《机器学习》课程系列

最大熵模型*

武汉纺织大学数学与计算机学院

杜小勤

2020/04/18

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 21, 2020。

1 最大熵原理

最大熵原理是选取概率模型的一般性准则——在所有可能的概率模型中，首先依据约束条件确定概率模型的集合，然后选取熵最大的模型。

假设离散随机变量 X 的概率分布是 $P(X)$，熵 (Entropy) 有以下几种等价表示方法：

$$\begin{aligned} H(P) &= -\sum_{x} P(x)\log P(x) \\ H(X) &= -\sum_{i=1}^{n} P(x_i)\log P(x_i) \\ H(X) &= -\sum_{i=1}^{n} p_i \log p_i \end{aligned} \tag{1}$$

其中，$p_i = P(x_i)$，n 表示随机变量 X 的取值个数 $|X|$。熵，又称为信息熵，用来衡量随机变量 X 取值的不确定性，它满足下面的不等式：

$$0 \leqslant H(X) \leqslant \log|X| \tag{2}$$

在 X 服从均匀分布时，上式取得最大值，即取得最大熵。直观地看，此时，随机变量 X 具有最好的不确定性——等可能性。

结合到最大熵原理，该原理认为，在满足约束条件的所有可能的概率模型中，如果已经没有更多的信息用于选取概率模型，那么一定要选取熵最大的模型，即最大熵模型。原因在于，最大熵模型为将来的进一步决策提供了最好的不确定性或等可能性。从某种角度来看，最大熵原理符合奥卡姆剃刀原理。该原理提倡“如无必要、勿增实体”，遵循简单即有效的原理，也体现了“化繁为简”的思想理念。最大熵模型的选取原则，既简单，又为将来的各种可能保持了最好的可能性，也是有效的。

下面，通过一个具体实例来阐述最大熵原理与求解的基本方法。假设随机变量 X 有 5 个取值 $\{A, B, C, D, E\}$，且 $P(A) + P(B) = \frac{3}{10}$，试估计各个取值的概率：$P(A), \ldots, P(E)$。

首先，写出随机变量 X 的熵。令 $x_i (i = 1, 2 \ldots, 5)$ 分别取值 A、B、C、D、E：

$$H(P) = -\sum_{i=1}^{5} P(x_i)\log P(x_i) \tag{3}$$

定义最优化问题：

$$\begin{aligned}
&\max_{P\in C} H(P)=\max_{P\in C}-\sum_{i=1}^{5} P\left(x_{i}\right)\log P\left(x_{i}\right)\\
&s.t.\quad P(x_1)+P(x_2)=\frac{3}{10}\\
&\qquad\sum_{i=1}^{5} P\left(x_{i}\right)=1
\end{aligned}\tag{4}$$

其中，C 表示所有满足约束条件的模型集合，上式的最后 2 行表示约束条件。按照惯例，将该最大化问题转换为等价的最小化问题：

$$\begin{aligned}
&\min_{P\in C} -H(P)=\min_{P\in C}\sum_{i=1}^{5} P\left(x_{i}\right)\log P\left(x_{i}\right)\\
&s.t.\quad P(x_1)+P(x_2)=\frac{3}{10}\\
&\qquad\sum_{i=1}^{5} P\left(x_{i}\right)=1
\end{aligned}\tag{5}$$

这是一个带约束的最优化问题，可以使用拉格朗日乘数法来求解。定义拉格朗日优化函数：

$$L(P,\boldsymbol{w})=\sum_{i=1}^{5} P\left(x_{i}\right)\log P\left(x_{i}\right)+w_0\left(\sum_{i=1}^{5} P\left(x_{i}\right)-1\right)+w_1\left(P\left(x_1\right)+P\left(x_2\right)-\frac{3}{10}\right)\tag{6}$$

由于拉格朗日函数 $L(P,\boldsymbol{w})$ 是 P 的凸函数，根据拉格朗日对偶性，原始问题的解与对偶问题的解等价，即可以通过求解对偶最优化问题而得到原始最优化问题的解。于是，求解下面的对偶最优化问题：

$$\max_{\boldsymbol{w}}\min_{P} L(P,\boldsymbol{w})\tag{7}$$

先求解 $\min\limits_{P} L(P,\boldsymbol{w})$(固定 w_0 和 w_1)：

$$\left\{\begin{aligned}
\frac{\partial L(P,\boldsymbol{w})}{\partial P\left(x_1\right)}&=1+\log P\left(x_1\right)+w_0+w_1\\
\frac{\partial L(P,\boldsymbol{w})}{\partial P\left(x_2\right)}&=1+\log P\left(x_2\right)+w_0+w_1\\
\frac{\partial L(P,\boldsymbol{w})}{\partial P\left(x_3\right)}&=1+\log P\left(x_3\right)+w_0\\
\frac{\partial L(P,\boldsymbol{w})}{\partial P\left(x_4\right)}&=1+\log P\left(x_4\right)+w_0\\
\frac{\partial L(P,\boldsymbol{w})}{\partial P\left(x_5\right)}&=1+\log P\left(x_5\right)+w_0
\end{aligned}\right.\tag{8}$$

得到：

$$\begin{cases}P\left(x_1\right)=P\left(x_2\right)=\mathrm{e}^{-w_0-w_1-1} \\ P\left(x_3\right)=P\left(x_4\right)=P\left(x_5\right)=\mathrm{e}^{-w_0-1}\end{cases} \tag{9}$$

将上述结果代入 $L(P, \boldsymbol{w})$ 中，得到：

$$\min_P L(P, \boldsymbol{w})=L\left(P_{\boldsymbol{w}}, \boldsymbol{w}\right)=-2 e^{-w_0-w_1-1}-3 e^{-w_0-1}-w_0-\frac{3}{10} w_1 \tag{10}$$

继续，求解 $\max\limits_{\boldsymbol{w}} L\left(P_{\boldsymbol{w}}, \boldsymbol{w}\right)$：

$$\max_{\boldsymbol{w}} L\left(P_{\boldsymbol{w}}, \boldsymbol{w}\right)=-2 e^{-w_0-w_1-1}-3 e^{-w_0-1}-w_0-\frac{3}{10} w_1 \tag{11}$$

对 w_i 求偏导并令其等于 0，可得：

$$\begin{cases}\mathrm{e}^{-\mathrm{w}_0-\mathrm{w}_1-1}=\frac{3}{20} \\ \mathrm{e}^{-\mathrm{w}_0-1}=\frac{7}{30}\end{cases} \tag{12}$$

最后得到：

$$\begin{cases}P\left(x_1\right)=P\left(x_2\right)=\frac{3}{20} \\ P\left(x_3\right)=P\left(x_4\right)=P\left(x_5\right)=\frac{7}{30}\end{cases} \tag{13}$$

上述结果符合我们的直觉：在缺少更多信息的情况下，根据已知条件，直接可以推断 X 取值 A 与 B 具有等可能性，而取值 C、D 与 E 也是等可能的，这是最简单而有效的确定模型的方法。因此，根据奥卡姆剃刀原理，上面求解得到的模型就是“最好”的模型，而最大熵原理给出了最优模型选择的一个准则。

2 最大熵模型

给定训练数据集：

$$T=\left\{\left(x_1, y_1\right),\left(x_2, y_2\right), \cdots,\left(x_N, y_N\right)\right\} \tag{14}$$

对于分类问题而言，学习的目标是，依据最大熵原理选取最好的分类模型。

对于此类问题，与上一节中例子的总体思路类似：首先需要确定最优化的目标，然后需要确定模型应该满足的约束条件，最后依据拉格朗日乘数法求解最大熵分类模型。当然，最大熵分类模型的求解要复杂许多，但基本原理与方法是类似的。

2.1 条件熵

首先给出条件熵 (Conditional Entropy) 的定义。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：

$$\begin{aligned} H(Y|X) &= \sum_x P(x)H(Y|X=x) \\ &= -\sum_x P(x)\sum_y P(y|x)\log P(y|x) \\ &= -\sum_x\sum_y P(x,y)\log P(y|x) \\ &= -\sum_{x,y} P(x,y)\log P(y|x) \end{aligned} \tag{15}$$

从上式可以看出，条件熵 $H(Y|X)$ 表示为给定 X 条件下 Y 的条件熵 $H(Y|X=x)$ 对 X 的数学期望。

另一种等价性定义，可表示为条件自信息 $I(y|x)$ 关于 X 与 Y 联合分布的数学期望：

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{P(x,y)}\left[I(y|x)\right] \\ &= -\sum_x\sum_y P(x,y)\log P(y|x) \\ &= \sum_x P(x)\left[-\sum_y P(y|x)\log P(y|x)\right] \\ &= \sum_x P(x)H(Y|x) \end{aligned} \tag{16}$$

其中，$H(Y|x)=H(Y|X=x)$ 表示 X 取某个特定值 x 时 Y 的条件熵。

2.2 模型的定义

设分类模型为条件概率分布 $P(Y|X)$，为应用最大熵原理，需要以条件熵最大化作为优化目标，同时，也需要考虑模型应该满足的约束条件。

根据问题性质，需要定义一些特征函数 (Feature Function)$f(x,y)$，用来描述样本点中某个特征 x 与输出 y 之间存在的某种关系[1]：

$$f(x,y)=\begin{cases} 1, & if\ (x,y)\ is\ in\ our\ facts. \\ 0, & else \end{cases} \tag{17}$$

