章节 1：从广义线性回归推导出逻辑回归

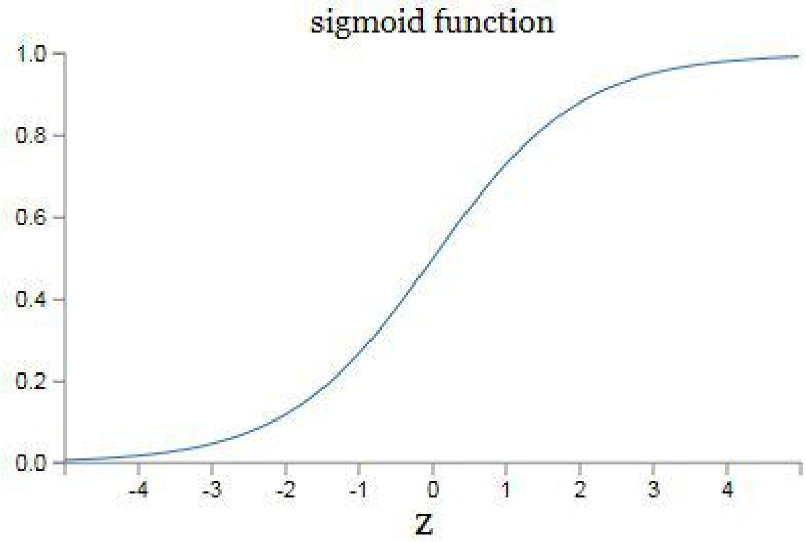
什么是逻辑回归

名字的由来

逻辑回归不是一个回归的算法，逻辑回归是一个分类的算法，就比如卡巴斯基不是司机。那为什么逻辑回归不叫逻辑分类？因为逻辑回归算法是基于多元线性回归的算法。而正因为此，逻辑回归这个分类算法是线性的分类器。未来我们去学的基于决策树的一系列算法，基于神经网络的算法等那些是非线性的算法。SVM 支持向量机的本质是线性的，但是也可以通过内部的核函数升维来变成非线性的算法。

S 形曲线

```
import numpy
import math
import matplotlib.pyplot as plt

def sigmoid(x):
    a = []
    for item in x:
        a.append(1.0/(1.0 + math.exp(-item)))
    return a

x = numpy.arange(-10, 10, 0.1)
y = sigmoid(x)
plt.plot(x,y)
plt.show()
```

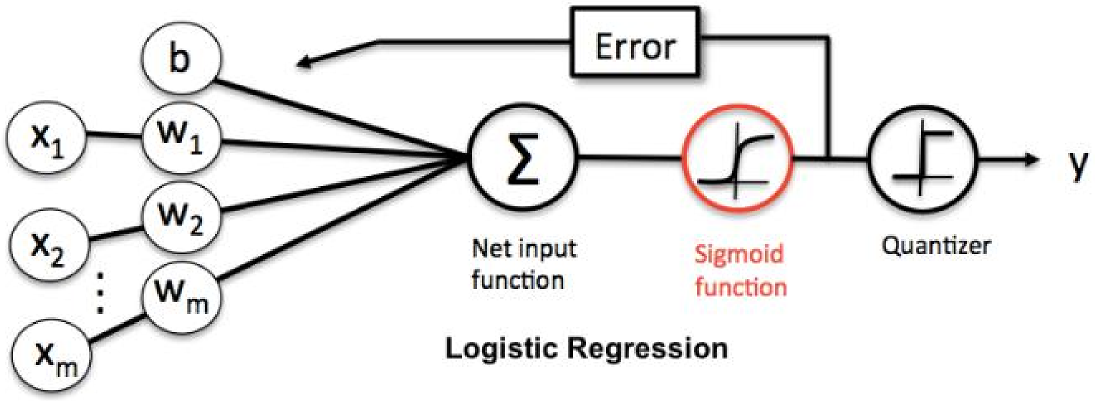


Sigmoid 作用

逻辑回归就是在多元线性回归基础上把结果缩放到 0 到 1 之间。 $h_\theta(x)$ 越接近+1 越是正例， $h_\theta(x)$ 越接近 0 越是负例，根据中间 0.5 分为二类。

