最大熵模型简介[例子+推导+GIS求解]

Posted on 2015年3月25日 by fuqingchuan

这篇文章是结合论文http://www.cqvip.com/Main/Detail.aspx?id=7707219对博文：http://www.cnblogs.com/hexinuaa/p/3353479.html加入自己的理解做了简化重写，另外本文末尾附上了最大熵模型的实现。

一个例子

我们通过一个简单的例子来了解最大熵的概念。假设现在需要做一个自动将英语到法语的翻译模型，为了方便说明，我们将这个问题简化为将英文句子中的单词{in}翻译成法语词汇。那么翻译模型p就是对于给定包含单词”in”的英文句子，需要给出选择某个法语单词f 做为”in”的翻译结果的概率p(f)。为了帮助开发这个模型，需要收集大量已经翻译好的样本数据。收集好样本之后，接下来需要做两件事情：一是从样本中抽取规则（特征），二是基于这些规则建立模型。

从样本中我们能得到的第一个规则就是in可能被翻译成的法语词汇有：

{dans, en, à, au cours de, pendant}。

也就是说，我们可以给模型p施加第一个约束条件：

p(dans)+p(en)+ p(à)+p(au cours de)+p(pendant) = 1。

这个等式是翻译模型可以用到的第一个对样本的统计信息。显然，有无数可以满足上面约束的模型p可供选择，例如：

p(dans)=1，即这个模型总是预测dans

或者

p(pendant)=1/2 and p(à)=1/2，即模型要么选择预测pendant，要么预测à。

这两个模型都只是在没有足够经验数据的情况下，做的大单假设。事实上我们只知道当前可能的选项是5个法语词汇，没法确定究竟哪个概率分布式正确。那么，一个更合理的模型假设可能是：

p(dans) = 1/5

p(en) = 1/5

p(à) = 1/5

p(au cours de) = 1/5

p(pendant) = 1/5

即该模型将概率均等地分给5个词汇。但现实情况下，肯定不会这么简单，所以我们尝试收集更多的经验知识。假设我们从语料中发现有30%的情况下，in会被翻译成dans 或者en，那么运用这个知识来更新我们的模型，得到2模型约束：

p(dans) + p(en) = 3/10

p(dans)+p(en)+ p(à)+p(au cours de)+p(pendant) = 1

同样，还是有很多概率分布满足这两个约束。在没有其他知识的情况下，最直观的模型p应该是最均匀的模型（例如，我拿出一个色子问你丢出5的概率是多少，你肯定会回答1/6），也就是在满足约束条件的情况下，将概率均等分配：

p(dans) = 3/20

p(en) = 3/20

p(à) = 7/30

p(au cours de) = 7/30

p(pendant) = 7/30

假设我们再一次观察样本数据，发现：有一般的情况，in被翻译成了dans 或 à。这样，我们有就了3个模型约束：

p(dans) + p(en) = 3/10

p(dans)+p(en)+ p(à)+p(au cours de)+p(pendant) = 1

p(dans)+ p(à)=1/2

我们可以再一次选择满足3个约束的最均匀的模型p，但这一次结果没有那么明显。由于经验知识的增加，问题的复杂度也增加了，归结起来，我们要解决两组问题：第一，均匀(uniform)究竟是什么意思?我们怎样度量一个模型的均匀度(uniformity)？第二，有了上述两个问题的答案，我们如何找到满足所有约束并且均匀的模型？

最大熵算法可以回答上面的2组问题。直观上来将，很简单，即：对已知的知识建模，对未知的知识不做任何假设。换句话说，在给定一组事实（features + output）的情况下，选择符合所有事实，且在其他方面请可能均匀的模型。这也是我们在上面的例子中，每次选择最恰当的模型用到的原理。俗话说，不把鸡蛋放在一个篮子里，正是运用的这个原理来规避风险。

最大熵(MaxEnt)建模

我们考察一个随机过程，它的输出是y，　y属于有穷集合Y。对于上面提到的例子，该过程输出”in”的翻译结果y，　y∈Y = {dans, en, à, au cours de, pendant}。在输出y时，该过程会受到”in”在句子中上下文信息x的影响，x属于有穷集合X。在上文的例子中，这个x主要就是在英文句子中”in”周围的单词。