[1]特征函数不限于二值函数，可以是任意实值函数。对于分类问题而言，y 具有离散值，用来表示类别。

根据贝叶斯定理，对于数据集的真实分布，下式成立：

$$P(X,Y)=P(X)P(Y|X) \tag{18}$$

在实际应用中，真实分布往往是未知的。然而，我们可以利用给定的数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$ 和最大似然估计对经验联合分布与经验边缘分布进行估计[2]：

$$\begin{aligned}
&\tilde{P}(X,Y)\Rightarrow\tilde{P}(X=x,Y=y)=\frac{\nu(X=x,Y=y)}{\mathrm{N}}\\
&\tilde{P}(X)\Rightarrow\tilde{P}(X=x)=\frac{\nu(X=x)}{\mathrm{N}}
\end{aligned} \tag{19}$$

其中，$\nu(X=x,Y=y)$ 表示训练集中特征 (x,y) 出现的次数，$\nu(X=x)$ 表示训练集中特征 x 出现的次数，N 表示相应特征类别出现的总次数。

由此，可以定义模型的约束条件为[3]：

$$E_{\tilde{P}}(f)=E_P(f)\Longleftrightarrow\sum_{x,y}\tilde{P}(x,y)f(x,y)=\sum_{x,y}\tilde{P}(x)P(y|x)f(x,y) \tag{20}$$

其中，$\sum\limits_{x,y}\tilde{P}(x,y)f(x,y)$ 表示特征函数 $f(x,y)$ 关于经验分布 $\tilde{P}(X,Y)$ 的期望值，使用 $E_{\tilde{P}}(f)$ 表示；$\sum\limits_{x,y}\tilde{P}(x)P(y|x)f(x,y)$ 表示特征函数 $f(x,y)$ 关于模型 $P(Y|X)$ 与经验分布 $\tilde{P}(X)$ 的期望值，使用 $E_P(f)$ 表示。公式 (20) 就是模型 $P(Y|X)$(基于数据集 T) 必须满足的约束条件。

该约束条件的意义在于，如果模型能够学习到训练集中的有用信息，那么这 2 个期望值应该是相等的，即：

$$E_{\tilde{P}}(f)=E_P(f) \tag{21}$$

一般地，在实际应用问题中，存在着多个特征函数 $f_i(x,y)(i=1,2,\ldots,n)$，则模型需要基于数据集 T 显式地考虑所有这些约束条件。

下面给出最大熵模型的定义。假设满足所有约束条件的模型集合为：

$$\mathcal{C}\equiv\{P\in\mathcal{P}|E_{\tilde{P}}(f_i)=E_P(f_i),\quad i=1,2,\cdots,n\} \tag{22}$$

[2]而贝叶斯定理三要素之一的条件分布 $P(Y|X)$ 正是最大熵模型的学习与优化目标，需要学习获得。

[3]需要注意的是，除非特别说明，否则，在本章求和或求积公式中出现的符号“x,y”，应被理解为枚举所有的全局特征对，而不是枚举所有的样本点对 $(\boldsymbol{x},y)$。

那么，定义在条件概率分布 $P(Y|X)$ 上的条件熵为：

$$H(P)=-\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x) \tag{23}$$

则模型集合 C 中条件熵 $H(P)$ 最大的模型称为最大熵模型。其中，log 表示自然对数；x 与 y 来自于数据集 T。

2.3 模型的推导

对于给定的数据集 T 和特征函数 $f_i(x,y)$，最大熵模型的学习等价于如下最大化问题：

$$\begin{aligned}
&\max_{P\in C} H(P)=-\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x)\\
&\text{s.t.}\quad E_{\tilde{P}}\left(f_i\right)=E_P\left(f_i\right)\quad i=1,2,\cdots,n\\
&\qquad \sum_y P(y|x)=1
\end{aligned} \tag{24}$$

按照最优化惯例，将其转换为等价的最小化问题：

$$\begin{aligned}
&\min_{P\in C} -H(P)=\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x)\\
&\text{s.t.}\quad E_{\tilde{P}}\left(f_i\right)=E_P\left(f_i\right)\quad i=1,2,\cdots,n\\
&\qquad \sum_y P(y|x)=1
\end{aligned} \tag{25}$$

为了方便求解，将约束最优化的原始问题转换为无约束最优化的对偶问题，然后通过求解对偶问题来获得原始问题的解。

首先，定义拉格朗日优化函数:

$$\begin{aligned}
L(P,\boldsymbol{w})\equiv&-H(P)+w_0\left(1-\sum_y P(y|x)\right)+\sum_{i=1}^n w_i\left(E_{\tilde{p}}\left(f_i\right)-E_P\left(f_i\right)\right)\\
=&\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x)+w_0\left(1-\sum_y P(y|x)\right)+\\
&\sum_{i=1}^n w_i\left(\sum_{x,y}\tilde{P}(x,y)f_i(x,y)-\sum_{x,y}\tilde{P}(x)P(y|x)f_i(x,y)\right)
\end{aligned} \tag{26}$$

拉格朗日原始最优化问题为：

$$\min_{P\in C}\max_{\boldsymbol{w}} L(P,\boldsymbol{w}) \tag{27}$$

其对偶问题为：

$$\max_{\boldsymbol{w}} \min_{P \in C} L(P, \boldsymbol{w}) \tag{28}$$

由于拉格朗日函数 $L(P, \boldsymbol{w})$ 是 P 的凸函数，根据拉格朗日对偶性，原始问题的解与对偶问题的解等价。将原始问题转换为对偶问题：

$$\min_{P \in \mathbf{C}} \max_{\boldsymbol{w}} L(P, \boldsymbol{w}) \quad \Rightarrow \quad \max_{\boldsymbol{w}} \min_{P \in \mathbf{C}} L(P, \boldsymbol{w}) \tag{29}$$

即通过求解对偶最优化问题而得到原始最优化问题的解。

于是，先求解最小化问题 $\min\limits_{P \in \mathbf{C}} L(P, \boldsymbol{w})$，它是 $\boldsymbol{w}$ 的函数，定义对偶函数：

$$\begin{aligned} \Psi(\boldsymbol{w}) &= \min_{P \in C} L(P, \boldsymbol{w}) = L\left(P_{\boldsymbol{w}}, \boldsymbol{w}\right) \\ P_{\boldsymbol{w}} &= \arg\min_{P \in \mathbf{C}} L(P, \boldsymbol{w}) = P_{\boldsymbol{w}}(y|x) \end{aligned} \tag{30}$$

其中，$P_{\boldsymbol{w}}$ 是最小化问题 $\min\limits_{P \in \mathbf{C}} L(P, \boldsymbol{w})$ 的解。为求解 $P_{\boldsymbol{w}}$，对 $L(P, \boldsymbol{w})$ 关于 $P(y|x)$ 求偏导数：

$$\frac{\partial L(P, \boldsymbol{w})}{\partial P(y|x)} = \tilde{P}(x)(\log P(y|x) + 1) - w_0 - \tilde{P}(x) \sum_{i=1}^{n} w_i f_i(x, y) \tag{31}$$

令偏导数等于 0，解得：

$$P(y|x) = \exp\left(\sum_{i=1}^{n} w_i f_i(x, y) + \frac{w_0}{\tilde{P}(x)} - 1\right) = \frac{\exp\left(\sum\limits_{i=1}^{n} w_i f_i(x, y)\right)}{\exp\left(1 - \frac{w_0}{\tilde{P}(x)}\right)} \tag{32}$$

上式两端同时对 y 求和，并利用 $\sum\limits_{y} P(y|x) = 1$，可得：

$$P_{\boldsymbol{w}}(y|x) = \frac{1}{Z_{\boldsymbol{w}}(x)} \exp\left(\sum_{i=1}^{n} w_i f_i(x, y)\right) \tag{33}$$

其中：

$$Z_{\boldsymbol{w}}(x) = \sum_{y} \exp\left(\sum_{i=1}^{n} w_i f_i(x, y)\right) \tag{34}$$

$Z_{\boldsymbol{w}}(x)$ 被称为规范化因子。其向量形式为：

$$P_{\boldsymbol{w}}(y|x) = \frac{\exp(\boldsymbol{w}^T F(x, y))}{\sum\limits_{y} \exp(\boldsymbol{w}^T F(x, y))} \tag{35}$$

其中，向量 $\boldsymbol{w}$ 的元素为 w_i，向量 $F(x, y)$ 的元素为 $f_i(x, y)$，$i = 1, 2, \ldots, n$。

最后，求解对偶问题外部的最大化问题：

$$\max_{\boldsymbol{w}} \Psi(\boldsymbol{w}) \tag{36}$$

即对公式 (33) 进行最优化求解，得到解 $\boldsymbol{w}^*$：

$$\begin{aligned} \boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}} \Psi(\boldsymbol{w}) \\ P^* &= P_{\boldsymbol{w}^*} = P_{\boldsymbol{w}^*}(y|x) \end{aligned} \tag{37}$$

由此得到最大熵模型 $P_{\boldsymbol{w}^*}(y|x)$。

从最大熵模型 (公式 (33))，可以看出，它是由特征函数 $f_i(x,y)$ 及其权重 w_i 表示的条件概率分布。公式 (33) 似曾相识！是的，如果简单地令 $f_i(x,y) = x_i$，那么最大熵模型就会退化为逻辑回归模型——对于二分类，表现为 Sigmoid 函数；对于多分类，表现为 Softmax 函数[4]。实际上，后文将正式地表明，由最大熵模型可以推导出逻辑回归模型。