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

我们知道分类器的本质就是要找到分界，所以当我们把 0.5 作为分界时，我们要找的就是 $\hat{y} = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = 0.5$ 的时候θ的解，即 $z = \theta^T x = 0$ 时θ的解。



为什么逻辑回归要用 sigmoid 函数

什么事情，都要做到知其然，知其所以然，sigmoid 函数的数学公式推导，我们知道二分类有个特点就是正例的概率+负例的概率=1。一个非常简单的试验是只有两个可能结果的试验，比如正面或反面，成功或失败，有缺陷或没有缺陷，病人康复或未康复。为方便

起见，记这两个可能的结果为 0 和 1，下面的定义就是建立在这类试验基础之上的。

如果随机变量 X 只取 0 和 1 两个值，并且相应的概率为

$$Pr(X=1)=p, Pr(X=0)=1-p, 0<p<1.$$

则称随机变量 X 服从参数为 p 的伯努利分布（0-1 分布），若令 q=1-p，则 X 的概率函数可写：

$$f(x|p)=\begin{cases} p^x q^{1-x}, & x=0,1; \\ 0, & x \neq 0,1. \end{cases}$$

联系上我们的机器学习中的二分类任务，逻辑回归二分类任务中我们会把正例的 label 设置为 1，负例的 label 设置为 0，对于上面公式就是 x=0,1。

广义线性回归

考虑一个分类或回归问题，我们就是想预测某个随机变量 y，y 是某些特征(feature)x 的函数。为了推导广义线性模式，我们必须做出如下三个假设

1. $p(y|x;\theta)$ 服从指数族分布
2. 给了x, 我们的目的是为了预测T(y)的在条件x下的期望。一般情况$T(y)=y$, 这就意味着我们希望预测 $h(x)=E[y|x]$
3. 参数η和输入x 是线性相关的：$\eta=\theta^T x$

指数族分布（The exponential family distribution）

指数族分布有：高斯分布、二项分布、伯努利分布、多项分布、泊松分布、指数分布、beta 分布、拉普拉斯分布、gamma 分布。对于回归来说，如果应变量 y 服从某个指数族分布，那么我们就可以用广义线性回归来建模。比如说如果 y 是服从伯努利分布，我们可以使用逻辑回归（也是一种广义线性模型）。

$$p(y;\eta)=b(y)\exp(\eta^T T(y)-a(\eta))$$

η 是 自然参数（natural parameter，also called thecanonical parameter）。

T(y) 是充分统计量 （sufficient statistic），一般情况下就是 y。

a(η) 是 对数部分函数（log partition function），这部分确保 Y 的分布 p(y:η) 计算的结果加起来（连续函数是积分）等于 1。

伯努利分布其实是指数族分布的一种，推导可以证明：

$$p(y;\phi) = \phi^y(1-\phi)^{1-y}$$

把上式的右边改写成指数分布族形式

$$= \exp(y \log \phi + (1 - y) \log(1 - \phi))$$
$$= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right) y + \log(1-\phi)\right).$$

由此可知，$\eta = \theta^T x = \log(\frac{p}{1-p})$

$$e^{\theta^T x} = \frac{p}{1-p}$$

$$p = e^{\theta^T x} - e^{\theta^T x} \cdot p$$
$$= \frac{e^{\theta^T x}}{1+e^{\theta^T x}}$$
$$= \frac{1}{1+e^{-\theta^T x}}$$