我们的任务就是构造一个统计模型，该模型的任务是：在给定上下午x的情况下，输出y的概率p(y|x)。

训练数据

为了研究上述过程，我们观察一段时间随机过程的行为，收集到大量的样本数据：（x_1, y_1），(x_2, y_2)，…，(x_N, y_N)。在之前讨论的例子中，每个样本包括：在英文句子中”in”周围的单词x，”in”的翻译y。假设我们已经拿到了足够多的训练样本，我们可以用样本的经验分布$\tilde{p}$来表示所有样本的分布特性：

$$\tilde{p}(x,y) = \frac{1}{N}\, num(x,y)$$

其中N为训练样本的大小，num(x,y)是样本中某一对(x,y)同时出现的次数。

特征和约束

我们的目标是构造一个能产生训练样本这一随机过程$\tilde{p}$(x,y)的统计模型。而我们能够使用的数据就是对训练样本的各种统计信息或者说特征。定义特征如下：

$$f(x,y)=\begin{cases}1, & \text{if } x=x_0 \; and \; y=y_0\\ 0, & \text{otherwise}\end{cases}$$

这个也叫指示函数(indicator function)，它表示某个特定的x和某个特定的y之间是否有一定的关系。例如，在之前的例子中，如果April这个词出现在in之后，那么in会被翻译成en，那么这个特征可以表示成：

$$f(x,y)=\begin{cases}1, & \text{if } y=en \; and \; \text{April follows in}\\ 0, & \text{otherwise}\end{cases}$$

特征f(x,y)关于训练样本经验分布$\tilde{p}$(x,y)的期望如下，这个是我们可以在语料中统计到的特征经验值：

$$\tilde{p}(f)=\sum_{x,y}\tilde{p}(x,y)f(x,y)\text{——————————}(1)$$

而特征关于模型分布p(y|x)的理论期望值是：

$$p(f)=\sum_{x,y}\tilde{p}(x)p(y|x)f(x,y)\text{——————————}(2)$$

其中$\tilde{p}$(x)是x在训练样本中的经验分布。我们约束这一期望值和训练样本中的经验值相等：即要求期望概率值等于经验概率值。

$$p(f)=\tilde{p}(f)\text{——————————————}(3)$$

结合等式(1)(2)(3)我们得到等式：

$$\sum_{x,y}\tilde{p}(x)p(y|x)f(x,y)=\sum_{x,y}\tilde{p}(x,y)f(x,y)\text{——————————}(4)$$

我们称等式(3)为约束(constraint)。我们只关注满足约束(3)的模型p(y|x)，也就是说不再考察跟训练样本的特征经验值不一致的模型。

到这里，我们已经有办法来表示训练样本中内在的统计现象（$\tilde{p}$(f)），同时也有办法来让模型拟合这一现象(p(f) = $\tilde{p}$(f))。

最大熵原理

再回到之前例子中的问题：什么是均匀?

数学上，条件分布p(y|x)的均匀度就是条件熵，定义如下：

$$H(p)=-\sum_{x,y}\tilde{p}(x)p(y|x)\log p(y|x)\text{——————————}(5)$$

熵的最小值是0，这时模型没有任何不确定性；熵的最大值是log|Y|，即在所有可能的y(|Y|个)上的均匀分布。

有了这个条件熵，最大熵的原理定义为：当从允许的概率分布集合C中选择一个模型时，选择模型p*∈C ，使得熵H(p)最大。即：

$$C = \{p \in P | p(f_i) = \tilde{p}(f_i) \;\; for \;\; i \in \{1, 2, ..., n\}\} \text{———}(6)$$

$$p^* = \arg\max_{p \in C} H(p) \text{————}(7)$$

其中C的含义是所有满足约束的模型集合，n为特征或者说特征函数fi的数量（注意跟样本数量N区别）。

指数形式

最大熵要解决的是约束优化问题：find the p*∈C which maximizes H(p)。对于上述翻译的例子，如果只施加了前面两个约束，很容易直接求得p的分布。但是，绝大多数情况下最大熵模模型的解无法直接给出，我们需要引入像拉格朗日乘子（Lagrange Multiplier）这样的方法。