下面将表明，对偶函数 $\Psi(\boldsymbol{w})$ 等价于如下形式的最大熵模型的对数似然函数：

$$L_{\tilde{P}}(P_{\boldsymbol{w}}) = \log \prod_{x,y} P_{\boldsymbol{w}}(y|x)^{\tilde{P}(x,y)} = \sum_{x,y} \tilde{P}(x,y) \log P_{\boldsymbol{w}}(y|x) \tag{38}$$

将最大熵模型 $P_{\boldsymbol{w}}(y|x)$(公式 (33)) 代入上式，可得：

$$\begin{aligned} L_{\tilde{P}}(P_{\boldsymbol{w}}) &= \sum_{x,y} \tilde{P}(x,y) \log P_{\boldsymbol{w}}(y|x) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n} w_i f_i(x,y) - \sum_{x,y} \tilde{P}(x,y) \log Z_{\boldsymbol{w}}(x) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n} w_i f_i(x,y) - \sum_{x} \tilde{P}(x) \log Z_{\boldsymbol{w}}(x) \end{aligned} \tag{39}$$

[4]Sigmoid 函数是 Softmax 函数的特例。

将最大熵模型 $P_{\boldsymbol{w}}(y|x)$(公式 (33)) 代入公式 (26)，可得[5]：

$$\begin{aligned}\Psi(\boldsymbol{w}) = &\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)\log P_{\boldsymbol{w}}(y|x)+\\&\sum_{i=1}^{n}w_i\left(\sum_{x,y}\tilde{P}(x,y)f_i(x,y)-\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)f_i(x,y)\right)\\=&\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_if_i(x,y)+\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)\left(\log P_{\boldsymbol{w}}(y|x)-\sum_{i=1}^{n}w_if_i(x,y)\right)\\=&\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_if_i(x,y)+\\&\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)\left(\log\left(\exp\left(\sum_{i=1}^{n}w_if_i(x,y)\right)\right)-\log Z_{\boldsymbol{w}}(x)-\sum_{i=1}^{n}w_if_i(x,y)\right)\\=&\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_if_i(x,y)-\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)\log Z_{\boldsymbol{w}}(x)\end{aligned}\tag{40}$$

上式最后一行，利用 $\sum\limits_{y}P_{\boldsymbol{w}}(y|x)=1$，可得：

$$\Psi(\boldsymbol{w})=\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_if_i(x,y)-\sum_{x}\tilde{P}(x)\log Z_{\boldsymbol{w}}(x)\tag{41}$$

于是，可以得出结论：

$$\begin{aligned}&\Psi(\boldsymbol{w})=L_{\tilde{P}}(P_{\boldsymbol{w}})\\&\max_{\boldsymbol{w}}\Psi(\boldsymbol{w})=\max_{\boldsymbol{w}}L_{\tilde{P}}(P_{\boldsymbol{w}})\end{aligned}\tag{42}$$

这表明，在最大熵模型中，对偶函数的极大化等价于对数似然函数的极大化。

需要注意的是，最大熵模型并不局限于二分类，它可以很自然地支持 K 分类：与逻辑回归情形一样，对每一个类别，定义一个权值向量，即定义 K 个权值向量[6]。

2.4 模型的优化

与逻辑回归模型一样，最大熵模型的优化函数是光滑的凸函数，具有很好的性质，有多种迭代最优化方法可以使用，且保证能够找到全局最优解[7]。

[5]将 $P_{\boldsymbol{w}}(y|x)$ 代入公式 (33) 后，乘数 w_0 所表示的约束 $1-\sum\limits_{y}P_{\boldsymbol{w}}(y|x)$ 已经得到满足，该项不起作用，结果为 0。原因在于，$P_{\boldsymbol{w}}(y|x)$ 已经被正确地归一化为条件概率。

[6]由于冗余性的存在，也可以只定义 $K-1$ 个权值向量。

[7]注意，缺省情况下，无论是逻辑回归模型，还是最大熵模型，其最佳的优化方法都可以理解为最大似然估计。从损失函数的角度来看，它们使用负对数似然作为损失函数。请阅读《逻辑回归》

对于最大熵模型 (公式 (33))，其权值参数 $\boldsymbol{w}$ 的求解，需要针对 $L_{\tilde{P}}(P_{\boldsymbol{w}})$(公式 (38)) 或 $\Psi(\boldsymbol{w})$(公式 (41)) 使用迭代优化方法进行求解[8]。常见的方法有梯度下降法、牛顿法、拟牛顿法和改进的迭代尺度法 (Improved Iterative Scaling, IIS) 等。

2.4.1 梯度下降法

下面介绍梯度下降法，需要推导对数似然函数 $L_{\tilde{P}}(P_{\boldsymbol{w}})$(公式 (38)) 关于权值参数 $\boldsymbol{w}$ 的偏导数。为了便于描述，需要重新表达相关记号与公式。将该对数似然函数表示为：

$$L_{\tilde{P}}(\boldsymbol{W}) = \log \prod_{i=1}^{N} \prod_{k=1}^{K} P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)^{\tilde{P}(\boldsymbol{x}_i, y_i^{(k)})} = \sum_{i=1}^{N} \sum_{k=1}^{K} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) \log P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i) \tag{43}$$

其中，N 表示数据集中样本点的个数；K 表示类别个数；$\boldsymbol{w}_k$ 表示类别 k 对应的权值向量，每个类别有一个权值向量，它们形成一个权值矩阵 $\boldsymbol{W}$；与多类别逻辑回归中的情形一样，类别 y_i 使用 One-Hot 向量表示方式，$y_i^{(k)}$ 表示第 k 个类别的取值 (0 或 1)，并且 $\sum\limits_{k=1}^{K} y_i^{(k)} = 1$；$P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)$ 表示样本点 $\boldsymbol{x}_i$ 划分为第 k 个类别的条件概率。

于是，最大熵模型的损失函数可以使用如下形式的负对数似然函数来表示：

$$L(\boldsymbol{W}) = -\frac{1}{N} L_{\tilde{P}}(\boldsymbol{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) \log P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i) \tag{44}$$

下面将推导损失函数 $L(\boldsymbol{W})$ 关于权值参数 $\boldsymbol{w}$ 的偏导数。为简便起见，使用向量记法表示条件概率 $P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)$：

$$P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i) = \frac{\exp\left(\sum\limits_{l=1}^{n} w_{kl} f_l(\boldsymbol{x}_i, y_i^{(k)})\right)}{\sum\limits_{j=1}^{K} \exp\left(\sum\limits_{l=1}^{n} w_{jl} f_l(\boldsymbol{x}_i, y_i^{(j)})\right)} = \frac{e^{\boldsymbol{w}_k^T F(\boldsymbol{x}_i, y_i^{(k)})}}{\sum\limits_{j=1}^{K} e^{\boldsymbol{w}_j^T F(\boldsymbol{x}_i, y_i^{(j)})}} \tag{45}$$

其中，向量 $\boldsymbol{w}_k$ 的每个分量为 $\boldsymbol{w}_{kl}$，向量 $F(\boldsymbol{x}_i, y_i^{(k)})$ 的每个分量为 $f_l(\boldsymbol{x}_i, y_i^{(k)})$。

在推导过程中，需要用到 $P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)$ 关于权值参数 $\boldsymbol{w}_t$ 的偏导数 $\frac{\partial P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)}{\boldsymbol{w}_t}$：

与《最大熵模型》课程系列的相关章节。在此前提下，它们都是能够得到全局最优解的凸优化问题。如果使用其它形式的损失函数或求解方法 (例如最小二乘法，即平方误差损失)，那么最优化问题不再是凸优化问题，易陷入局部最优。当然，一般情况下，不会使用这些方法。

[8]前面已经证明，两种方法是等价的。可以看出，最大熵模型 (公式 (33)) 与多分类逻辑回归模型中的 Softmax 函数在形式上非常相似。后文将表明，尽管两者的 (基于数据集的) 对数似然函数有差别，但是梯度求解的过程是一样的，结果也很相似。

- 如果 $t=k$，那么：

$$\begin{aligned}\frac{\partial P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)}{\partial \boldsymbol{w_t}} &= \frac{e^{\boldsymbol{w_t^T} F(\boldsymbol{x}_i, y_i^{(t)})}}{\sum\limits_{j=1}^{K} e^{\boldsymbol{w_j^T} F(\boldsymbol{x}_i, y_i^{(j)})}} F(\boldsymbol{x}_i, y_i^{(t)}) - \left(\frac{e^{\boldsymbol{w_t^T} F(\boldsymbol{x}_i, y_i^{(t)})}}{\sum\limits_{j=1}^{K} e^{\boldsymbol{w_j^T} F(\boldsymbol{x}_i, y_i^{(j)})}}\right)^2 F(\boldsymbol{x}_i, y_i^{(t)}) \\ &= P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\left(1 - P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\right) F(\boldsymbol{x}_i, y_i^{(t)})\end{aligned} \tag{46}$$

- 如果 $t \neq k$，那么：

$$\begin{aligned}\frac{\partial P_{\boldsymbol{w}_k}(y_i^{(k)}|\boldsymbol{x}_i)}{\partial \boldsymbol{w_t}} &= -\frac{e^{\boldsymbol{w_k^T} F(\boldsymbol{x}_i, y_i^{(k)})} e^{\boldsymbol{w_t^T} F(\boldsymbol{x}_i, y_i^{(t)})}}{\left(\sum\limits_{j=1}^{K} e^{\boldsymbol{w_j^T} F(\boldsymbol{x}_i, y_i^{(j)})}\right)^2} F(\boldsymbol{x}_i, y_i^{(t)}) \\ &= -P_{\boldsymbol{w_k}}(y_i^{(k)}|\boldsymbol{x}_i) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i) F(\boldsymbol{x}_i, y_i^{(t)})\end{aligned} \tag{47}$$