到此，我们可以看出来 sigmoid 函数就是这样推导出来

回看线性回归

线性回归当时不是假设属于高斯分布嘛，高斯分布也是一种指数族分布

当设置方差为 1 时

$$p(y;\mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-\mu)^2\right)$$

指数分布族的形式为

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right)$$

这里面的μ就是指数族分布中的η，所以多元线性回归的形式就是 $\hat{y} = \eta = \theta^T x$

章节 2：推导逻辑回归的损失函数

损失函数推导

这里我们依然会用到最大似然估计思想，根据若干已知的 X,y（训练集）找到一组 W 使得 X 作为已知条件下 y 发生的概率最大。

逻辑回归中既然 g(w,x)的输出含义为 P(y=1|w,x)，那么 P(y=0|w,x)=1-g(w,x)

w1	w2	w3	w4		
x1	x2	x3	x4	y	预测正确的概率
24	3	2	3	1	g(wx)
4	3	2	2	1	g(wx)
23	1	3	3	1	g(wx)
1	2	5	5	0	1-g(wx)
3	12	1	2	0	1-g(wx)

只要让我们的 g(w,x)函数在训练集上预测正确的概率最大，g(w,x)就是最好的解。

$$P(\text{正确}) = \begin{cases} g(w,xi) & \text{当 } yi=1 \text{ 时} \\ 1-g(w,xi) & \text{当 } yi=0 \text{ 时} \end{cases}$$

对于每一条数据预测正确的概率，也可以把下面公式看成是上面式子的合并形式

$$P(\text{正确}) = (g(w,xi))^{y^i} * (1-g(w,xi))^{1-y^i}$$

换符号重写一下

$$\begin{aligned} P(y=1 \mid x;\theta) &= h_\theta(x) \\ P(y=0 \mid x;\theta) &= 1-h_\theta(x) \end{aligned}$$

$$p(y \mid x;\theta) = (h_\theta(x))^y \left(1-h_\theta(x)\right)^{1-y}$$

我们假设训练样本相互独立，那么似然函数表达式为：

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1-h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

同样对似然函数取 log，转换为：

$$\begin{aligned}\ell(\theta) &= \log L(\theta)\\ &= \sum_{i=1}^{m} y^{(i)}\log h(x^{(i)}) + (1-y^{(i)})\log(1-h(x^{(i)}))\end{aligned}$$

总结，得到了逻辑回归的表达式，下一步跟线性回归类似，构建似然函数，然后最大似然估计，最终推导出θ的迭代更新表达式。只不过这里用的不是梯度下降，而是梯度上升，因为这里是最大化似然函数不是最小化似然函数。通常我们一提到损失函数，往往是求最小，这样我们就可以用梯度下降来求解。最终损失函数就是上面公式加负号的形式：

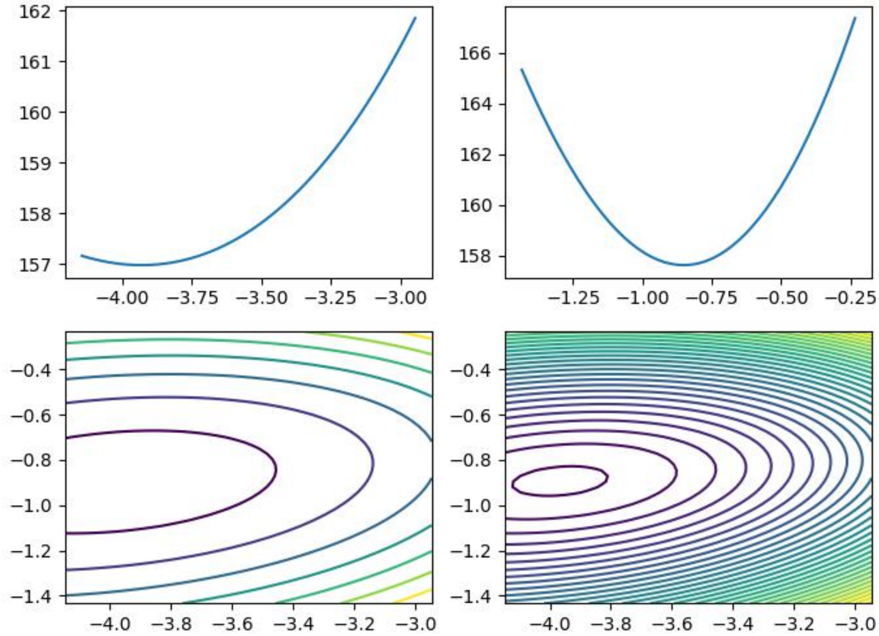
$$J(\theta) = -[\sum_{i=1}^{m} y^{(i)}\log h(x^{(i)}) + (1-y^{(i)})\log(1-h(x^{(i)})]$$

