阵V 的列向量被称右奇异向量。

SVD最有用的一个性质可能是拓展矩阵求逆到非方矩阵上。奇异值分解，看起来很复杂，如果用python来实现，却非常简单，具体请看如下示例：

import numpy as np

Data=np.mat([[1,1,1,0,0],

[2,2,2,0,0],

[3,3,3,0,0],

[5,5,3,2,2],

[0,0,0,3,3],

[0,0,0,6,6]])

u,sigma,vt=np.linalg.svd(Data)

#print(u)

print(sigma)

#转换为对角矩阵

diagv=np.mat([[sigma[0],0,0],[0,sigma[1],0],[0,0,sigma[2]]])

print(diagv)

#print(vt)

打印结果：

[ 1.09824632e+01 8.79229347e+00 1.03974857e+00 1.18321522e-15

2.13044868e-32]

[[ 10.98246322 0. 0. ]

[ 0. 8.79229347 0. ]

[ 0. 0. 1.03974857]]

## 3.8迹运算

迹运算返回的是矩阵对角元素的和:

(3.17)

迹运算在某些场合非常有用。若不使用求和符号，有些矩阵运算很难描述，而通过矩阵乘法和迹运算符号可以清楚地表示。例如，迹运算提供了另一种描述矩阵Frobenius 范数的方式：

= (3.18)

对迹运算的表达式，我们可以使用很多等式来表示。例如，迹运算在转置运算下是不变的:

Tr(A)=Tr()

多个矩阵相乘得到的方阵的迹，和将这些矩阵中的最后一个挪到最前面之后相乘的迹是相同的。当然，我们需要考虑挪动之后矩阵乘积依然有定义：

Tr(ABC)=Tr(CAB)=Tr(BCA)

利用Python的Numpy对矩阵求迹同样方便。请看以下示例。

C=np.array([[1,2,3],[4,5,6],[7,8,9]])

TrC=np.trace(C)

D=C-2

TrCT=np.trace(C.T)

TrCD=np.trace(np.dot(C,D))

TrDC=np.trace(np.dot(D,C))

print(TrC)

print(TrCT)

print(TrCD)

print(TrDC)

打印结果：

15

15

171

171

## 3.9实例：Python实现主成分分析

主成分分析（Principal Component Analysis，PCA）是一种统计方法。通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量叫主成分。

在许多机器学习、深度学习的应用中，往往需要处理大量样本或大的矩阵，多变量大样本无疑会为研究和应用提供丰富的信息，但也在一定程度上增加了数据采集的工作量。更重要的是在多数情况下，许多变量之间可能存在相关性，从而增加了问题分析的复杂性，同时对分析带来不便。如果分别对每个指标进行分析，分析往往是孤立的，而不是综合的。而盲目减少指标又会损失很多信息，且容易产生错误的结论。

因此需要找到一个合理有效的方法，在减少需要分析的指标或维度的同时，尽量减少原指标所含信息的损失，以达到对所收集数据进行全面分析的目的。由于各变量间存在一定的相关关系，因此有可能用较少的综合指标分别存在变量的各类信息。主成分分析就属于这类降维的方法。

如何实现以上目标呢？这里我们简要说明一下原理，然后使用Python来实现，至于详细的推导过程，大家可参考相关书籍或网上资料。

问题：设在n维空间中有m个样本点：{},假设m比较大，需要对这些点进行压缩，使其投影到k为空间中，其中k<n,同时使损失的信息最小。

该如何实现呢？以下简要说明一下思路。

设投影到k维空间后的新坐标系为{},其中,是标准的正交基向量，即满足：

， (i) (3.19)

设矩阵W={}是一个大小为nk维的正交矩阵，它是一个投影矩阵。X={}是训练数据集，为nm维的矩阵，由矩阵乘法的定义可知，投影到k维空间的点的坐标为Z=X

利用该坐标系重构数据，即把数据集Z从k为空间重新映射回n维空间，得到新的坐标点，

要使信息损失最小，一种合理的设想就是重构后的点与原来的数据点之间距离最小，据此，PCA可转换为求带约束的最优化问题：

= (3.20)

s.t. W=I

根据范数与矩阵迹的关系（3.18），上式可进一步简化为：

=

最后可以化简为：

（3.21）

s.t. W=I

然后，利用拉格朗日乘子法求解式（3.21）的最优解：

L(W,)=tr()+ （3.22）

最后对（3.22）式两端对w求导，并令导数为0，化简后就可得到：

(3.23)

由（3.23）式可知，W是由协方差矩阵的特征向量构成的特征矩阵，利用特征值分解的方法就可求出W。

以下我们Python具体实现一个PCA实例。以iris为数据集，该数据集可以通过load\_iris自动下载。

1）iris数据集简介：

Iris数据集是常用的分类实验数据集，由Fisher, 1936收集整理。Iris也称鸢尾花卉数据集。数据集包含150个数据集，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三类中的哪一类。

2）算法主要步骤为：

（1）对向量X进行去中心化

（2）计算向量X的协方差矩阵，自由度可以选择0或者1

（3）计算协方差矩阵的特征值和特征向量

（4）选取最大的k个特征值及其特征向量

（5）用X与特征向量相乘

3）代码实现

from sklearn.datasets import load\_iris

import numpy as np

from numpy.linalg import eig

def pca(X, k):

# Standardize by remove average

X = X - X.mean(axis = 0)

# Calculate covariance matrix:

X\_cov = np.cov(X.T, ddof = 0)

# Calculate eigenvalues and eigenvectors of covariance matrix

eigenvalues, eigenvectors = eig(X\_cov)

# top k large eigenvectors

klarge\_index = eigenvalues.argsort()[-k:][::-1]

k\_eigenvectors = eigenvectors[klarge\_index]

return np.dot(X, k\_eigenvectors.T)

iris = load\_iris()

X = iris.data

k = 2 #选取贡献最大的前2个特征

X\_pca = pca(X, k)

4）我们看一下各特征值的贡献率

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from numpy.linalg import eig

%matplotlib inline

iris = load\_iris()

X = iris.data

X = X - X.mean(axis = 0)

# 计算协方差矩阵

X\_cov = np.cov(X.T, ddof = 0)

#计算协方差矩阵的特征值和特征向量

eigenvalues, eigenvectors = eig(X\_cov)

tot=sum(eigenvalues)

var\_exp=[(i / tot) for i in sorted(eigenvalues,reverse=True)]

cum\_var\_exp=np.cumsum(var\_exp)

plt.bar(range(1,5),var\_exp,alpha=0.5,align='center',label='individual var')

plt.step(range(1,5),cum\_var\_exp,where='mid',label='cumulative var')

plt.ylabel('variance rtion')

plt.xlabel('principal components')

plt.legend(loc='best')

plt.show()

各特征值的贡献率如图3-1所示，可以看出，前2个特征值的方差贡献率超过95%，所以k取2有其合理性。

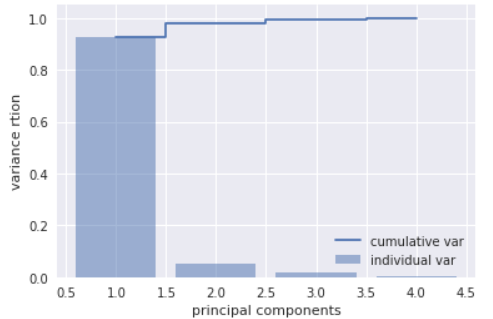


图3-1 各特征值的贡献率示意图

## 3.10小结

本章主要介绍线性代数中矩阵及向量有关概念，以及相关规则和运算等。线性代数是机器学习、深度学习的重要基础，与之相当的还有概率与信息论，我们将在下一章介绍。

# 第4章概率与信息论

机器学习、深度学习有三块基石：线性代数、概率与信息论、数值分析。线性代数上章已介绍，数值分析后续将介绍，三块基石缺一块都会地动山摇。本章讨论概率和信息论，概率是用于表示不确定性陈述的数学框架，即它是对事物不确定性的度量，而信息论主要研究信号或随机变量所包含的信息量。

在人工智能领域，概率法则告诉我们AI系统应该如何推理，概率和统计从理论上分析我们提出的AI系统的行为。

计算机科学的许多分支处理的对象都是完全确定的实体，但机器学习却大量使用概率。如果你了解机器学习的工作原理，或许你有更深的体会。因为机器学习大部分时候处理的都是不确定量或随机量。