可以看出，上述推导结果在形式上与 Softmax 的偏导数非常相似[9]。

针对最大熵模型的损失函数 (公式 (44))，求解偏导数：

$$\begin{aligned}&\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{w_t}} \\ &= -\frac{1}{N}\sum_{i=1}^{N}\left[\tilde{P}(\boldsymbol{x}_i, y_i^{(t)}) \frac{P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\left(1 - P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\right) F(\boldsymbol{x}_i, y_i^{(t)})}{P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)} - \right. \\ &\qquad \left. \sum_{k\neq t} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) \frac{P_{\boldsymbol{w_k}}(y_i^{(k)}|\boldsymbol{x}_i) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i) F(\boldsymbol{x}_i, y_i^{(t)})}{P_{\boldsymbol{w_k}}(y_i^{(k)}|\boldsymbol{x}_i)}\right] \\ &= -\frac{1}{N}\sum_{i=1}^{N}\left[\tilde{P}(\boldsymbol{x}_i, y_i^{(t)})\left(1 - P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\right) F(\boldsymbol{x}_i, y_i^{(t)}) - \right. \\ &\qquad \left. \sum_{k\neq t} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i) F(\boldsymbol{x}_i, y_i^{(t)})\right] \\ &= -\frac{1}{N}\sum_{i=1}^{N}\left[\tilde{P}(\boldsymbol{x}_i, y_i^{(t)}) - \tilde{P}(\boldsymbol{x}_i, y_i^{(t)}) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i) - \sum_{k\neq t}\tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\right] F(\boldsymbol{x}_i, y_i^{(t)}) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\left[\tilde{P}(\boldsymbol{x}_i, y_i^{(t)}) - \tilde{P}(\boldsymbol{x}_i) P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)\right] F(\boldsymbol{x}_i, y_i^{(t)})\end{aligned} \tag{48}$$

[9] 请参考《机器学习》课程系列之《逻辑回归》部分。

其中，上式中最后一行的推导，利用了公式：

$$\tilde{P}(\boldsymbol{x}_i, y_i^{(t)}) + \sum_{k\neq t} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) = \sum_{k=1}^{K} \tilde{P}(\boldsymbol{x}_i, y_i^{(k)}) = \tilde{P}(\boldsymbol{x}_i)$$

$\tilde{P}(\boldsymbol{x}_i, y_i^{(t)})$ 表示样本点对 $(\boldsymbol{x}_i, y_i^{(t)})$ 的经验联合分布 (理想值)，$P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)$ 表示模型对样本点 $\boldsymbol{x}_i$ 预测为类别 $y_i^{(t)}$ 的条件概率，$\tilde{P}(\boldsymbol{x}_i)P_{\boldsymbol{w_t}}(y_i^{(t)}|\boldsymbol{x}_i)$ 表示样本点对 $(\boldsymbol{x}_i, y_i^{(t)})$ 的联合分布 (预测值)，$F(\boldsymbol{x}_i, y_i^{(t)})$ 表示样本点对 $(\boldsymbol{x}_i, y_i^{(t)})$ 的特征向量 (输入)。可以看出，公式所表达的意义非常明确。

对权值参数 $\boldsymbol{w}_t$ 应用负梯度 (公式 (48) 取反) 更新，反复迭代，直至收敛，得到最优权值 $\boldsymbol{w}_t^*$。该方法就是最大熵模型的梯度下降求解方法。

2.4.2 改进的迭代尺度法

下面介绍改进的迭代尺度法 (Improved Iterative Scaling, IIS)。为明确优化任务，将优化问题重新列出：

$$\begin{aligned}
&P_{\boldsymbol{w}}(y|x) = \frac{1}{Z_{\boldsymbol{w}}(x)} \exp\left(\sum_{i=1}^{n} w_i f_i(x,y)\right) \\
&Z_{\boldsymbol{w}}(x) = \sum_{y} \exp\left(\sum_{i=1}^{n} w_i f_i(x,y)\right) \\
&L(\boldsymbol{w}) = \Psi(\boldsymbol{w}) = L_{\tilde{P}}(P_{\boldsymbol{w}}) = \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n} w_i f_i(x,y) - \sum_{x} \tilde{P}(x) \log Z_{\boldsymbol{w}}(x)
\end{aligned} \tag{49}$$

其中，$P_{\boldsymbol{w}}(y|x)$ 和 $Z_{\boldsymbol{w}}(x)$ 构成最大熵模型，$L(\boldsymbol{w})$ 是该模型的对数似然函数，它来自于公式 (39或40)。现在，求解的目标是，使用最大似然估计求解最优参数 $\boldsymbol{w}^*$。

前一节中介绍的梯度下降法，直接对损失函数或负对数似然函数求解各权值参数的梯度，并应用负梯度对权值参数进行更新。本节从对数似然函数的增量角度，给出另一种优化方法。在迭代优化的过程中，在当前时刻 t，最大熵模型的权值向量为 $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^T$；在下一时刻 $t+1$，新的权值向量为：

$$\boldsymbol{w} + \boldsymbol{\delta} = (w_1 + \delta_1, w_2 + \delta_2, \ldots, w_n + \delta_n)^T$$

在理想的情况下，权值向量的改变，应该要使得模型的对数似然函数值增大。直

接利用公式 (38)，其变化量为：

$$
\begin{aligned}
L(\boldsymbol{w}+\boldsymbol{\delta})-L(\boldsymbol{w}) &= \sum_{x,y}\tilde{P}(x,y)\log P_{\boldsymbol{w}+\boldsymbol{\delta}}(y|x)-\sum_{x,y}\tilde{P}(x,y)\log P_{\boldsymbol{w}}(y|x) \\
&= \sum_{x,y}\tilde{P}(x,y)\left[\sum_{i=1}^{n}(w_i+\delta_i)\,f_i(x,y)-\log Z_{\boldsymbol{w}+\boldsymbol{\delta}}(x)\right]- \\
&\quad \sum_{x,y}\tilde{P}(x,y)\left[\sum_{i=1}^{n}w_i f_i(x,y)-\log Z_{\boldsymbol{w}}(x)\right] \\
&= \sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)-\sum_{x,y}\tilde{P}(x,y)\log\frac{Z_{\boldsymbol{w}+\boldsymbol{\delta}}(x)}{Z_{\boldsymbol{w}}(x)} \\
&= \sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)-\sum_{x}\tilde{P}(x)\log\frac{Z_{\boldsymbol{w}+\boldsymbol{\delta}}(x)}{Z_{\boldsymbol{w}}(x)}
\end{aligned}
\tag{50}
$$

利用不等式 $-\log x \geqslant 1-x$(其中 $x>0$)[10]，得到上述增量的下界：

$$
\begin{aligned}
L(\boldsymbol{w}+\boldsymbol{\delta})-L(\boldsymbol{w}) &\geqslant \sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)+1-\sum_{x}\tilde{P}(x)\frac{Z_{\boldsymbol{w}+\boldsymbol{\delta}}(x)}{Z_{\boldsymbol{w}}(x)} \\
&= \sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)+1-\sum_{x}\tilde{P}(x)\sum_{y}P_{\boldsymbol{w}}(y|x)\exp\sum_{i=1}^{n}\delta_i f_i(x,y)
\end{aligned}
\tag{51}
$$

其中：

$$
\begin{aligned}
\frac{Z_{\boldsymbol{w}+\boldsymbol{\delta}}(x)}{Z_{\boldsymbol{w}}(x)} &= \frac{\sum\limits_{y}\exp\left(\sum\limits_{i=1}^{n}(w_i+\delta_i)f_i(x,y)\right)}{\sum\limits_{y}\exp\left(\sum\limits_{i=1}^{n}w_i f_i(x,y)\right)} \\
&= \frac{\sum\limits_{y}\left[\exp\left(\sum\limits_{i=1}^{n}w_i f_i(x,y)\right)\exp\left(\sum\limits_{i=1}^{n}\delta_i f_i(x,y)\right)\right]}{\sum\limits_{y}\exp\left(\sum\limits_{i=1}^{n}w_i f_i(x,y)\right)} \\
&= \sum_{y}\left[\frac{\exp\left(\sum\limits_{i=1}^{n}w_i f_i(x,y)\right)}{\sum\limits_{y}\exp\left(\sum\limits_{i=1}^{n}w_i f_i(x,y)\right)}\exp\left(\sum_{i=1}^{n}\delta_i f_i(x,y)\right)\right] \\
&= \sum_{y}P_{\boldsymbol{w}}(y|x)\exp\sum_{i=1}^{n}\delta_i f_i(x,y)
\end{aligned}
\tag{52}
$$

[10]令 $F(x)=\log x-x+1$，求一阶导数，并令其等于 0：$F'(x)=\frac{1}{x}-1=0$，解得 $x=1$；二阶导数 $F''(x)=-\frac{1}{x^2}<0$。因此，$x=1$ 为极大值，则 $F(x)\leqslant\log 1-1+1=0$，则 $-\log x\geqslant 1-x$ 成立。

令：

$$A(\boldsymbol{\delta}|\boldsymbol{w})=\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)+1-\sum_{x}\tilde{P}(x)\sum_{y}P_{\boldsymbol{w}}(y|x)\exp\sum_{i=1}^{n}\delta_i f_i(x,y) \tag{53}$$

于是：

$$L(\boldsymbol{w}+\boldsymbol{\delta})-L(\boldsymbol{w})\geqslant A(\boldsymbol{\delta}|\boldsymbol{w}) \tag{54}$$