我们具体要优化的问题是：

$$p^* = \arg\max_{p \in C} H(p)$$
$$= \arg\max_{p \in C} (-\sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x)) \text{————}(8)$$

相应的约束为：

$$1. p(y|x) >= 0, \text{ for all x,y}$$

$$2. \sum_y p(y|x) = 1, \text{ for all x}$$

$$3. \sum_{x,y} \tilde{p}(x) p(y|x) f(x,y) = \sum_{x,y} \tilde{p}(x,y) f(x,y), \text{for } i \in \{1, 2, ..., n\}$$

为解决这个优化问题，为每一个f_i引入参数λ_i ，得到拉格朗日函数ξ(p, Λ, γ)如下：

$$\xi(p, \Lambda, \gamma) = -\sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x)$$
$$= +\sum_{i=1}^{n} \lambda_i (\sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) f_i(x,y))$$
$$= +\gamma (\sum_y p(y|x) - 1) \text{————}(9)$$

其中实值参数Λ = {λ1, λ2, …, λn}和γ对应着约束2和3的n+1个限制。保持Λ和γ不变，计算拉格朗日函数ξ(p, Λ, γ)在不受限的情况下的最大值，即可得到p(y|x)的最优解。因此对

ζ(p, Λ, γ)在p(y|x)上求导得（注意为了方便计算，我们这里对数以自然对数e为底）：

$$\frac{\partial \xi}{\partial p(y|x)} = -\tilde{p}(x)(\log p(y|x) + 1) + \sum_{i=1}^{n} \lambda_i \, \tilde{p}(x) f_i(x, y) + \gamma \text{ ———— } (10)$$

令上式（10）等于0，即：

$$-\tilde{p}(x)(\log p(y|x) + 1) + \sum_{i=1}^{n} \lambda_i \, \tilde{p}(x) f_i(x, y) + \gamma = 0 \text{ ———— } (11)$$

解上面的单变量方程(把p(y|x)当未知数)，可得p(y|x)：

$$p(y|x) = \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \exp(\frac{\gamma}{\tilde{p}(x)} - 1) \text{ ———— } (12)$$

根据约束2，对于任意x，Σ_yp(y|x) = 1，我们对等式(12)两边分别求和，可以得到：

$$\sum_y p(y|x) = \sum_y \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \exp(\frac{\gamma}{\tilde{p}(x)} - 1)$$

$$\Rightarrow 1 = \sum_y \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \exp(\frac{\gamma}{\tilde{p}(x)} - 1)$$

$$\Rightarrow \exp(\frac{\gamma}{\tilde{p}(x)} - 1) = \frac{1}{\sum_y \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y))} \text{ ———— } (13)$$

将等式(13)代入等式（12），可以得到：

$$p(y|x) = \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \frac{1}{\sum_y \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y))} \text{ ———— } (14)$$

令Z(x) 为：

$$Z(x) = \sum_y \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \text{ ———— } (15)$$

则模型p(y|x)的最优解为，其中Z(x)称为归一化因子：

$$p^*(y|x) = \frac{1}{Z(x)} \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \text{ ———— } (16)$$

到这里，我们可以看到，满足约束C的最大熵模型具有（16）的参数化形式。由于拉格朗日函数ζ(p, Λ, γ)中的p(y|x)和γ都可以表示成Λ，所以接下来的问题就转化成了求解参数Λ = {λ1, λ2, …, λn}。为此，我们定义对偶函数ψ(Λ) 及其优化问题：

$$\Psi(\Lambda) = \xi(p^*, \Lambda, \gamma^*) \text{ ———————— } (17)$$

$$\text{Find } \Lambda^* = \arg\min_{\Lambda} \Psi(\Lambda) \text{ ———————— } (18)$$

根据所谓的Kuhn-Tucker theorem(KTT)原理（这个目前还没研究，暂不展开），我们有如下结论：公式（16）中模型p*(y|x)中的参数Λ，可以通过最小化对偶函数ψ(Λ) 求解，如(17)(18)所示。

计算参数

求解Λ = {λ1, λ2, …, λn}，解析的方法是行不通的。这种最大熵模型对应的最优化问题，一般有GIS/IIS，IIS是GIS的优化版。这里讨论最简单的GIS。GIS的一般步骤是：