概率论和信息论是众多学科的基础，也是机器学习、深度学习的重要基础。

如果你对概率论和信息论很熟悉了，可以跳过这章。如果你觉得本章内容还不够，希望进一步了解更多，可以参考相关专业教材。本章主要内容包括：

* 为何要学概率与信息论
* 样本空间与随机变量
* 概率分布
* 边缘概率
* 条件概率
* 期望、方差及协方差
* 贝叶斯定理
* 信息论

## 4.1为何要学概率、信息论

机器学习、深度学习需要借助概率、信息论？

概率研究对象不是预先知道或确定的事情，而是预先不确定或随机的事件，研究这些不确定或随机事件背后的规律或规则。或许有人会说，这些不确定或随机事件有啥好研究？他们本来就不确定或随机的，飘忽不定、不可捉摸。表面上看似如此，有句话说得好：偶然中有必然，必然中有偶然。就拿我们比较熟悉微积分来说吧，如果单看有限的几步，很多问题都显得杂乱无章，毫无规律可言，而且还很难处理，但是一旦加上一个无穷大（）这个“照妖镜”，其背后规律立显，原来难处理的也好处理了。如大数定律、各种分布等带给我们这样的认识。

信息论主要研究对一个信号包含信息的多少进行量化。它的基本思想是一个不太可能的事件居然发生了，其提供的信息量要比一个非常可能发生的事件更多。这中情况也似乎与我们的直觉相矛盾。

机器学习、深度学习与概率、信息论有哪些内在关联呢？

（1）被建模系统内在的随机性。例如一个假想的纸牌游戏，在这个游戏中我们假设纸牌被真正混洗成了随机顺序。

（2不完全观测。即使是确定的系统，当我们不能观测到所有驱动系统行为的所有变量或因素时，该系统也会呈现随机性。

（3）不完全建模。假设我们制作了一个机器人，它可以准确观察周围每一个对象的位置。在对这些对象将来的位置进行预测时，如果机器人采用的是离散化的空间，那么离散化的方法将使得机器人无法确定对象们的精确位置：因为每个对象都可能处于它被观测到的离散单元的任何一个角落。也就是说，当不完全建模时，我们不能明确的确定结果，这个时候的不确定，就需要借助概率来处理。

由此看来，概率、信息论很重要，机器学习、深度学习确实很需要它们。后续我们可以看到很多实例，见证概率、信息论在机器学习、深度学习中是如何发挥它们作用的。

## 4.2样本空间与随机变量

随机试验中，每一个可能的结果，在试验中发生与否，都带有随机性，所以称为随机事件。而所有可能结果构成的全体，称为样本空间。随机变量、样本空间这两个概念非常重要，以下就这两个概念作进一步说明。

* 样本空间

样本空间是一个实验或随机试验所有可能结果的集合，随机试验中的每个可能结果称为样本点。例如，如果抛掷一枚硬币，那么样本空间就是集合{正面，反面}。如果投掷一个骰子，那么样本空间就是{1,2,3,4,5,6}。

* 随机变量

随机变量，顾名思义，就是“其值随机而定”的变量，一个随机试验有许多可能结果，到底出现哪个预先是不知道的，其结果只有等到试验完成后，才能确定。如掷骰子，掷出的点数X是一个随机变量，它可以取1,2,3,4,5,6中的任何一个，到底是哪一个，要等掷了骰子以后才知道。因此，随机变量又是试验结果的函数，它为每一个试验结果分配一个值。比如，在一次扔硬币事件中，如果把获得的背面的次数作为随机变量X,则X可以取两个值，分别是0和1。如果随机变量X的取值是有限的或者是可数无穷尽的值，如：

X={}

则称 X为离散随机变量。如果X由全部实数或者由一部分区间组成，如：

X={*x|* },其中*a<b,*它们都为实数。

则称 X为连续随机变量，连续随机变量的取值是不可数及无穷尽的。

有些随机现象需要同时用多个随机变量来描述。例如对地面目标射击，弹着点的位置需要两个坐标(X,Y）才能确定，X,Y都是随机变量，而（X,Y）称为一个二维随机变量或二维随机向量，多维随机向量（,,...）含义依次类推。

## 4.3概率分布

概率分布用来描述随机变量（含随机向量）在每一个可能状态的可能性大小。概率分布有不同方式，这取决于随机变量是离散的还是连续的。

对于随机变量X,其概率分布通常记为P(X=*x*),或X P(*x*)，表示X服从概率分布P(*x*)。

概率分布描述了取单点值的可能性或概率，但在实际应用中，我们并不关心取某一值的概率，特别是对连续型随机变量，它在某点的概率都是0，这个后续章节将介绍。因此，我们通常比较关心随机变量落在某一区间的概率，为此，引入分布函数的概念。

定义：设X是一个随机变量，是任意实数值，函数：

F()=P(X) (4.1)

称为随机变量X的分布函数。

由（4.1）式不难发现，对任意的实数（）,有：

P(X)=P(X) -P(X)

=F() -F() (4.2)

成立。式（4.2）表明，若随机变量X的分布函数已知，那么可以求出X落在任意一区间[]的概率。

### 4.3.1 离散型随机变量

设*，，…，*是随机变量X的所有可能取值，对每个取值，X = 是其样本空间S上的一个事件，为描述随机变量X，还需知道这些事件发生的可能性(概率)。

设离散型随机变量X的所有可能取值为 (i=1，2，…,n)。

　　P(X = ) = 1,2,...n

　　称之为X的概率分布或分布律，也称概率函数。

常用表格形式来表示X的概率分布:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X |  |  | .... |  |
|  |  |  | ... |  |

由概率的定义， (i = 1,2,...)必然满足:

(1)

(2) =1

例1：某篮球运动员投中篮圈的概率是0.8，求他两次独立投篮投中次数X的概率分布。

解 X可取0，1，2为值，记={第i次投中篮圈}，i=1，2，则P() = P() = 0.8

由此不难得到下列各情况的概率：

投了两次没一次投中，即：P(X=0)=P()=P()P()=0.2x0.2=0.04

投了两次只投中一次，即：P(X=1)=P()=P(

=0.2x0.8+0.8x02=0.32

投了两次两次都投中，即：P(X=2)=P()=P()P()=0.8x0.8=0.64

且P(X=0)+P(X=1)+P(X=2)=0.04+0.32+0.64=1

于是随机变量X的概率分布可表示为：

|  |  |  |  |
| --- | --- | --- | --- |
| X | 0 | 1 | 2 |
|  | 0.04 | 0.32 | 0.64 |
|  |  |  |  |

若已知一个离散型随机变量X的概率分布：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X |  |  | .... |  |
|  |  |  | ... |  |

则由概率的可列可加性，可得随机变量X的分布函数为：

F(*x*)=P()= (4.3)

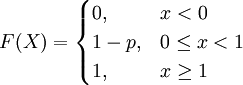
例如，设X的概率分布由例1给出，则

　　 F(2)=P(X2)=P(X=0)+P(X=l)=0.04+0.32=0.36

常见的离散随机变量的分布有：

（1）两点分布

若随机变量X只可能取0和1两个值，且它的分布列为P(X=1)=p，P(X = 0) = l − P其中(0 < P < 1)，则称X服从参数为p的两点分布，记作X~B(1, p)。其分布函数为



（2）二项分布

二项分布是重要的离散概率分布之一，由瑞士数学家雅各布·伯努利（Jokab Bernoulli）提出。一般用二项分布来计算概率的前提是，每次抽出样品后再放回去，并且只能有两种试验结果，比如黑球或红球，正品或次品等。二项分布指出，假设某样品在随机一次试验出现的概率为p，那么在n次试验中出现k次的概率为：

P(X=*k*)=

假设随机变量X满足二项分布，且知道n，p，k等参数，我们如何求出各种情况的概率值呢？方法比较多，这里介绍一种比较简单的方法，利用scipy库的统计接口stats即可，具体如下：

import numpy as np

import matplotlib.pyplot as plt

import math

from scipy import stats

%matplotlib inline

n = 20

p = 0.3

k = np.arange(0,41)

#定义二项分布

binomial = stats.binom.pmf(k,n,p)

#二项分布可视化

plt.plot(k, binomial, 'o-')

plt.title('binomial:n=%i,p=%.2f'%(n,p),fontsize=15)

plt.xlabel('number of success')

plt.ylabel('probalility of success', fontsize=15)

plt.grid(True)

plt.show()

运行后的二项分布图如图4-1所示。

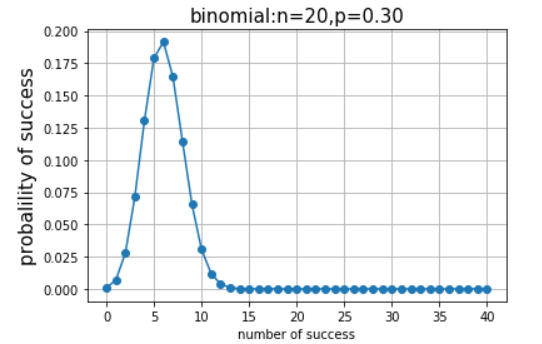


图4-1 二项分布图

（3）泊松(Poisson)分布

若随机变量X所有可能取值为0，1，2，…，它取各个值的概率为：

P(X=*k*)= (k=0,1,2，…)