其中，$A(\boldsymbol{\delta}|\boldsymbol{w})$ 是对数似然函数增量的一个下界。可以看出，下界 $A(\boldsymbol{\delta}|\boldsymbol{w})$ 是关于 $\boldsymbol{\delta}$ 的函数，它以当前时刻 t 的权值参数 $\boldsymbol{w}$ 为基础，即权值参数 $\boldsymbol{w}$ 是相对固定的。

如果在当前时刻 t 能够找到适当的 $\boldsymbol{\delta}$，使下界 $A(\boldsymbol{\delta}|\boldsymbol{w})>0$，这样就能够不断地提高对数似然函数的值，即使得 $L(\boldsymbol{w}+\boldsymbol{\delta})>L(\boldsymbol{w})$ 成立[11]。在理想情况下，最好能够让 $A(\boldsymbol{\delta}|\boldsymbol{w})$ 最大化，即取得最好的增量。在这种情况下，$L(\boldsymbol{w}+\boldsymbol{\delta})$ 将以最大幅度增长。

此时，最优化目标变成了函数 $A(\boldsymbol{\delta}|\boldsymbol{w})$，即求解该函数中的最优 $\boldsymbol{\delta}^*$。但是，直接求解 $\boldsymbol{\delta}$，还是稍显复杂，并且由于 $\boldsymbol{\delta}$ 是一个向量，不易同时优化多个分量。于是，考虑进一步降低下界函数 $A(\boldsymbol{\delta}|\boldsymbol{w})$，且算法每次只优化一个分量。具体地，为进一步降低下界，IIS 算法引进计数函数 $f^{\#}(\boldsymbol{x},y)$，从而为应用 Jensen 不等式并简化下界函数 $A(\boldsymbol{\delta}|\boldsymbol{w})$ 创造了条件：

$$f^{\#}(\boldsymbol{x},y)=\sum_{i=1}^{n}f_i(x,y) \tag{55}$$

其中，n 表示整个数据集中需要使用的特征函数种类数。一般情况下，$f_i(x,y)$ 是二值函数，故 $f^{\#}(\boldsymbol{x},y)$ 表示样本点 $(\boldsymbol{x},y)$ 中出现的特征函数种类数。于是，将下界函数 $A(\boldsymbol{\delta}|\boldsymbol{w})$ 进行改写：

$$\begin{aligned}A(\boldsymbol{\delta}|\boldsymbol{w})&=\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)+1-\sum_{x}\tilde{P}(x)\sum_{y}P_{\boldsymbol{w}}(y|x)\exp\sum_{i=1}^{n}\delta_i f_i(x,y)\\&=\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}\delta_i f_i(x,y)+1-\\&\quad\sum_{x}\tilde{P}(x)\sum_{y}P_{\boldsymbol{w}}(y|x)\exp\sum_{i=1}^{n}\frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)}\left(\delta_i f^{\#}(\boldsymbol{x},y)\right)\end{aligned} \tag{56}$$

由于 $\frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)}\geqslant 0$ 以及 $\sum\limits_{i=1}^{n}\frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)}=1$，且指数函数为（下）凸函数，利用 Jensen 不

[11]对于适当的 $\boldsymbol{\delta}$，当 $L(\boldsymbol{w}+\boldsymbol{\delta})=L(\boldsymbol{w})$ 成立时，意味着对数似然函数的值已经到达了最大值。

等式，可得：

$$\exp \sum_{i=1}^{n} \frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)} \left(\delta_i f^{\#}(\boldsymbol{x},y)\right) \leqslant \sum_{i=1}^{n} \frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)} \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) \tag{57}$$

于是，可继续将公式 (56) 进行变换：

$$\begin{aligned} A(\boldsymbol{\delta}|\boldsymbol{w}) \geqslant & \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n} \delta_i f_i(x,y) + 1 - \\ & \sum_{x} \tilde{P}(x) \sum_{y} P_{\boldsymbol{w}}(y|x) \sum_{i=1}^{n} \frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)} \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) \end{aligned} \tag{58}$$

从而，得到一个新下界，将右端记为：

$$\begin{aligned} B(\boldsymbol{\delta}|\boldsymbol{w}) = & \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n} \delta_i f_i(x,y) + 1 - \\ & \sum_{x} \tilde{P}(x) \sum_{y} P_{\boldsymbol{w}}(y|x) \sum_{i=1}^{n} \frac{f_i(x,y)}{f^{\#}(\boldsymbol{x},y)} \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) \end{aligned} \tag{59}$$

虽然新下界 $B(\boldsymbol{\delta}|\boldsymbol{w})$ 比原下界 $A(\boldsymbol{\delta}|\boldsymbol{w})$ 要宽松，但是它对分量 δ_i 的偏导数为：

$$\frac{\partial B(\boldsymbol{\delta}|\boldsymbol{w})}{\partial \delta_i} = \sum_{x,y} \tilde{P}(x,y) f_i(x,y) - \sum_{x} \tilde{P}(x) \sum_{y} P_{\boldsymbol{w}}(y|x) f_i(x,y) \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) \tag{60}$$

可以看出，上式除了 δ_i 自身之外，没有包含别的 $\boldsymbol{\delta}$ 分量，这正是迭代优化算法所渴望的特征。于是，令该偏导数为 0，得到：

$$\begin{aligned} & \sum_{x} \tilde{P}(x) \sum_{y} P_{\boldsymbol{w}}(y|x) f_i(x,y) \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) = \sum_{x,y} \tilde{P}(x,y) f_i(x,y) = E_{\tilde{P}}(f_i) \\ & \sum_{x,y} \tilde{P}(x) P_{\boldsymbol{w}}(y|x) f_i(x,y) \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) = E_{\tilde{P}}(f_i) \end{aligned} \tag{61}$$

上面的公式，容易求解得到 δ_i。在迭代的过程中，可依次求解出所有的 δ_i，从而得到 $\boldsymbol{\delta}$ 向量，并使用它来更新权值向量 $\boldsymbol{w}$，直至其收敛，算法停止，最后得到最大熵模型 $P_{\boldsymbol{w}^*}(y|x)$。

如果所有样本点中出现的特征函数种类数目一致，那么 $f^{\#}(\boldsymbol{x},y)$ 是一个常量，令 $f^{\#}(\boldsymbol{x},y) = M$，则可直接从公式 (61) 求解出 δ_i，得到：

$$\delta_i = \frac{1}{M} \log \frac{E_{\tilde{P}}(f_i)}{E_P(f_i)} \tag{62}$$

如果 $f^{\#}(\boldsymbol{x},y)$ 不是常量，那么可以考虑使用数值迭代的方法计算出 δ_i。例如，使用牛顿法 (即 Newton-Raphson 方法)，令公式 (61) 所表示的方程为：

$$g(\delta_i) = \sum_{x,y} \tilde{P}(x) P_{\boldsymbol{w}}(y|x) f_i(x,y) \exp\left(\delta_i f^{\#}(\boldsymbol{x},y)\right) - E_{\tilde{P}}(f_i) = 0 \tag{63}$$

迭代公式为：

$$\delta_i^{(k+1)} = \delta_i^{(k)} - \frac{g\left(\delta_i^{(k)}\right)}{g'\left(\delta_i^{(k)}\right)} \tag{64}$$

方程 $g(\delta_i)$ 有单根。因此，只要适当地选取初始值 $\delta_i^{(0)}$，牛顿法将会很快收敛而得到解 δ_i^*。

下面给出 IIS 算法。

算法 **2.1**（*IIS* 算法）

> Input:
>
> Feature Function: f_i, $i = 1, 2, \ldots, n$
>
> Train Dataset: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$
>
> Output:
>
> Model Parameters $\boldsymbol{w}^*$: $(w_1^*, w_2^*, \ldots, w_n^*)$
>
> Algorithm:
>
> $w_i^{(1)} = 0$, $i = 1, 2, \ldots, n$
>
> for $t = 1, 2 \cdots$ T:
>
> for $i = 1, 2 \cdots n$:
>
> $\sum_{x,y} \tilde{P}(x) P_{\boldsymbol{w}}(y|x) f_i(x, y) \exp\left(\delta_i f^{\#}(\boldsymbol{x}, y)\right) = E_{\tilde{P}}(f_i) \quad \Rightarrow \quad \delta_i$
>
> $w_i^{(t+1)} = w_i^{(t)} + \delta_i$

2.4.3 拟牛顿法：BFGS 算法

在牛顿法中，需要计算 Hessian 矩阵的逆矩阵。当自变量的维度很大时，所需的时间与空间都很多。

拟牛顿法 (Quasi-Newton Method) 采取近似计算的方式，避免直接计算 Hessian 矩阵。它的基本思想是，从某个初始的正定矩阵开始，迭代地近似计算出 Hessian 矩阵或逆矩阵，然后作用于自变量，对其进行优化[12]。

BFGS(Broyden-Fletcher-Goldfarb-Shanno) 算法是最流行的拟牛顿算法之一，它对 Hessian 矩阵进行近似计算。下面，讨论针对最大熵模型的 BFGS 算法。

对于最大熵模型 (公式 (33))：

$$P_{\boldsymbol{w}}(y|x) = \frac{1}{Z_{\boldsymbol{w}}(x)} \exp\left(\sum_{i=1}^{n} w_i f_i(x, y)\right) \tag{65}$$

[12] 关于拟牛顿法的详细讨论，请阅读《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

其中：

$$Z_{\boldsymbol{w}}(x)=\sum_{y}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right) \tag{66}$$