1. 初始化所有λi 为任意值，一般可以设置为0，即：

$$\lambda_i^{(0)} = 0, i \in \{1, 2, 3, ..., n\}$$

其中λ的上标(t)表示第t论迭代，下标i表示第i个特征，n是特征总数。

2. 重复下面的权值更新直至收敛：

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \frac{1}{C} \log \frac{E_{\tilde{p}} f_i}{E_{p^{(n)}} f_i}, i \in \{1, 2, ..., n\} \text{ ———— } (19)$$

收敛的判断依据可以是λi 前后两次的差价足够小。其中C一般取所有样本数据中最大的特征数量，$E_{\tilde{p}}$和Ep如下:

$$E_{\tilde{p}} f_i = \sum_{x,y} \tilde{p}(x, y) f_i(x, y)$$

$$= \frac{1}{N} \sum_{j=1}^{N} f_i(x_j, y_j) \text{ ———————— } (20)$$

$$E_{p^{(n)}} f_i = \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x, y)$$

$$\cong \frac{1}{N} \sum_{j=1}^{N} \sum_{y} p^{(n)}(y|x_j) f_i(x_j, y) \text{ ———— } (21)$$

$$p^{(n)}(y|x) = \frac{1}{Z(x)} \exp(\sum_{i=1}^{n} \lambda_i f_i(x, y)) \text{ ———— } (22)$$

为了是代码简短，方便阅读，去掉了很多健壮性检测的代码以及特殊处理。下面的代码实现的是：使用最基础GIS训练最大熵模型。GIS由于性能问题在实际中不适用，但是可以帮助我们理解最大熵训练到底在做什么。

MaxEnt 的Python实现 by fuqingchuan

```
#!/usr/bin/python
```

```
#coding=utf8
import sys;
import math;
from collections import defaultdict

class MaxEnt:
def __init__(self):
self._samples = []; #样本集, 元素是[y,x1,x2,...,xn]的元组
self._Y = set([]); #标签集合,相当于去重之后的y
self._numXY = defaultdict(int);    #Key是(xi,yi)对，Value是count(xi,yi)
self._N = 0;#样本数量
self._n = 0;#特征对(xi,yi)总数量
self._xyID = {};#对(x,y)对做的顺序编号(ID), Key是(xi,yi)对,Value是ID
self._C = 0;#样本最大的特征数量,用于求参数时的迭代，见IIS原理说明
self._ep_ = [];#样本分布的特征期望值
self._ep = [];#模型分布的特征期望值
self._w = [];#对应n个特征的权值
self._lastw = [];#上一轮迭代的权值
self._EPS = 0.01;#判断是否收敛的阈值
def load_data(self, filename):
for line in open(filename, "r"):
sample = line.strip().split("\t");
if len(sample) < 2: #至少：标签+一个特征
continue;
y = sample[0];
X = sample[1:];
self._samples.append(sample); #labe + features
self._Y.add(y); #label
for x in set(X): #set给X去重
self._numXY[(x, y)] += 1;
```

```
def _initparams(self):
self._N = len(self._samples);
self._n = len(self._numXY);
self._C = max([len(sample) - 1 for sample in self._samples]);
self._w = [0.0] * self._n;
self._lastw = self._w[:];
self._sample_ep();
def _convergence(self):
for w, lw in zip(self._w, self._lastw):
if math.fabs(w - lw) >= self._EPS:
return False;
return True;
def _sample_ep(self):
self._ep_ = [0.0] * self._n;
#计算方法参见公式(20)
for i, xy in enumerate(self._numXY):
self._ep_[i] = self._numXY[xy] * 1.0 / self._N;
self._xyID[xy] = i;
def _zx(self, X):
#calculate Z(X)，计算方法参见公式(15)
ZX = 0.0;
for y in self._Y:
sum = 0.0;
for x in X:
if (x, y) in self._numXY:
sum += self._w[self._xyID[(x, y)]];
ZX += math.exp(sum);
return ZX;
def _pyx(self, X):
#calculate p(y|x)，计算方法参见公式(22)
```

```
ZX = self._zx(X);
results = [];
for y in self._Y:
sum = 0.0;
for x in X:
if (x, y) in self._numXY: #这个判断相当于指示函数的作用
sum += self._w[self._xyID[(x, y)]];
pyx = 1.0 / ZX * math.exp(sum);
results.append((y, pyx));
return results;
def _model_ep(self):
self._ep = [0.0] * self._n;
#参见公式(21)
for sample in self._samples:
X = sample[1:];
pyx = self._pyx(X);
for y, p in pyx:
for x in X:
if (x, y) in self._numXY:
self._ep[self._xyID[(x, y)]] += p * 1.0 / self._N;
def train(self, maxiter = 1000):
self._initparams();
for i in range(0, maxiter):
print "Iter:%d..."%i;
self._lastw = self._w[:]; #保存上一轮权值
self._model_ep();
#更新每个特征的权值
for i, w in enumerate(self._w):
#参考公式(19)
self._w[i] += 1.0 / self._C * math.log(self._ep_[i] / self._ep[i]);
```

```
print self._w;
#检查是否收敛
if self._convergence():
break;
def predict(self, input):
X = input.strip().split("\t");
prob = self._pyx(X)
return prob;

if __name__ == "__main__":
maxent = MaxEnt();
maxent.load_data('data.txt');
maxent.train();
print maxent.predict("sunny\thot\thigh\tFALSE");
print maxent.predict("overcast\thot\thigh\tFALSE");
print maxent.predict("sunny\tcool\thigh\tTRUE");
sys.exit(0);
```