这里介绍了离散型随机变量的分布情况，如果X是连续型随机变量，其分布函数通常通过密度函数来描述，具体请看下一节。

### 4.3.2 连续型随机变量

与离散型随机变量不同，连续型随机变量采用概率密度函数来描述变量的概率分布。如果一个函数f(x)是密度函数，满足以下三个性质，我们就称f(x)为概率密度函数。

（1）f(x)注意这里不要求f(x)。

（2）。

（3）对于任意实数和，且,有：

P()= (4.4)

第（2）个性质表明，概率密度函数f(x)与x轴形成的区域的面积等于1，第（3）个性质表明，连续随机变量在区间[]的概率等于密度函数在区间[]上的积分，也即是与X轴在[[]内形成的区域的面积，如图4-2所示。

0

x

f(x)

图4-2 概率密度函数

对连续型随机变量在任意一点的概率处处为0。

假设有任意小的实数，由于{X=*x*},由式（4.1）分布函数的定义可得：

=F*(x*)-F() (4.5)

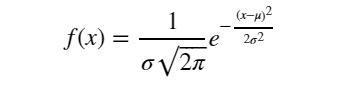
令,由夹逼准则，式（4.5）可求得：

P(X=*x*)=0 （4.6）

式（4.6）表明，对于连续型随机变量，它在任意一点的取值的概率都为0。因此，在连续型随机变量中，当讨论区间的概率定义时，一般对开区间和闭区间不加区分，即：

P()=P()=P()=P()成立。

最常见的正态分布的密度函数为：



这个连续分布被称之为正态分布，或者高斯分布。其密度函数的曲线呈对称钟形，因此又被称之为钟形曲线，其中μ是平均值，σ是标准差（何为平均值、标准差后续我们会介绍）。正态分布是一种理想分布。

正态分布如何用Python实现呢？同样，我们可以借助其scipy库中stats来实现，非常方便。

import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

%matplotlib inline

#平均值或期望值

mu=0

#标准差

sigma1=1

sigma2=2

#随机变量的取值

x=np.arange(-6,6,0.1)

y1=stats.norm.pdf(x,0,1) #定义正态分布的密度函数

y2=stats.norm.pdf(x,0,2) #定义正态分布的密度函数

plt.plot(x,y1,label='sigma is 1')

plt.plot(x,y2,label='sigma is 2')

plt.title('normal $\mu$=%.1f,$\sigma$=%.1f or %.1f '%(mu,sigma1,sigma2))

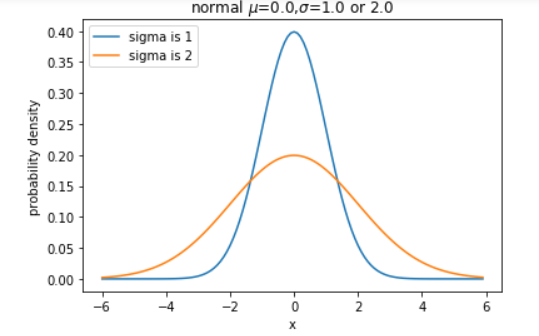
plt.xlabel('x')

plt.ylabel('probability density')

plt.legend(loc='upper left')

plt.show()

sigmal系统与正态分布如图4-3所示。

 图4-3 sigmal系统与正态分布

正态分布的取值可以从负无穷到正无穷。这里我们为便于可视化，只取把X数据定义在[-6,6]之间，用stats.norm.pdf得到正态分布的概率密度函数。另外从图形可以看出，上面两图的均值u都是0，只是标准差（）不同，这就导致图像的离散程度不同，标准差大的更分散，个中原因，我们在介绍随机变量的数字特征时将进一步说明。

## 4.4边缘概率

对于多维随机变量，如二维随机变量(X,Y),假设其联合概率分布为F(x,y),我们经常遇到求其中一个随机变量的概率分布的情况。这种定义在子集上的概率分布称为边缘概率分布。

例如，假设有两个离散的随机变量X,Y,且知道P(X,Y),那么我们可以通过下面求和的方法，得到边缘概率P(X):

P(X=x)= (4.7)

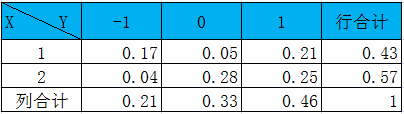
对于连续型随机变量（X,Y），我们可以通过联合密度函数f(x,y)来得到边缘密度函数。

f(x)= （4.8）

f(y)= （4.9）

边缘概率如何计算呢？我们通过一个实例来说明。假设有两个离散型随机变量X,Y，其联合分布概率如表4-1所示。

表4-1：X与Y的联合分布



如果我们要求P(Y=0)的边缘概率，根据式（4.7）可得：

P(Y=0)=P(X=1,Y=0)+P(X=2,Y=0)=0.05+0.28=0.33

## 4.5条件概率

上一节我们介绍了边缘概率，它是多维随机变量一个子集（或分量）上的概率分布。对于含多个随机变量的事件中，经常遇到求某个事件在其他事件发生的概率，例如，在表4-1的分布中，假设我们要求当Y=0的条件下，求X=1的概率？这种概率叫作条件概率。条件概率如何求？我们先看一般情况。

设有两个随机变量X,Y,我们将把X=x，Y=y发生的条件概率记为P(Y=y|X=x),那么这个条件概率可以通过以下公式计算：

P(Y=y|X=x)= (4.10)

条件概率只有在P(X=x)>0时，才有意义，如果P(X=x)=0，即X=x不可能发生，以它为条件就毫无意义。

现在我们来看上面这个例子，根据式（4.10），我们要求的问题就转换为：

P(X=1|Y=0)= （4.11）

其中P(Y=0)是一个边缘概率，其值为：P(X=1,Y=0)+P(X=2,Y=0)=0.05+0.28=0.33

而P(X=1，Y=0)=0.05.故P(X=1|Y=0)=0.05/0.33=5/33

式（4.10）为离散型随机变量的条件概率，对连续型随机变量也有类似公式。假设（X,Y）为二维连续型随机变量，它们的密度函数为f(*x,y*),关于Y的边缘概率密度函数为,且满足,假设

（4.12）

为在Y=y条件下，关于X的条件密度函数，则

4.13）

称为在Y=y的条件下，关于X的条件分布函数。

同理，可以得到，在X=x的条件下，关于Y的条件密度函数；

4.14）

在X=x的条件下，关于Y的条件分布函数为：

（4.15）

## 4.6条件概率的链式法则

条件概率的链式法则，又称为乘法法则，把式（4.10）变形，可得到条件概率的乘法法则：

P(X,Y)=P(X)P(Y|X) (4.16)

根据式（4.16）可以推广到多维随机变量，如：

P(X,Y,Z)=P(Y,Z)P(X|Y,Z)

而P(Y,Z)=P(Z)P(Y|Z)

由此可得：P(X,Y,Z)=P(X|Y,Z)P(Y|Z)P(Z) （4.17）

推广到n维随机变量的情况，可得：

（4.18）

## 4.7独立性及条件独立性

两个随机变量X,Y,如果它们的概率分布可以表示为两个因子的乘积，且一个因子只含x，另一个因子只含y，那么我们就称这两个随机变量互相独立。这句话可能不好理解，我们换一种方式的来表达。或许更好理解。

如果对P(X=*x*,Y=*y*)=P(X=*x)*P(Y=*y*) 成立，那么随机变量X,Y互相独立。

在机器学习中，随机变量为互相独立的情况非常普遍，一旦互相独立，联合分布的计算就变得非常简单。

这是不带条件的随机变量的独立性定义，如果两个随机变量带有条件，如P(X,Y|Z),它的独立性如何定义呢？这个与上面的定义类似。具体定义如下：

如果对P(X=*x*,Y=*y|Z=z*)=P(X=*x|Z=z)*P(Y=*y|Z=z*) 成立

那么随机变量X,Y在给定随机变量Z时是条件独立的。

为便于表达，如果随机变量X,Y互相独立，又可记为XY,如果随机变量X,Y在给定时互相独立，则可记为XY|Z。

以上主要介绍离散型随机变量的独立性和条件独立性，如果是连续型随机变量，我们只要把概率换成随机变量的密度函数即可。

假设X,Y为连续型随机变量，其联合概率密度函数为f(x,y),(x),(y)分别表示关于X,Y的边缘概率密度函数，如果f(x,y)=(x)(y)成立，则称随机变量X,Y互相独立。