待优化的目标函数 (公式39) 为[13]：

$$\begin{aligned}\mathrm{L}(\boldsymbol{w})=-L(\boldsymbol{w})&=-\left(\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_i f_i(x,y)-\sum_{x}\tilde{P}(x)\log Z_{\boldsymbol{w}}(x)\right)\\&=\sum_{x}\tilde{P}(x)\log Z_{\boldsymbol{w}}(x)-\sum_{x,y}\tilde{P}(x,y)\sum_{i=1}^{n}w_i f_i(x,y)\end{aligned} \tag{67}$$

为应用 BFGS 算法，需要求解偏导数 $\frac{\partial \mathrm{L}(\boldsymbol{w})}{\partial w_i}$：

$$\begin{aligned}\frac{\partial \mathrm{L}(\boldsymbol{w})}{\partial w_i}&=\sum_{x}\tilde{P}(x)\frac{\sum_{y}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)}{Z_{\boldsymbol{w}}(x)}f_i(x,y)-\sum_{x,y}\tilde{P}(x,y)f_i(x,y)\\&=\sum_{x}\tilde{P}(x)\sum_{y}P_{\boldsymbol{w}}(y|x)f_i(x,y)-\sum_{x,y}\tilde{P}(x,y)f_i(x,y)\\&=\sum_{x,y}\tilde{P}(x)P_{\boldsymbol{w}}(y|x)f_i(x,y)-\sum_{x,y}\tilde{P}(x,y)f_i(x,y)\\&=\mathbb{E}_{P}(f_i(x,y))-\mathbb{E}_{\tilde{P}}(f_i(x,y))\end{aligned} \tag{68}$$

下面给出 BFGS 算法。

算法 2.2（*BFGS* 算法）

[13]按照惯例，将最大化问题转换为等价的最小化问题。

Input:

Global Feature Function: f_i, $i = 1, 2, \ldots, n$

Train Dataset: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Model Parameters: $\boldsymbol{w}^*$: $(w_1^*, w_2^*, \ldots, w_n^*)$

Algorithm:

Initialize randomly: $w_i^{(1)}$, $i = 1, 2, \ldots, n$

$\boldsymbol{B}_1 = \boldsymbol{I}$

for $t = 1, 2 \cdots T$:

for $i = 1, 2 \cdots n$:

$\nabla \mathrm{L}(w_i^{(t)}) = \frac{\partial \mathrm{L}(\boldsymbol{w}^{(t)})}{\partial w_i^{(t)}} = \mathbb{E}_P(f_i(x, y)) - \mathbb{E}_{\tilde{P}}(f_i(x, y))$

$\boldsymbol{B}_t \Delta w_i^{(t)} = -\nabla \mathrm{L}(w_i^{(t)}) \quad \Rightarrow \quad \Delta w_i^{(t)}$

Line Search for best λ_i^*:

$\mathrm{L}(w_i^{(t)} + \lambda_i^* \Delta w_i^{(t)}) = \min_{\lambda \geqslant 0} \mathrm{L}(w_i^{(t)} + \lambda \Delta w_i^{(t)})$

$w_i^{(t+1)} = w_i^{(t)} + \lambda_i^* \Delta w_i^{(t)}$

$\nabla \mathrm{L}(w_i^{(t+1)}) = \frac{\partial \mathrm{L}(\boldsymbol{w}^{(t+1)})}{\partial w_i^{(t+1)}} = \mathbb{E}_P(f_i(x, y)) - \mathbb{E}_{\tilde{P}}(f_i(x, y))$

$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\Delta \mathbf{G} \Delta \mathbf{G}^T}{\Delta \mathbf{G}^T \Delta \mathbf{W}} - \frac{\mathbf{B}_t \Delta \mathbf{W} (\mathbf{B}_t \Delta \mathbf{W})^T}{(\mathbf{B}_t \Delta \mathbf{W})^T \Delta \mathbf{W}}$

where: $\Delta \mathbf{G} = \nabla \mathrm{L}(w_i^{(t+1)}) - \nabla \mathrm{L}(w_i^{(t)})$

$\Delta \mathbf{W} = w_i^{(t+1)} - w_i^{(t)}$

3 最大熵模型与逻辑回归

在《机器学习》课程系列之《逻辑回归》章节中，我们从广义线性模型的角度，推导出了用于多分类逻辑回归的 Softmax 函数。

现在，假设给定一个数据集 T：

$$T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\},$$

能否仅仅使用最大熵原理就能够推导出多分类逻辑回归的 Softmax 函数？答案是肯定的，下面将介绍具体的推导思路与过程。

首先，分析一下多分类函数应该具备的功能或需要满足的条件。假设数据集中类别的个数为 K，设多分类函数为 $\sigma(\boldsymbol{x}_i, k)$，它表示将样本点 $\boldsymbol{x}_i$ 划分为第 k 个类别的概率。

显然，多分类函数 $\sigma(\boldsymbol{x}_i,k)$ 需要满足下面的概率先决条件：

$$
\begin{aligned}
&\sigma(\boldsymbol{x}_i,k) \geqslant 0 \\
&\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k) = 1
\end{aligned}
\tag{69}
$$

另外，在数据集 T 给定的情况下，多分类函数 $\sigma(\boldsymbol{x}_i,k)$ 还必须能够“精确地”识别出每个数据样本点 $\boldsymbol{x}_i$ 的类别。然而，需要注意的是，精确度 (Accuracy) 不是一个绝对因素，而且在一些情况下，分类函数对提供的数据集真的不能做到 100% 的精确度，这涉及到一些因素。例如，在数据集本身含有噪声的情况下，以 100% 精确度识别出所有数据样本点的分类函数可能是有害的。此外，精确度还与分类函数的泛化能力有关。过于精确的分类，被称为过拟合 (Over-fit)。在过拟合的情况下，模型的泛化能力较差——这样的模型对于未见数据的识别精确度不高，这是我们需要极力避免的。一般而言，提高模型对可见或给定数据集的性能，不是模型的最终目标，而只是实现模型对未见数据的识别能力 (泛化能力) 的必经过程——可见与未见数据的识别能力需要平衡。在极端情况下，一个以查表方式工作的模型，虽然能够完美地识别出可见数据集，但是对于未见数据却无能为力。一般而言，对于一定规模的数据集，这种查表函数的分类规则将非常复杂，这不符合“奥卡姆剃刀”原理。从概率分布的角度而言，这种形式的分布也是复杂的，因而其熵也较低。根据熵定义，极其简单的均匀分布将取得熵的最大值。因此，模型的熵是衡量模型复杂度的一个重要准则，也是衡量模型泛化能力的一个重要因素。在缺少更多信息的情况下，选取熵最大的模型 (最大熵模型) 是较为理想的。

综上所述，对于给定的数据集，分类函数需要考虑以下两个因素：

- 分类函数能够对可见数据集进行识别；
- 在没有更多的信息用于选取分类函数时，以最大熵原理选取分类函数；

由于第 2 项的存在，训练后的分类函数将具有上述平衡能力。

因此，针对第 1 项，提出如下的约束条件：

$$
\begin{aligned}
&y_i^k = \sigma(\boldsymbol{x}_i,k) \quad \Rightarrow \quad y_i^k\boldsymbol{x}_i = \sigma(\boldsymbol{x}_i,k)\boldsymbol{x}_i \quad \Rightarrow \\
&\sum_{k=1}^{K} y_i^k\boldsymbol{x}_i = \sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)\boldsymbol{x}_i \quad \Rightarrow \quad \sum_{i=1}^{N}\sum_{k=1}^{K} y_i^k\boldsymbol{x}_i = \sum_{i=1}^{N}\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)\boldsymbol{x}_i
\end{aligned}
\tag{70}
$$

其中，N 表示数据集中样本点的个数，K 表示数据集中类别的个数。

针对第 2 项，利用最大熵原理提出如下优化目标：

$$
\begin{aligned}
&\max_{\sigma(\boldsymbol{x}_i,k)} -\sigma(\boldsymbol{x}_i,k)\log\sigma(\boldsymbol{x}_i,k) \quad \Rightarrow \quad \max_{\sigma(\boldsymbol{x}_i,k)} \sum_{k=1}^{K} -\sigma(\boldsymbol{x}_i,k)\log\sigma(\boldsymbol{x}_i,k) \\
&\max_{\sigma(\boldsymbol{x}_i,k)} -\sum_{i=1}^{N}\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)\log\sigma(\boldsymbol{x}_i,k) \quad \Rightarrow \quad \min_{\sigma(\boldsymbol{x}_i,k)} \sum_{i=1}^{N}\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)\log\sigma(\boldsymbol{x}_i,k)
\end{aligned} \tag{71}
$$

上述优化目标，利用了数据样本点以及类别之间的独立性假设。

于是，对于上述带约束的优化问题，可以使用拉格朗日优化函数表示如下：

$$
\begin{aligned}
L(\sigma(\boldsymbol{x}_i,k),\boldsymbol{\alpha},\boldsymbol{\beta}) = &\sum_{i=1}^{N}\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)\log\sigma(\boldsymbol{x}_i,k) + \sum_{i=1}^{N}\alpha_i\left(\sum_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)-1\right) + \\
&\sum_{i=1}^{N}\sum_{k=1}^{K}\boldsymbol{\beta}_{i,k}^{T}\left(y_i^k\boldsymbol{x}_i - \sigma(\boldsymbol{x}_i,k)\boldsymbol{x}_i\right)
\end{aligned} \tag{72}
$$