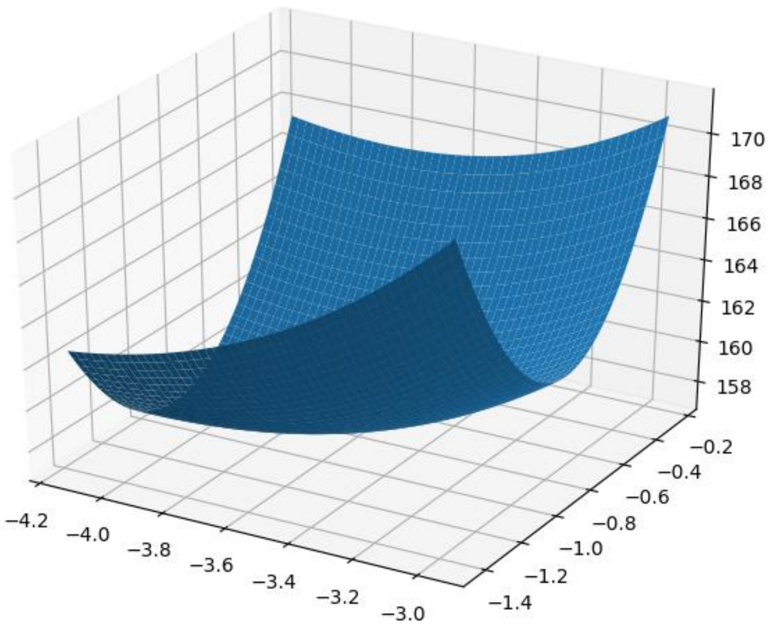
绘制损失函数

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import scale

data = load_breast_cancer()
X, y = scale(data['data'][:, :2]), data['target']
# 求出两个维度对应的数据在逻辑回归算法下的最优解
lr = LogisticRegression(fit_intercept=False)
lr.fit(X, y)
# 分别把两个维度所对应的参数 W1 和 W2 取出来
theta1 = lr.coef_[0, 0]
theta2 = lr.coef_[0, 1]
print(theta1, theta2)



# 已知 W1 和 W2 的情况下，传进来数据的 X，返回数据的 y_predict
def p_theta_function(features, w1, w2):
    z = w1*features[0] + w2*features[1]
    return 1 / (1 + np.exp(-z))



# 传入一份已知数据的 X，y，如果已知 W1 和 W2 的情况下，计算对应这份数据的 Loss
```

```
损失
def loss_function(samples_features, samples_labels, w1, w2):
    result = 0
    # 遍历数据集中的每一条样本，并且计算每条样本的损失，加到 result 身上得到整体的数据集损失
    for features, label in zip(samples_features, samples_labels):
        # 这是计算一条样本的 y_predict
        p_result = p_theta_function(features, w1, w2)
        loss_result = -1*label*np.log(p_result)-(1-label)*np.log(1-p_result)
        result += loss_result
    return result


theta1_space = np.linspace(theta1-0.6, theta1+0.6, 50)
theta2_space = np.linspace(theta2-0.6, theta2+0.6, 50)

result1_ = np.array([loss_function(X, y, i, theta2) for i in theta1_space])
result2_ = np.array([loss_function(X, y, theta1, i) for i in theta2_space])

fig1 = plt.figure(figsize=(8, 6))
plt.subplot(2, 2, 1)
plt.plot(theta1_space, result1_)

plt.subplot(2, 2, 2)
plt.plot(theta2_space, result2_)

plt.subplot(2, 2, 3)
theta1_grid, theta2_grid = np.meshgrid(theta1_space, theta2_space)
loss_grid = loss_function(X, y, theta1_grid, theta2_grid)
plt.contour(theta1_grid, theta2_grid, loss_grid)

plt.subplot(2, 2, 4)
plt.contour(theta1_grid, theta2_grid, loss_grid, 30)

fig2 = plt.figure()
ax = Axes3D(fig2)
ax.plot_surface(theta1_grid, theta2_grid, loss_grid)
```