## 4.8期望、方差及协方差

在机器学习、深度学习中经常需要分析随机变量的数据特征及随机变量间的关系等，对于这些指标的衡量在概率统计中有相关的内容，如用来衡量随机变量的取值大小的期望(Expectation)值或平均值、衡量随机变量数据离散程度的方差(Variance)、揭示随机向量间关系的协调方差(Convariance)等。这些衡量指标的定义及公式就是本节主要内容。

首先我们看随机变量的数学期望的定义：

对离散型随机变量X,设其分布律为：

P(X=)= k=1,2,3，... （4.19）

若级数绝对收敛，则称级数的值为随机变量X的数学期望，记为：

E(X)= (4.20)

对于连续型随机变量X,设其概率密度函数为f(x),若积分：

(4.21)

绝对收敛，则积分的值称为随机变量X的数学期望，记为：

E(X)= (4.22)

如果是随机变量函数，如随机变量X的g(x)的期望，公式与式（4.21）或式（4.22）类似，只要把x换成g(x)即可，即随机变量函数g(x)的期望为：

设Y=g(X) 则E(Y)=E(g(X))= （4.23）

或E(g(X))= （4.24）

期望有一些重要性质，具体如下：

设a,b为一个常数，X和Y是两个随机变量。则有：

（1）E(a)=a

（2）E(aX)=aE(X)

（3）E(aX+bY)=aE(X)+bE(Y) (4.25)

（4）当X和Y相互独立时，则有：

E(XY)=E(X)E(Y) (4.26)

数学期望也常称为均值，即随机变量取值的平均值之意，当然这个平均，是指以概率为权的加权平均。期望值可大致描述数据的大小，但无法描述数据的离散程度，这里我们介绍一种刻画随机变量在其中心位置附近离散程度的数字特征，即方差。如何定义方差？

假设随机向量X有均值E(X)=a。试验中，X取的值当然不一定恰好是a，可能会有所偏离。偏离的量X-a本身也是一个随机变量。如果我们用X-a来刻画随机变量X的离散程度，当然不能取X-a的均值，因E(X-a)=0 ,说明正负偏离抵消了，当然我们可以取|X-a|这样可以防止正负抵消的情况，但绝对值在实际运算时很不方便。人们就考虑另一种方法，先对X-a平方以便消去符号，然后再取平均得E或E把它作为度量随机变量X的取值的离散程度衡量，这个量就叫做X的方差（即差的方）,随机变量的方差记为：

Var(X)=E (4.27)

方差的平方根被称为标准差。

对于多维随机向量，如二维随机向量(X,Y)如何刻画这些分量间的关系？显然均值、方差都无能为力。这里我们引入协方差的定义，我们知道方差是X-EX乘以X-EX的均值，如果我们把其中一个换成Y-EY，就得到E(X-EX)(Y-EY),其形式接近方差，又有X,Y两者的参与，由此得出协方差的定义，随机变量X,Y的协方差，记为：Cov(X,Y)

Cov(X,Y) =E(X-EX)(Y-EY) (4.28)

协方差的另一种表达方式：

Cov(X,Y) =E(XY)-EXEY (4.29)

方差可以用来衡量随机变量与均值的偏离程度或随机变量取值的离散度，而协方差则可衡量随机变量间的相关性强度，如果X与Y独立，那么它们的协方差为0。反之，并不一定成立，独立性比协方差为0的条件更强。不过如果随机变量X、Y都是正态分布，此时独立和协方差为0是一个概念。

当协方差为正时，表示随机变量X、Y为正相关；如果协方差为负，表示随机变量X、Y为负相关。

为了更好的衡量随机变量间的相关性，我们一般使用相关系数来衡量，相关系数将每个变量的贡献进行归一化，使其只衡量变量的相关性而不受各变量尺寸大小的影响，相关系统的计算公式如下：

(4.30)

由式（4.30）可知，相关系统是在协方差的基础上进行了正则化，从而把相关系数的值限制在[-1,1]之间。如果，说明随机变量X、Y是线性相关的，即可表示为Y=kX+b，其中k，b为任意实数，且k>0;如果，说明随机变量X、Y是负线性相关的，即可表示为Y=-kX+b，其中k>0。

上面我们主要以两个随机变量为例，实际上协方差可以推广到n个随机变量的情况或n维的随机向量。对n维的随机向量，我们可以的第一个nxn的协方差矩阵，而且满足：

（1）协方差矩阵为对称矩阵，即Cov()=Cov()

（2）协方差矩阵的对角元素为方差：即Cov()=Var(

求随机变量的方差、协方差、相关系统等，使用Python的numpy相关的函数，如用numpy.var求方差，numpy.cov求协方差，使用numpy.corrcoef求相关系数，比较简单，这里就不展开来说。

在机器学习中多维随机向量，通常以矩阵的方式出现，所以求随机变量间的线性相关性，就转换为求矩阵中列或行的线性相关性。这里我们举一个简单实例，来说明如果分析向量间的线性相关性并可视化结果。这个例子中使用的随机向量（或特征值）共有三个，一个是气温(temp),一个体感温度(atemp),一个是标签（label）说明共享单车每日出租量，以下是这三个特征的部分数据：

表4-2 共享单车示例数据

|  |  |  |
| --- | --- | --- |
| temp | atemp | label |
| 0.24 | 0.2879 | 16 |
| 0.22 | 0.2727 | 40 |
| 0.22 | 0.2727 | 32 |
| 0.24 | 0.2879 | 13 |

这里使用Python中数据分析库pandas及画图库matplotlib、sns等。

###探索特征间分布、相关性等

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

data\_pd=data1.toPandas()

sns.set(style='whitegrid',context='notebook')

cols=['temp','atemp','label']

sns.pairplot(data\_pd[cols],size=2.5)

plt.show()

从图4-4可以看出，特征temp与atemp是线性相关的，其分布接近正态分布。

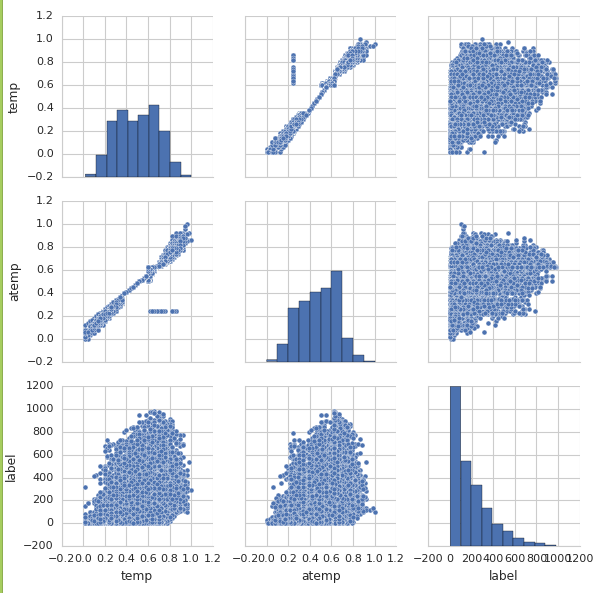


图4-4 特征分布及相关性

## 4.9贝叶斯定理

贝叶斯定理是概率论中的一个定理，它跟随机变量的条件概率以及边缘概率分布有关。在有些关于概率的解释中，贝叶斯定理（贝叶斯公式）能够告知我们如何利用新证据修改已有的看法。这个名称来自于托马斯·贝叶斯。

通常，事件A在事件B（发生）的条件下的概率，与事件B在事件A（发生）的条件下的概率是不一样的；然而，这两者是有确定的关系的，贝叶斯定理就是这种关系的陈述。贝叶斯公式的一个用途在于通过已知的三个概率函数推出第四个。

贝叶斯公式为：

P(B|A)= (4.31)

在贝叶斯定理中，每项都有约定俗成的名称：

* P(B|A)是已知A发生后B的条件概率，也由于得自A的取值而被称作B的后验概率。
* P(B)是B的先验概率（或边缘概率）。之所以称为"先验"是因为它不考虑任何A方面的因素。
* P(A|B)是已知B发生后A的条件概率，也称为似然（likelihood），也由于得自B的取值而被称作A的后验概率。
* P(A)是A的先验概率或边缘概率。

其中P(A)的计算，通常使用全概率公式，设,则A=A=,从而P(A)==，对任意,贝叶斯公式又可表示为：

P(|A)= （4.32）

## 4.10信息论

信息论是应用数学的一个分支，主要研究的是对信号所含信息的多少进行量化。它的基本想法是一个不太可能的事件居然发生了，要比一个非常可能的事件发生能提供更多的信息。本节主要介绍度量信息的几种常用指标，如信息量、信息熵、条件熵、互信息、交叉熵等。