注意，为便于求解，没有将不等式约束 $\sigma(\boldsymbol{x}_i,k)\geqslant 0$ 置入优化函数中。求解后，我们将验证该约束是否满足即可。

下面求解拉格朗日优化函数 $L(\sigma(\boldsymbol{x}_i,k),\boldsymbol{\alpha},\boldsymbol{\beta})$ 关于 $\sigma(\boldsymbol{x}_i,k)$ 的偏导数：

$$
\frac{\partial L(\sigma(\boldsymbol{x}_i,k),\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial\sigma(\boldsymbol{x}_i,k)} = \log\sigma(\boldsymbol{x}_i,k) + 1 + \alpha_i - \boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i \tag{73}
$$

令上述偏导数为 0，解得：

$$
\begin{aligned}
&\log\sigma(\boldsymbol{x}_i,k) + 1 + \alpha_i - \boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i = 0 \quad \Rightarrow \\
&\sigma(\boldsymbol{x}_i,k) = e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i-\alpha_i-1} \quad \Rightarrow \\
&\sigma(\boldsymbol{x}_i,k) = \frac{e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i-1}}{e^{\alpha_i}}
\end{aligned} \tag{74}
$$

可以看出，由于指数函数的存在，约束条件 $\sigma(\boldsymbol{x}_i,k)>0$ 成立。对上式两端按 k 求和，并利用公式 $\sum\limits_{k=1}^{K}\sigma(\boldsymbol{x}_i,k)=1$，可得：

$$
e^{\alpha_i} = \sum_{k=1}^{K} e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i-1} \tag{75}
$$

将其代入公式 (74) 中，可得：

$$
\sigma(\boldsymbol{x}_i,k) = \frac{e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i-1}}{\sum\limits_{k=1}^{K} e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i-1}} = \frac{e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i}}{\sum\limits_{k=1}^{K} e^{\boldsymbol{\beta}_{i,k}^{T}\boldsymbol{x}_i}} \tag{76}
$$

需要注意的是，在前文设计最大熵模型的约束条件与拉格朗日优化函数时，我们引入了参数冗余性——为数据集 T 中的每个样本点的每个类别引入了一个权

值向量 $\boldsymbol{\beta}_{i,k}$(拉格朗日乘数的向量形式)。这种处理方式，相当于让最大熵模型完全精确地匹配数据集 T，显然这是极不合理的。因此，需要对其施加最后一个约束条件，即令 $\boldsymbol{\beta}_k = \boldsymbol{\beta}_{i,k}$。这样，就降低了模型完全精确匹配给定数据集的可能性，在模型的匹配能力与泛化能力之间取得了某种平衡。于是，公式 (76) 重新写为：

$$\sigma(\boldsymbol{x}_i, k) = \frac{e^{\boldsymbol{\beta}_k^T \boldsymbol{x}_i}}{\sum\limits_{k=1}^{K} e^{\boldsymbol{\beta}_k^T \boldsymbol{x}_i}} \tag{77}$$

这就是逻辑回归模型中的 Softmax 函数。

因此，基于上述认识，可以将拉格朗日优化函数 (公式 (72)) 修改为：

$$\begin{aligned} L(\sigma(\boldsymbol{x}_i, k), \boldsymbol{\alpha}, \boldsymbol{\beta}) = &\sum_{i=1}^{N} \sum_{k=1}^{K} \sigma(\boldsymbol{x}_i, k) \log \sigma(\boldsymbol{x}_i, k) + \sum_{i=1}^{N} \alpha_i \left(\sum_{k=1}^{K} \sigma(\boldsymbol{x}_i, k) - 1 \right) + \\ &\sum_{k=1}^{K} \boldsymbol{\beta}_k^T \sum_{i=1}^{N} \left(y_i^k \boldsymbol{x}_i - \sigma(\boldsymbol{x}_i, k) \boldsymbol{x}_i \right) \end{aligned} \tag{78}$$

可以看出，上式对原公式的第 3 项进行了修改，消除了 $\boldsymbol{\beta}$ 向量的冗余性。显然，其优化结果将与公式 (77) 一致。

4 最大熵模型实验

0.1 改进的迭代尺度 (Improved Iterative Scaling, IIS) 算法

In [24]:
```python
import math
import copy

class MaximumEntropy:
    def __init__(self, max_step = 1000, threshold = 0.005):
        self.max_step = max_step # 最大迭代步数
        self.threshold = threshold # 收敛阈值，小于该阈值，停止迭代

    # 计算样本点 x 的 Zw(x) 值，它是条件分布 Pw(y|x) 的归一化因子
    #x: 向量
    def zwx(self, x):
        zx = 0
        for y in self.labels: # 每个标签类别
            zx_y = 0
            for xi in x: # 每个特征分量
                if (xi, y) in self.x_y:
                    zx_y += self.x_y[(xi, y)][1] #sum wi*fi(xi,y)
            zx += math.exp(zx_y)
        return zx

    # 计算样本点 x 的条件分布 Pw(y|x)
    #x: 向量
    #y: 标量
    def pyx(self, x, y):
        zx = self.zwx(x)
        zx_y = 0
        for xi in x: # 每个特征分量
            if (xi, y) in self.x_y:
                zx_y += self.x_y[(xi, y)][1] #sum wi*fi(xi,y)
        return math.exp(zx_y) / zx

    # 计算样本点分量 xi 的模型期望 EPF: 特征函数 fi(xi,y) 关于条件分布
    # Pw(y|xi)(即最大熵模型) 的期望，EPF = sum P(xi)Pw(y|xi)fi(xi,y)
    # xi: 样本点 x 的分量
    # y: 标量
    # P(xi)Pw(y|xi)=P(xi,y)，而从本类 P(xi) 和 Pw(y|xi) 的计算公式可以
    # 看出，它们分别等同于 P(x) 和 Pw(y|x),
    # 另外，从 fit 函数中的 P(xi,y) 计算公式又可以看出 P(xi,y)<=>P(x,y)
    # (此处，x 表示 xi 所在的样本点)。
    # 因此，最大熵模型需要满足约束条件：P(x)Pw(y|x)=P(x,y)，从公式推导可
    # 以看出这一点。
    def epf(self, xi, y, X):
        ep = 0
        for x in X:
            if xi not in x:
                continue #fi(x, y)=0
```

```
            pyx = self.pyx(x, y) # 计算单个样本点的 P(y|x)
            # 计算 sum P(x)P(y|x)fi(x,y), 其中 P(x)=1/N, fi(x,y)=1
            ep += pyx/len(X)
        return ep

    # 优化最大熵模型的对偶函数或对数似然函数
    def fit(self, X, Y):
        N = len(X) # 样例个数
        # 样例的特征分量个数，此处假定所有样例的特征分量个数都一样（常数），
        # 最大熵模型允许样例的维度不一样
        M = len(X[0])
        self.labels = set(Y) # 类别（不重复）

        # 统计所有 (x,y) 对的相关信息
        self.x_y = {}
        for i_sample in range(N):
            y = Y[i_sample]
            for xi in X[i_sample]:
                if (xi, y) in self.x_y:
                    self.x_y[(xi, y)][0] += 1
                else:
                    # 分别表示特征 (x,y) 的次数、拉格朗日乘数 w、联合期望
                    # EPF_HAT
                    self.x_y[(xi, y)] = [1, 0, 0]
        # 计算所有 (xi,y) 对的联合期望 EPF_HAT: 特征函数 fi(xi,y) 关于经验
        # 分布 P(xi,y) 的期望, EPF_HAT = sum P(xi,y)fi(xi,y)
        # 从本类 P(xi,y) 的计算公式可以看出, P(xi,y)<=>P(x,y), 此处 x 表示
        # xi 所在的样本点
        for x_y in self.x_y: # 遍历每个特征对 (xi,y)
            self.x_y[x_y][2] = self.x_y[x_y][0] / N

        # 应用 IIS 算法优化对偶函数或对数似然函数 L(P,w)
        for step in range(self.max_step):
            last_x_y = copy.deepcopy(self.x_y)
            # 计算所有 (x,y) 对的模型期望 EPF: 特征函数 fi(x,y) 关于条件分
            # 布 Pw(y|x)(即最大熵模型) 的期望, EPF = sum P(x)Pw(y|x)fi(x,y)
            for x_y in self.x_y: # 遍历每个特征对 (x,y)
                ep = self.epf(*x_y, X) # 计算第 i 个特征的模型期望
                self.x_y[x_y][1] += math.log(self.x_y[x_y][2]/ep)/M
            # 判断任意权值之差是否小于阈值
            stop = True
            for x_y in self.x_y:
                if abs(last_x_y[x_y][1]-self.x_y[x_y][1]) >=
                 self.threshold:
                    stop = False
                    break
            if stop:
                print('After {} step, maximum entropy model training
```

```
                completed!'.format(step))
                break
        print()
        print(self.x_y)

    def predict(self, X):
        results = []
        for x in X:
            zx = self.zwx(x)
            result = []
            for y in self.labels:
                zx_y = 0
                for xi in x:
                    if (xi, y) in self.x_y:
                        zx_y += self.x_y[(xi, y)][1]
                result.append((y, math.exp(zx_y)/zx))
            results.append(result)
        return results
```