训练数据来自各种天气情况下是否打球的例子：data.txt

其中字段依次是:

play outlook temperature humidity windy

部分运行结果：

```
71928268592, -1.2645221442480512, 1.3179713774210247, 1.3912022828649
Iter:273...
[2.1527896880880952, -2.6065160738693214, -3.0646845197820611, 2.75252
 -6.6375186004047677, -0.8333632914063607 8, -0.0952842618813497 11, 0.1
90802324, -1.2665758264458553, 1.3200420698923396, 1.3929989388592232,
Iter:274...
[2.1554798878521 08, -2.6100611147670869, -3.0692453017677539, 2.757057
28, -6.6476770277300776, -0.8348546807617 4207, -0.0950930819806331 12,
9040945894, -1.2686240834784444, 1.3221072470751107, 1.394790490242561
Iter:275...
[2.1581638857327206, -2.6135962244840023, -3.0737943138948682, 2.76157
134, -6.6578072850176433, -0.8363411557373835 5, -0.0949030738900313 31,
240021866144, -1.2706669460688358, 1.3241669404420342, 1.3965769690509
Iter:276...
[2.1608417150360353, -2.6171214645275134, -3.0783316227948547, 2.76606
552, -6.6679095400503776, -0.8378227491664365 1, -0.0947142285009480 31,
419872657004, -1.2727044446685232, 1.3262211811857085, 1.3983584070022
Iter:277...
[2.1635134087411996, -2.6206368958016415, -3.0828572944910917, 2.77055
569, -6.6779839590634511, -0.8392994935655933, -0.0945265367860320 53,
29028387938, -1.2747366094606956, 1.3282700002219789, 1.4001348355006 8
Iter:278...
[2.1661789995052967, -2.6241425786150172, -3.0873713944065613, 2.77502
67, -6.6880307067633789, -0.8407714211389413 2, -0.0943399897984560 23,
67920151711, -1.2767634703634096, 1.3303134281932312, 1.4019062856406 4
Iter:279...
[2.1688385196681441, -2.6276385726887788, -3.0918739873714016, 2.77947
231, -6.6980499463468153, -0.8422385637817604 3, -0.0941545786711987 66,
536975111465, -1.2787850570327159, 1.332351495471638, 1.4036727882112 0
Iter:280...
[2.1714920012569943, -2.631124937164341, -3.0963651376303387, 2.783913
09, -6.7080418395190602, -0.8437009530842665 1, -0.0939702946163310 65,
36616547121, -1.2808013988657383, 1.3343842321623569, 1.4054343737000 2
[('yes', 0.0041626518719793002), ('no', 0.99583734812802072)]
[('yes', 0.99436821023604471), ('no', 0.0056317897639553702)]
[('yes', 1.4464465173635744e-07), ('no', 0.99999985535534819)]
[root@iZ25ttq9nypZ maxent-python]#
```