#### 1.信息量

1948年克劳德·香农（Claude Shannon）发表的论文“通信的数学理论”是世界上首次将通讯过程建立了数学模型的论文，这篇论文和1949年发表的另一篇论文一起奠定了现代信息论的基础。信息量是信息论中度量信息多少的一个物理量，它从量上反应具有确定概率的事件发生时所传递的信息。香农把信息看作是“一种消除不确定性”的量，而概率正好是表示随机事件发生的可能性大小的一个量，因此，可以用概率来定量地描述信息。

在实际运用中，信息量常用概率的负对数来表示，即，I=-lo。为此，可能有人会问，为何要用对数，前面还要带上负号？

用对数表示是为了计算方便。因为直接用概率表示，在求多条信息总共包含的信息量时，要用乘法，而对数可以变求积为求和。另外，随机事件的概率总是小于1，而真实小于1的对数为负的，概率的对数之前冠以负号，其值便成为正数。所以通过消去不确定性，获取的信息量总是正的。

#### 2.信息熵

信息熵（entropy）又简称为熵，是对随机变量不确定性的度量。熵的概念由鲁道夫·克劳修斯（Rudolf Clausius）于1850年提出，并应用在热力学中。1948年，克劳德·艾尔伍德·香农（Claude Elwood Shannon）第一次将熵的概念引入信息论中，因此它又称为香农熵。

用熵来评价整个随机变量X平均的信息量，而平均最好的量度就是随机变量的期望，即熵的定义如下：

H(X)=- (4.33)

这里假设随机变量X的概率分布为：P(X=)= i=1,2,3,....,n

信息熵越大，包含的信息就越多，那么随机变量的不确定性就越大。

以下我们通过一个实例进一步说明这个关系。

假设随机变量X服从0-1分布，其概率分布为：

P(X=1)=p,P(X=0)=1-p

这时，X的熵为：

H(X)=-p(p)-(1-p)(1-p)

我们利用Python具体实现以下概率p与H(X)的关系：

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#定义概率列表

p=np.arange(0,1.05,0.05)

HX=[]

for i in p:

if i==0 or i==1:

HX.append(0)

else:

HX.append(-i\*np.log2(i)-(1-i)\*np.log2(1-i))

plt.plot(P,HX,label='entropy')

plt.xlabel('p')

plt.ylabel('H(X)')

plt.show()

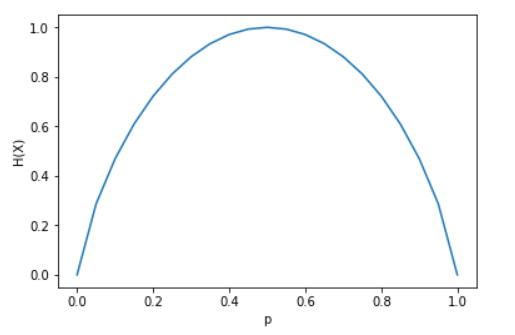


图4-5 概率与信息熵

从这个图形可以看出，当概率为0或1时，H(X)为0，说明此时随机变量没有不确定性，当p=0.5时，随机变量的不确定性最大，即信息量最大。H(X)此时取最大值。

#### 3.条件熵

设二维随机变量（X,Y）,其联合概率分布为：

P(X=,Y=)=, i=1,2,...,n; j=1,2,...,m (4.34)

条件熵H(Y|X)表示在已知随机变量X的条件下，随机变量Y的不确定性，它的计算公式为：

H(Y|X)=- (4.35)

注意，这个条件熵，不是指随机变量X在给定某个数的情况下，另一个变量的熵是多少，变量的不确定性是多少？而是期望！因为条件熵中X也是一个变量，意思是在一个变量X的条件下（变量X的每个值都会取），另一个变量Y熵对X的期望。

条件熵比熵多了一些背景知识，按理说条件熵的不确定性小于熵的不确定，即H(Y|X)，事实也是如此，下面这个定理有力地说明了这一点。

定理：对二维随机变量（X,Y）,条件熵H（Y|X）和信息熵H（Y）满足如下关系：

H(Y|X) (4.36)

#### 4.互信息

互信息（mutual information）又称为信息增益，用来评价一个事件的出现对于另一个事件的出现所贡献的信息量。记为：

I(X,Y)=H(Y)-H(Y|X) (2.37)

在决策树的特征选择中，信息增益为主要依据。在给定训练数据集D,假设数据集由n维特征构成，构建决策树时，一个核心问题就是选择哪个特征来划分数据集，使得划分后的纯度最大，一般而言，信息增益越大，意味着使用某属性a来划分所得“纯度提升”越大。因此，我们常用信息增益来构建决策树划分属性。

#### 5.相对熵

相对熵（relative entropy），所谓相对，一般是在两个随机变量之间来说，又被称为KL散度(Kullback–Leibler divergence，KLD)，这里我们假设 p(x) 和 q(x) 是 X 取值的两个概率分布，如p(x)表示X的真实分布，q(x)表示X的训练分布或预测分布，则 p 对 q 的相对熵为：

KL(p(x)||q(x))= (4.38)

相对熵有些重要性质：

（1）相对熵不是传统意义上的距离，它没有对称性，即

KL(p(x)||q(x))KL(q(x)||p(x))

（2）当预测分布q(x)与真实分布p(x)完全相等时，相对熵为0；

（3）如果两个分别差异越大，那么相对熵也越大；反之，如果两个分布差异越小，相对熵也越小。

（4）相对熵满足非负性，即 KL(p(x)||q(x))

#### 6.交叉熵

交叉熵可在神经网络(机器学习)中作为代价函数，p表示真实标记的分布，q则为训练后的模型的预测标记分布，交叉熵代价函数可以衡量p与q的相似性。交叉熵作为代价函数还有一个好处是使用sigmoid函数在梯度下降时能避免均方误差代价函数学习速率降低的问题，因为学习速率可以被输出的误差所控制。

交叉熵（cross entropy），其定义为：

H(p(x0,q(x))=H(X)+KL(p(x)||q(x)) (4.39)

其中

H(X)=-

KL(p(x)||q(x)) =)

故H(p(x0,q(x))化简后为：

H(p(x0,q(x))=- （4.40）

## 4.11 小结

概率论与信息论是机器学习的重要基础及重要理论依据。本章介绍了概率论、信息论的一些基本概念，如样本空间、随机变量等。根据随机变量取值不同，又可分为有离型和连续型随机变量；根据随机变量的维度又可分为一维或多维随机变量。概率分布、边缘分布是刻画随机变量的重要特征，而期望、方差及协方差是随机变量三个常用统计量。信息论是刻画随机变量的另一种方式，信息论在深度学习、人工智能中应用非常广泛，后续章节也经常会出现。

# 第5章概率图模型

概率图模型（probabilistic graphical model）是一类用图的形式表示随机变量之间条件依赖关系的概率模型,是概率论与图论的结合，图中的节点表示一个或一组随机变量，节点之间的边表示变量间的概率相关关系。根据图中边的有向、无向性，模型可分为两类：有向图、无向图。有向图又称为贝叶斯网，无向图又称为马尔科夫网。

数据流图是深度学习框架TensorFlow、Theano等编程核心，了解概率图模型有利于理解这些深度学习框架，此外，概率图模型中很多内容在深度学习中运用非常广泛，如隐马尔可夫模型（HMM）广泛运用于语音识别、模式识别、故障诊断等方面。条件随机场（CRF）广泛应用于自然语言处理。

以下我们首先简单说明采用概率图的一些优点，然后从大家比较熟悉的监督学习入手引入生成模型及判别模型，在此基础上简单介绍一些图论基础知识，最后根据模型是否有向，把概率图分为贝叶斯网络和马尔可夫网络，并分别对这两种网络进行简单介绍。

## 5.1为何要引入概率图

对于一般的统计推断问题，概率模型能够很好的解决，那么引入概率图模型又能带来什么好处呢？

概率图模型吸取了图论和概率二者的长处，利用图来表示与模型有关的变量的联合概率分布。具体来说：

* 它使概率模型可视化了，这样就使得一些变量之间的关系能够很容易的从图中观测出来；
* 有一些概率上的复杂的计算可以理解为图上的信息传递，这时我们就无须关注太多的复杂表达式了；
* 图模型能够用来设计新的模型。

总之，图的表达能力非常强，仅仅用点和线就可以表达随机变量之间复杂的关系。如果给关联随机变量的边再附加上概率，就可进一步表达随机变量之间关系的强弱和推理逻辑了。

## 5.2使用图描述模型结构

图结构是一种重要的数据结构类型，也是概率图模型的两大支柱之一。图结构的优点很多，其中把复杂的分布关系通过可视化的形式展示出来，无疑是最明显的。概率图模型使用图来表示随机变量间互相关系，在图中，每个节点代表一个随机变量，每一条边代表随机变量间的依赖关系或相互关系。