生成数据集

In [25]:
```
import pandas as pd

dataset = [['sunny', 'hot', 'high', 'false', 'no'],
           ['sunny', 'hot', 'high', 'true', 'no'],
           ['overcast', 'hot', 'high', 'false', 'yes'],
           ['rainy', 'mild', 'high', 'false', 'yes'],
           ['rainy', 'cool', 'normal', 'false', 'yes'],
           ['rainy', 'cool', 'normal', 'true', 'no'],
           ['overcast', 'cool', 'normal', 'true', 'yes'],
           ['sunny', 'mild', 'high', 'false', 'no'],
           ['sunny', 'cool', 'normal', 'false', 'yes'],
           ['rainy', 'mild', 'normal', 'false', 'yes'],
           ['sunny', 'mild', 'normal', 'true', 'yes'],
           ['overcast', 'mild', 'high', 'true', 'yes'],
           ['overcast', 'hot', 'normal', 'false', 'yes'],
           ['rainy', 'mild', 'high', 'true', 'no']]
labels = ['Outlook', 'Temperature', 'Humidity', 'Windy', 'Play']
train_data = pd.DataFrame(dataset, columns = labels)
X = np.array(train_data.iloc[:, :-1])
Y = np.array(train_data.iloc[:, -1])
train_data
```

Out[25]:

	Outlook	Temperature	Humidity	Windy	Play
0	sunny	hot	high	false	no
1	sunny	hot	high	true	no
2	overcast	hot	high	false	yes
3	rainy	mild	high	false	yes

```
4      rainy     cool  normal  false  yes
5      rainy     cool  normal   true   no
6   overcast     cool  normal   true  yes
7      sunny     mild    high  false   no
8      sunny     cool  normal  false  yes
9      rainy     mild  normal  false  yes
10     sunny     mild  normal   true  yes
11  overcast     mild    high   true  yes
12  overcast      hot  normal  false  yes
13     rainy     mild    high   true   no
```

```
In [26]: me = MaximumEntropy()
         me.fit(X, Y)
         me.predict([['overcast', 'mild', 'high', 'false'],
                     ['overcast', 'hot', 'high', 'true'],
                     ['overcast', 'hot', 'high', 'false'],
                     ['rainy', 'hot', 'high', 'false'],
                    ])
```

```
After 661 step, maximum entropy model training completed!

{('hot', 'no'): [2, 0.06209367542798627, 0.14285714285714285],
('high', 'yes'): [3, -2.238436267220732, 0.21428571428571427],
('hot', 'yes'): [2, 0.00647972101280744, 0.14285714285714285],
('rainy', 'no'): [2, 2.91587923674726, 0.14285714285714285],
('sunny', 'yes'): [2, -5.395840212106481, 0.14285714285714285],
('true', 'yes'): [3, -1.7212783901580846, 0.21428571428571427],
('mild', 'yes'): [4, 2.0403040677696396, 0.2857142857142857],
('false', 'yes'): [6, 1.634484168985822, 0.42857142857142855],
('cool', 'no'): [1, 3.6076010156607587, 0.07142857142857142],
('high', 'no'): [4, 1.5948247077075397, 0.2857142857142857],
('normal', 'yes'): [6, 1.8218352470476797, 0.42857142857142855],
('normal', 'no'): [1, -9.463290583750394,  0.07142857142857142],
('cool', 'yes'): [3, -1.4128887226475888, 0.21428571428571427],
('mild', 'no'): [2, -3.514271177403205, 0.14285714285714285],
('overcast', 'yes'): [4, 5.262612273515946, 0.2857142857142857],
('rainy', 'yes'): [3, -1.9705537826863249, 0.21428571428571427],
('true', 'no'): [3, 2.096617449552706, 0.21428571428571427],
('false', 'no'): [2, -4.137983158938763, 0.14285714285714285],
('sunny', 'no'): [3, 4.0680729722680145, 0.21428571428571427]}
```

```
Out[26]: [[('yes', 0.9999971161857726), ('no', 2.8838142274653416e-06)],
          [('yes', 0.07986677792510691), ('no', 0.9201332220748931)],
          [('yes', 0.9992127716589034), ('no', 0.0007872283410965032)],
          [('yes', 0.047297708269788956), ('no', 0.952702291730211)]]
```

测试另一个数据集

In [27]:
```
datasets = [['youth-age', 'false', 'no', 'normal', 'reject'],
            ['youth-age', 'false', 'no', 'good', 'reject'],
            ['youth-age', 'true', 'no', 'good', 'accept'],
            ['youth-age', 'true', 'yes', 'normal', 'accept'],
            ['youth-age', 'false', 'no', 'normal', 'reject'],
            ['middle-age', 'false', 'no', 'normal', 'reject'],
            ['middle-age', 'false', 'no', 'good', 'reject'],
            ['middle-age', 'true', 'yes', 'good', 'accept'],
            ['middle-age', 'false', 'yes', 'excellent', 'accept'],
            ['middle-age', 'false', 'yes', 'excellent', 'accept'],
            ['old-age', 'false', 'yes', 'excellent', 'accept'],
            ['old-age', 'false', 'yes', 'good', 'accept'],
            ['old-age', 'true', 'no', 'good', 'accept'],
            ['old-age', 'true', 'no', 'excellent', 'accept'],
            ['old-age', 'false', 'no', 'normal', 'reject'],
           ]
labels = [u'年龄', u'有工作', u'有自己的房子', u'信贷情况', u'类别']
train_data = pd.DataFrame(datasets, columns = labels)
X = np.array(train_data.iloc[:, :-1])
Y = np.array(train_data.iloc[:, -1])
train_data
```

Out[27]:

	年龄	有工作	有自己的房子	信贷情况	类别
0	youth-age	false	no	normal	reject
1	youth-age	false	no	good	reject
2	youth-age	true	no	good	accept
3	youth-age	true	yes	normal	accept
4	youth-age	false	no	normal	reject
5	middle-age	false	no	normal	reject
6	middle-age	false	no	good	reject
7	middle-age	true	yes	good	accept
8	middle-age	false	yes	excellent	accept
9	middle-age	false	yes	excellent	accept
10	old-age	false	yes	excellent	accept
11	old-age	false	yes	good	accept
12	old-age	true	no	good	accept
13	old-age	true	no	excellent	accept
14	old-age	false	no	normal	reject

In [28]:
```
me = MaximumEntropy(max_step = 20000, threshold = 0.00005)
me.fit(X, Y)
```

```
After 19303 step, maximum entropy model training completed!

{('no', 'accept'): [3, -4.135416815833319, 0.2], ('middle-age', 'accept'):
[3, 1.7527725553214912, 0.2], ('old-age', 'accept'): [4, 1.9334272600663023,
0.26666666666666666], ('excellent', 'accept'): [4, 5.495961967387707,
0.26666666666666666], ('normal', 'reject'): [4, 1.0993440265131085,
```

```
0.26666666666666666], ('false', 'reject'): [6, 2.807374818150756, 0.4],
('good', 'reject'): [2, -0.6898320995895466, 0.13333333333333333],
('false', 'accept'): [4, -4.060431387625057, 0.26666666666666666],
('good', 'accept'): [4, 0.38752990650040114, 0.26666666666666666],
('youth-age', 'accept'): [2, 3.0603392441365322, 0.13333333333333333],
('true', 'accept'): [5, 7.179910762509736, 0.3333333333333333],
('no', 'reject'): [6, 2.134461132899133, 0.4], ('yes', 'accept'):
[6, 5.291723929592758, 0.4], ('old-age', 'reject'): [1, -6.853247773072396,
0.06666666666666667], ('middle-age', 'reject'): [2, -2.5009272993075147,
0.13333333333333333], ('youth-age', 'reject'): [3, -1.8850684170026522, 0.2],
('normal', 'accept'): [1, -3.9625009584142985, 0.06666666666666667]}
```

```
In [29]: XTEST = [['old-age', 'false', 'no', 'normal'],
                  ['middle-age', 'true', 'no', 'normal'],
                  ['middle-age', 'false', 'no', 'normal'],
                  ['youth-age', 'false', 'no', 'normal'],
                  ['youth-age', 'false', 'no', 'good'],
                  ['youth-age', 'false', 'yes', 'normal'],
                  ['youth-age', 'false', 'yes', 'good'],
                  ['old-age', 'false', 'yes', 'good'],
                  ['middle-age', 'true', 'no', 'good'],
                  ['youth-age', 'false', 'yes', 'good'],
                  ['youth-age', 'true', 'yes', 'normal'],
                 ]
         me.predict(XTEST)
```

```
Out[29]: [[('accept', 8.166085756384233e-05), ('reject', 0.9999183391424362)],
          [('accept', 0.5254499081744641), ('reject', 0.47455009182553587)],
          [('accept', 8.77814832373704e-07), ('reject', 0.9999991221851676)],
          [('accept', 1.7531029402765733e-06), ('reject', 0.9999982468970597)],
          [('accept', 0.0008122299670197994), ('reject', 0.9991877700329802)],
          [('accept', 0.15544477552731326), ('reject', 0.8445552244726867)],
          [('accept', 0.9884183877857486), ('reject', 0.01158161221425133)],
          [('accept', 0.9997485348784596), ('reject', 0.00025146512154035214)],
          [('accept', 0.9980560640137665), ('reject', 0.001943935986233591)],
          [('accept', 0.9884183877857486), ('reject', 0.01158161221425133)],
          [('accept', 0.9999956927102314), ('reject', 4.307289768680757e-06)]]
```

5 参考文献

1. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

2. 史春奇、卜晶祎、施智平。《机器学习：算法背后的理论与优化》，清华大学出版社，2019 年 7 月第 1 版。

3. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。

4. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。

5. 最大熵模型的推导。https://zhuanlan.zhihu.com/p/83765331.

6. John Mount. The equivalence of logistic regression and maximum entropy models. http://www.win-vector.com/.

7. https://github.com/wzyonggege/statistical-learning-method;