概率图模型中图的边可能有方向。也可能没方向。如果一个图包含有向的边，那么这个图模型就是有向概率图模型（或称为贝叶斯网络），否则就是无向概率图模型（又称为马尔可夫网）。有向概率图模型（如图5-1）可以表达随机变量间的依赖关系或因果关系，而无向概率图（如图5-2）可以非常直观表达随机变量间的相互关系。



图5-1 有向概率图模型图5-2 无向概率图模型

有向概率图模型使用带有方向边的图，它们用条件概率分布来表示分解。假设有向概率模型中对于分布中的每一个随机变量都含有一个影响因子，这个组成条件概率的影响因子称为的父节点，记为g(,则有：

p(x)= (5.1)

根据（5.1）式，我们不难得到图5-1对应的概率分布可以分解为：

p(A,B,C,D)=p(A)p(B|A)p(C|A)p(D|B,C) (5.2)

其中B、C的父节点都是A,D的父节点为B,C。

无向概率图模型使用带有无向的图，它们不像有向模型那样含有影响因子，对于无向模型来说，由无向边连接的随机变量，它们之间的影响是等价的，不存在一个变量影响或决定另一个变量的概念，它们之间的影响更像一种函数关系。因此，对无向概率图模型通常将其分解为一组函数，这些函数通常不是任何类型的概率分布。无向模型图中任何满足两两之间有边连接的节点集合称为团，若在一个团中加入任何一个节点都不再形成团，则称该团为极大团（maximal clique）。

无向概率图模型中每个团(或极大团)都伴随着一个因子（或称为势函数），记为。这些因子仅仅是函数，并不是概率分布。虽然每个因子的输出都必须为非负，但不要求这些因子的和或积分为1。

随机变量的联合概率与所有这些因子的乘积成比例，虽然不能保证乘积为1，但我们可以使其归一化来得到一个概率分布，除以一个常数Z使其归一化，这个常数被定义为函数乘积的所有状态的求和或积分。则联合概率p(x)定义为：

p(x)= (5.3)

根据式（5.3），我们可以将到图5-2的概率分布分解为：

p(A,B,C,D)=(A,B)(A,C)(B,D)(C,D)/Z (5.4)

由此可以看出，有向图的联合概率可以写成各条件概率的乘积，而无向图的联合概率可以写成团（或极大团）随机变量函数的乘积。

对于概率图模型的分类，我们可以粗略地表示成下图的树形结构：



图5-3 概率图模型

这里我们粗略地将概率图模型分为贝叶斯网络和马尔可夫网络，实际应用中，在很多时候是它们的某种形式的结合。以下我们就贝叶斯网络、马尔可夫网络分别进行简单介绍。

## 5.3贝叶斯网络

贝叶斯网络分为静态贝叶斯网络和动态贝叶斯网络，其中动态贝叶斯网络(dynamic Bayesian networks，DBN)应用非常广泛，可用于处理随时间变化的动态系统中的推断和预测等问题。而隐马尔可夫模型(hidden Markov model，HMM)是动态贝叶斯网络的典型代表，它被广泛应用于语音识别、自动分词与词性标注和统计机器翻译等。限于篇幅考虑，这节我们主要介绍隐马尔可夫模型。

### 5.3.1隐马尔可夫模型简介

首先介绍隐马尔可夫模型的结构，然后介绍利用它可以解决哪些问题，最后通过一个简单实例说明HMM的结构及问题求解。

如图5-4所示，隐马尔可夫模型分为上下两行，上行为马尔可夫转移过程，下行则为输出。

**...**

**...**

图5-4 隐马尔可夫模型

图中， 表示状态转移；表示观察值输出。

如果用变量来表示，可分为两组：一组是状态变量{,},其中Z 表示第i时刻的系统状态，通常状态变量是隐藏的，不可被观察的（不可被观察这个概念，这么说你可能还不清楚，没关系，后面有例子会介绍）。因此状态变量又称为隐变量。另一组变量为观察变量{,}，其中X 表示第i时刻的观察值。在隐马尔可夫模型中，系统通常在N个状态之间转换，所以状态变量的取值范围Z通常有N个可能取值的离散空间{}。观察变量的取值可以是离散的，也可以是连续的。这里我们以离散为例，连续类似，假设其取值范围为{}。

图5-4中箭头，不管是横向的还是纵向的，都是说明变量间一种依赖关系。横向箭头表示t时刻的状态仅依赖于其前一个时刻t-1的状态，与t-1之前的任何状态无关；纵向箭头表示观察值的取值仅依赖于当前的状态变量，即由确定，与其他状态变量或观察值无关。这就是所谓的马尔可夫链。这种变量间的关系，可以用图5-5表示。

图5-5 隐马尔可夫模型变量间的关系

基于这种依赖关系，依赖于(或称为的父节点)，依赖于(或称为)，根据式(5.1),不难得到所有变量的联合概率分布为：

p(,)=p()p(|) (5.5)

### 5.3.2隐马尔可夫模型三要素

隐马尔可夫模型的三要素，即确定它的三组参数，初始状态项链π、状态转移概率矩阵A和观测概率矩阵B。π和A决定状态序列，B决定观测序列。因此A、B和π就是隐马尔可夫模型的三要素，而θ = {π, A, B} 表示控制隐马尔可夫模型参数的集合。

以下我们根据式（5.5），看如何表示隐马尔可夫模型的三组参数：

（1）初始状态项链π

模型在初始时刻各状态出现的概率，记为π={},其中

, 1

（2）状态转移概率矩阵A

模型在各个状态间转换的概率，记为矩阵A=[,其中

, 1

（3）观测概率矩阵B

模型根据当前状态获得各个观测值的概率，记为矩阵B=[,其中

, 1，1

假设隐马尔可夫模型的三要素，即θ= {π,A,B}已知，我们该如何得到观测序列{}呢？一般可以通过以下步骤实现：

1）根据初始状态概率π，获取初始状态；

2）根据状态和输出观测概率矩阵B选择观测变量取值；

3）根据状态和状态转移矩阵A，选择确定

4）若t<n,令t=t+1,并返回第（2）步，否则停止。

以上我们介绍了隐马尔可夫模型的结构、确定它的三要素及根据三要素产生观测序列的一般步骤。但我们还不清楚隐马尔可夫模型是如何解决一些实际问题的，如语言识别、机器翻译、参数学习等。下一节我们介绍这方面的内容。

### 5.3.3隐马尔可夫模型三个基本问题

隐马尔可夫模型在实际应用中非常广泛，它可以解决的问题很多，一般我们可以归结三个基本问题。即评估问题、解码问题、学习问题。

（1）评估问题

给定模型θ= {π,A,B}，如何计算其产生观测序列X=}的概率p(x|θ)?

这个问题在实际应用中非常重要，如许多任务需要根据以往的观测序列}来推测当前时刻最有可能的观测值。这个问题可以转换为求概率p(x|θ)。

（2）解码问题

给定模型θ= {π,A,B}和观测序列X=}，如何找到与之最匹配的隐含状态序列Z={,}？

这个问题可以运用在语音识别中，在语音识别任务中，观测值为语言信号，隐藏状态为文字，目标就是根据观测信号来推断最有可能的隐藏状态序列，即文字。

（3）学习问题

给定观测序列X=}，如何调整模型参数θ= {π,A,B}，使得该序列出现的概率p(x|θ)最大。这个问题就是如何根据训练样本学得最优模型参数。

对这个问题，都有对应方法，如对评估问题可以采用前向算法，对解码问题可以采用维特比（Viterbi）算法，对学习问题可以采用Baum-Welch算法。

看到这里或许你对隐马尔可夫模型还不是很清楚，如观测序列如何产生，隐含状态是不可观测是什么意思等。没关系，接下来我们通过介绍一个具体实例帮助你进一步理解。

### 5.3.4隐马尔可夫模型简单实例

下面我们用一个简单的例子[[1]](#footnote-2)来阐述隐马尔可夫模型的主要内容和核心思想：

假设我们手里有三个不同的骰子。第一个骰子是我们平常见的骰子（称这个骰子为D6），6个面，每个面（1，2，3，4，5，6）出现的概率是1/6。第二个骰子是个四面体（称这个骰子为D4），每个面（1，2，3，4）出现的概率是1/4。第三个骰子有八个面（称这个骰子为D8），每个面（1，2，3，4，5，6，7，8）出现的概率是1/8。这三个骰子具体信息如图5-6。

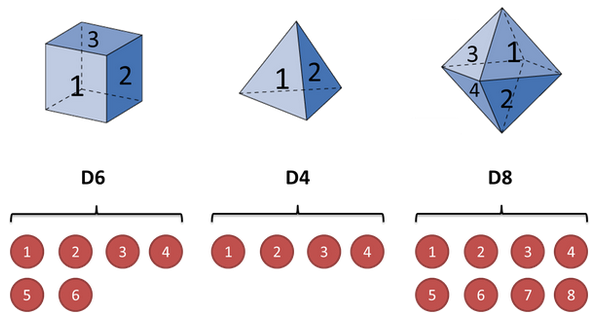


图5-6 三个不同骰子的所含的数字

假设开始掷骰子，我们开始下例步骤：

（1）从三个骰子里挑一个（挑到每一个骰子的概率都是）；

（2）掷骰子，得到一个数字（这个数字为1，2，3，4，5，6，7，8中的一个）。

不停的重复上述过程，我们会得到一串数字，每个数字都是1，2，3，4，5，6，7，8中的一个。例如我们可能得到这么一串数字（假设掷骰子10次）：1 6 3 5 2 7 3 5 2 4

这串数字叫做可见状态链或称为观测序列。但是在隐马尔可夫模型中，我们不仅仅有这么一串可见状态链，还有一串隐含状态链。在这个例子里，这串隐含状态链就是你用的骰子的序列。比如，隐含状态链有可能是：D6 D8 D8 D6 D4 D8 D6 D6 D4 D8

一般来说，HMM中说到的马尔可夫链其实是指隐含状态链，因为隐含状态（骰子）之间存在转换概率（transition probability）。在我们这个例子里，D6的下一个状态是D4、D6、D8的概率都是。D4、D8的下一个状态是D4、D6、D8的转换概率也都一样是。这样设定是为了便于说明，但是我们其实是可以随意设定转换概率的。比如，我们可以这样定义，D6后面不能接D4，D6后面是D6的概率是0.9，是D8的概率是0.1等等。这样就是一个新的HMM。

同样的，尽管可见状态之间没有转换概率，但是隐含状态和可见状态之间有一个概率叫做输出概率（emission probability）。就我们的例子来说，六面骰（D6）产生1的输出概率是。产生2，3，4，5，6的概率也都是。我们同样可以对输出概率进行其他定义。比如，我有一个被赌场动过手脚的六面骰子，掷出来是1的概率更大，是，掷出来是2，3，4，5，6的概率是。

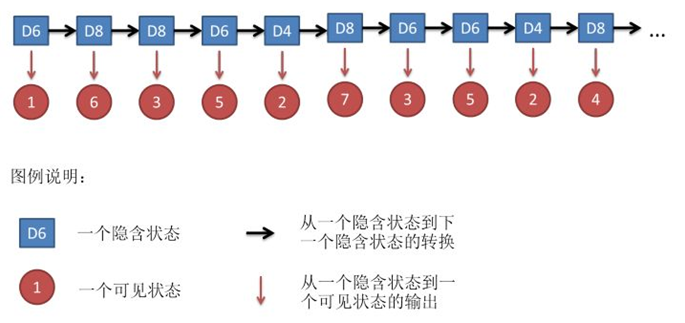


图5-7 HMM示意图

其实对于HMM来说，如果提前知道所有隐含状态之间的转换概率和所有隐含状态到所有可见状态之间的输出概率，做模拟是相当容易的。但是应用HMM模型时候呢，往往是缺失了一部分信息的，有时候你知道骰子有几种，每种骰子是什么，但是不知道掷出来的骰子序列；有时候你只是看到了很多次掷骰子的结果，剩下的什么都不知道。如果应用算法去估计这些缺失的信息，就成了一个很重要的问题，这些问题可归结为我们上面提到的三个基本问题。这三个基本问题落实到这个具体实例就是：

（1）评估问题

知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道掷出这个结果的概率。看似这个问题意义不大，因为你掷出来的结果很多时候都对应了一个比较大的概率。问这个问题的目的呢，其实是检测观察到的结果和已知的模型是否吻合。如果很多次结果都对应了比较小的概率，那么就说明我们已知的模型很有可能是错的，有人偷偷把我们的骰子給换了。

（2）解码问题

知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道每次掷出来的都是哪种骰子（隐含状态链）。

（3）学习问题

知道骰子有几种（隐含状态数量），不知道每种骰子是什么（转换概率），观测到很多次掷骰子的结果（可见状态链），我想反推出每种骰子是什么（转换概率）。这个问题很重要，因为这是最常见的情况。很多时候我们只有可见结果，不知道HMM模型里的参数，我们需要从可见结果估计出这些参数，这是建模的一个必要步骤。

## 5.4马尔可夫网络

概率图模型中，我们根据图的边是否有向，大致分为两类，有向的称为贝叶斯网络，无向的称为马尔可夫网络。马尔可夫网络是关于一组有马尔可夫性质随机变量X的全联合概率分布模型。

马尔可夫网络类似贝叶斯网络用于表示依赖关系。但是，一方面它可以表示贝叶斯网络无法表示的一些依赖关系，如循环依赖；另一方面，它不能表示贝叶斯网络能够表示的某些关系，如推导关系。

在5.2节我们介绍了马尔可夫网络基于团或最大团的因子分子分解，其分解公式为式（5.3），这个因子分解也称为Hammersley–Clifford定理。

Hammersley–Clifford定理：定义在随机变量集x上的马尔可夫网络，其联合概率分布p(x)可表示为式（5.3)。

在贝叶斯网络中，我们重点介绍了隐马尔可夫模型。在马尔可夫网络中我们也将重点介绍一种常见的模型，即马尔可夫随机场，简称为MRF。它是典型的马尔可夫网络，也是著名的一种无向图模型，马尔可夫随机场有一组势函数，也称因子，这是定义在变量子集上的非负函数，其联合概率的表达式就是式（5.3）。

### 5.4.1 马尔可夫随机场

在概率图模型中，一个很重要的任务把概率联合函数进行分解，而函数分解通常利用随机变量的独立性或条件独立性。当然利用概率图，还可以这种独立性可视化。

在马尔可夫随机场中如何得到这种独立性呢？这里我们可借助“分离”的概念。如图5-8所示。如果从节点集X中的节点到Y中的节点都必须经过节点集Z中的节点，则称节点集X和Y被节点集Z分离，其中Z又称为分离集。



图5-8 节点集Z分离节点集X和Y

对马尔可夫随机场，有

* **全局马尔可夫独立性**：

给定两个变量子集的分离集，则这两个变量子集条件独立。

如在无向图5-8中，集合X=(X1,X2,X3)和集合Y=(Y1，Y2，Y3)被集合Z=(Z1,Z2)所分离，则X和Y在给定Z的条件下独立，记为：X|Z,用概率表示为：

p(X,Y|Z)=p(X|Z)p(Y|Z) (5.6)

由全局马尔可夫独立性，可以推广到两个局部的独立性质：

* **局部马尔可夫独立性：**

如图5-9 ，是无向图中的一个节点，W={}是与相连的所有节点，O是,W外的所有节点,O={}。则有：

P(,O|W)=P(|W)P(O|W) (5.7)

图5-9 局部马尔可夫独立性图5-10 成对马尔可夫独立性

* **成对马尔可夫独立性：**

,是无向图中任意两个不相邻的结点，其他所有结点是O={}，则有

P(|O)=P(|O)P(|O) (5.8)

实际上，成对马尔科夫性、局部马尔可夫性、全局马尔可夫性是等价的。

### 5.4.2条件随机场

条件随机场（Conditional random field，CRF）是条件概率分布模型 P(Y|X) ，表示的是给定一组输入随机变量 X 的条件下另一组输出随机变量 Y 的马尔可夫随机场，也就是说 CRF 的特点是假设输出随机变量构成马尔可夫随机场。这时，在条件概率模型P(Y|X)中，Y是输出变量，表示标记序列，X是输入变量，表示需要标注的观测序列。也把标记序列称为状态序列（参见隐马尔可夫模型)。学习时，利用训练数据集通过极大似然估计或正则化的极大似然估计得到条件概率模型(Y|X)；预测时，对于给定的输入序列X求出条件概率(Y|X)最大的输出序列Y。

* **条件随机场的定义：**

设X与Y是随机变量，P(Y |X)是在给定X的条件下Y的条件概率分布。若随机变量Y构成一个由无向图G=(V,E)表示的马尔可夫随机场，即

(5.9)

对任意节点*v*成立，则称条件概率分布P(Y|X)为条件随机场。

式中*w~v*表示在图G=(V,E)中与节点*v*有边连接的所有节点*w，w!=v*表示节点*v*以外的所有节点，Y*v*，Yw为节点*v，w*对应的随机变量。

一般假设X和Y有相同的图结构。线性链条件随机场的情况为：

G=(V={1,2,},E={(i,i+1)}),i=1,2,,n-1

在此情况下，X=(),Y=(),最大团是相邻两个节点的集合。如图5-11所示。

X={}

图5-11 线性链条件随机场

* **线性链条件随机场**

设X=(),Y=(),为线性链表示的随机变量序列，若在给定随机变量序列X的条件下，随机变量序列Y的条件概率分布P(Y |X)构成条件随机场。即满足马尔可夫性

p()=p(),i=1,2, （5.10）

其中 i=1 或n 时，只考虑单边。则称P(Y |X)为线性链条件随机场。

### 5.4.3实例：用Tensorflow实现条件随机场

条件随机场在词性标注中发挥重要作用，首先我们简单介绍何为词性标注，然后通过一个实例来实现一个具体的词性标注。

为了更好的理解词性标注这个概念，我们还是通过示例来说明。

词性标注就是给一个句子中的每个单词注明词性。比如这句话：“He drank coffee at Starbucks”，注明每个单词的词性后是这样的：“He (代词) drank(动词) coffee(名词) at(介词) Starbucks(名词)”。我们用图5-12 表示词性标注过程：

Y = {}

He drank coffee at Starbucks

代词 动词 名词 介词 名词

X = {}

图5-12 词性标注

如何用条件随机场来解决词性标注问题？

以上面的这句话为例，这句话为它共有5个单词，我们将：(代词，动词，名词，介词，名词)作为一个标注序列，称为Y，可选的标注序列有很多种，比如Y还可以是这样：（名词，动词，动词，介词，名词），我们要在这么多的可选标注序列中，挑选出一个最靠谱的作为我们对这句话的标注。

怎么判断一个标注序列靠谱不靠谱呢？

就我们上面展示的两个标注序列来说，第二个显然不如第一个靠谱，因为它把第二、第三个单词都标注成了动词，动词后面接动词，这在一个句子中通常是说不通的。

假如我们给每一个标注序列打分，打分越高代表这个标注序列越靠谱，我们至少可以说，凡是标注中出现了动词后面还是动词的标注序列，要给它负分。

上面所说的动词后面还是动词就是一个特征函数，我们可以定义一个特征函数集合，用这个特征函数集合来为一个标注序列打分，并据此选出最靠谱的标注序列。也就是说，每一个特征函数都可以用来为一个标注序列评分，把集合中所有特征函数对同一个标注序列的评分综合起来，就是这个标注序列最终的评分值。

定义CRF中的特征函数。现在，我们正式地定义一下什么是CRF中的特征函数，所谓特征函数，就是这样的函数，它接受四个参数：

句子X（就是我们要标注词性的句子）

i，用来表示句子X中第i个单词

，表示要评分的标注序列给第i个单词标注的词性

，表示要评分的标注序列给第i-1个单词标注的词性

它的输出值是0或者1,0表示要评分的标注序列不符合这个特征，1表示要评分的标注序列符合这个特征。这里，我们的特征函数仅仅依靠当前单词的标签和它前面的单词的标签对标注序列进行评判，这样建立的CRF也叫作线性链CRF，这是CRF中的一种简单情况。

定义好一组特征函数后，我们要给每个特征函数赋予一个权重。现在，只要有一个句子X，有一个标注序列Y，我们就可以利用前面定义的特征函数集来对l评分。

score（Y|X）= (5.11)

上式中有两个求和，外面的求和用来求每一个特征函数评分值的和，里面的求和用来求句子中每个位置的单词的的特征值的和。

对这个分数进行指数化和标准化，我们就可以得到标注序列l的概率值p(Y|X)，如下所示：

P(Y|X)= (5.12)

CRF它不像LSM等模型，能够考虑长远的上下文信息，它更多考虑的是整个句子的局部特征的线性加权组合（通过特征模版去扫描整个句子）。关键的一点是，CRF的模型为p(Y | X)，注意这里Y和X都是序列，优化的是一个序列Y= (,, …, )，而不是某个时刻的(，即找到一个概率最高的序列Y= (,, …, )使得p(Y| X)最高，它计算的是一种联合概率，优化的是整个序列（最终目标），而不是将每个时刻的最优拼接起来，在这一点上CRF要优于LSTM。

以下是用Tensorflow来具体实现一个CRF,在代码前，先说明几个重要函数的功能：

1）tf.contrib.crf.crf\_log\_likelihood

在一个条件随机场里面计算标签序列的log-likelihood，其格式为：

crf\_log\_likelihood(inputs,tag\_indices,sequence\_lengths,transition\_params=None)

2）tf.contrib.crf.viterbi\_decode

其作用就是返回最好的标签序列。这个函数只能够在测试时使用,在tensorflow外部解码。其格式为：

viterbi\_decode(score,transition\_params)

3）tf.contrib.crf.crf\_decode

在tensorflow内解码，其格式为：

crf\_decode(potentials,transition\_params,sequence\_length)

import numpy as np

import tensorflow as tf

# 参数设置

num\_examples = 10

num\_words = 20

num\_features = 100

num\_tags = 5

# 构建随机特征

x = np.random.rand(num\_examples, num\_words, num\_features).astype(np.float32)

# 构建随机tag

y = np.random.randint(

num\_tags, size=[num\_examples, num\_words]).astype(np.int32)

# 获取样本句长向量（因为每一个样本可能包含不一样多的词），在这里统一设为 num\_words - 1，真实情况下根据需要设置

sequence\_lengths = np.full(num\_examples, num\_words - 1, dtype=np.int32)

# 训练，评估模型

with tf.Graph().as\_default():

with tf.Session() as session:

x\_t = tf.constant(x)

y\_t = tf.constant(y)

sequence\_lengths\_t = tf.constant(sequence\_lengths)

# 在这里设置一个无偏置的线性层

weights = tf.get\_variable("weights", [num\_features, num\_tags])

matricized\_x\_t = tf.reshape(x\_t, [-1, num\_features])

matricized\_unary\_scores = tf.matmul(matricized\_x\_t, weights)

unary\_scores = tf.reshape(matricized\_unary\_scores,

[num\_examples, num\_words, num\_tags])

# 计算log-likelihood并获得transition\_params

log\_likelihood, transition\_params = tf.contrib.crf.crf\_log\_likelihood(

unary\_scores, y\_t, sequence\_lengths\_t)

# 进行解码（维特比算法），获得解码之后的序列viterbi\_sequence和分数viterbi\_score

viterbi\_sequence, viterbi\_score = tf.contrib.crf.crf\_decode(

unary\_scores, transition\_params, sequence\_lengths\_t)

loss = tf.reduce\_mean(-log\_likelihood)

train\_op = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

session.run(tf.global\_variables\_initializer())

mask = (np.expand\_dims(np.arange(num\_words), axis=0) < # np.arange()创建等差数组

np.expand\_dims(sequence\_lengths, axis=1)) # np.expand\_dims()扩张维度

# 得到一个num\_examples\*num\_words的二维数组，数据类型为布尔型，目的是对句长进行截断

# 将每个样本的sequence\_lengths加起来，得到标签的总数

total\_labels = np.sum(sequence\_lengths)

# 进行训练

for i in range(1000):

tf\_viterbi\_sequence, \_ = session.run([viterbi\_sequence, train\_op])

if i % 100 == 0:

correct\_labels = np.sum((y == tf\_viterbi\_sequence) \* mask)

accuracy = 100.0 \* correct\_labels / float(total\_labels)

print("Accuracy: %.2f%%" % accuracy)

运行结果：

Accuracy: 20.53%

Accuracy: 56.84%

Accuracy: 67.37%

Accuracy: 73.68%

Accuracy: 80.00%

Accuracy: 82.63%

Accuracy: 84.21%

Accuracy: 88.42%

Accuracy: 88.42%

Accuracy: 90.00%

## 5.5小结

概率图模型可以直观理解为概率模型的图形化或可视化，概率模型的图形化给我们带来很大的方便，一些复杂逻辑表达式用概率图模型来描述就变得非常直观和简洁，所以概率图模型是深度学习重要工具，我们在很多深度学习参考资料上都会看到大量的概率图，概率图模型这种思想也影响到很多深度学习框架，如Theano、TensorFlow等。

本章介绍了概念图模型的基本概念及基本结构，根据构造图是否有向，我们把概率图模型分为贝叶斯网络和马尔可夫网络。在贝叶斯网络中重点介绍了隐马尔可夫模型，在马尔可夫网络中主要介绍了马尔可夫随机场、马尔可夫独立性、条件随机场等内容。

机器学习、深度学习基础部分就介绍完了，接下来开始介绍机器学习、深度学习的理论和应用了。

1. 这个例子来源：知乎，

   作者:Yang Eninala,链接：https://www.zhihu.com/question/20962240/answer/33438846 [↑](#footnote-ref-2)