```
plt.show()
```


162
161
160
159
158
157
−4.00 −3.75 −3.50 −3.25 −3.00
166
164
162
160
158
−1.25 −1.00 −0.75 −0.50 −0.25
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−4.0 −3.8 −3.6 −3.4 −3.2 −3.0
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−4.0 −3.8 −3.6 −3.4 −3.2 −3.0


170
168
166
164
162
160
158
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−4.2 −4.0 −3.8 −3.6 −3.4 −3.2 −3.0

章节 3：逻辑回归如何求解得到最优解模型

对θ求偏导

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta_{\theta_j}} J(\theta)$$

逻辑回归函数求导

$$g(z) = \frac{1}{1+e^{-z}}$$

Logistic 函数求导时有一个特性，这个特性将在下面的推导中用到，这个特性为：

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

回到对逻辑回归损失函数求导

$$\begin{aligned}\frac{\delta}{\delta_{\theta_j}}J(\theta) &= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\frac{1}{h_\theta(x_i)}\frac{\delta}{\delta_{\theta_j}}h_\theta(x_i)-(1-y_i)\frac{1}{1-h_\theta(x_i)}\frac{\delta}{\delta_{\theta_j}}h_\theta(x_i)\right)\\
&= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\frac{1}{g(\theta^Tx_i)}-(1-y_i)\frac{1}{1-g(\theta^Tx_i)}\right)\frac{\delta}{\delta_{\theta_j}}g(\theta^Tx_i)\\
&= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\frac{1}{g(\theta^Tx_i)}-(1-y_i)\frac{1}{1-g(\theta^Tx_i)}\right)g(\theta^Tx_i)(1-g(\theta^Tx_i))\frac{\delta}{\delta_{\theta_j}}\theta^Tx_i\\
&= -\frac{1}{m}\sum_{i=1}^{m}(y_i(1-g(\theta^Tx_i))-(1-y_i)g(\theta^Tx_i))x_i^j\\
&= -\frac{1}{m}\sum_{i=1}^{m}(y_i-g(\theta^Tx_i))x_i^j\\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i))x_i^j\end{aligned}$$

这里我们发现导函数的形式和多元线性回归一样，毕竟都是从广义线性回归来的嘛

作业

模仿我们前面的多元线性回归算法基于梯度下降的实现代码 ,这里把逻辑回归算法基于梯度下降的版本代码实现出来。

实战鸢尾花分类

```
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
```

```
iris = datasets.load_iris()
print(list(iris.keys()))
print(iris['DESCR'])
print(iris['feature_names'])
```

```
X = iris['data'][:, 3:]
print(X)
print(iris['target'])
y = (iris['target'] == 2).astype(np.int)
print(y)
```

```
log_reg = LogisticRegression(solver='sag', max_iter=1000)
log_reg.fit(X, y)
```

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
```

```
print(X_new)
y_proba = log_reg.predict_proba(X_new)
y_hat = log_reg.predict(X_new)
print(y_proba)
print(y_hat)
```

含正则化项的损失函数的偏导：

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\,log\left(h_\theta(x^{(i)})\right)-(1-y^{(i)})\,log\left(1-h_\theta(x^{(i)})\right)\right]+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$\frac{\delta J(\theta)}{\delta\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}+\frac{\lambda}{m}\theta_j$$

正则项超参数 C=1000

```
C : float, default: 1.0
    Inverse of regularization strength; must be a positive float.
    Like in support vector machines, smaller values specify stronger
    regularization.
```

章节 4：逻辑回归如何做多分类

多分类任务

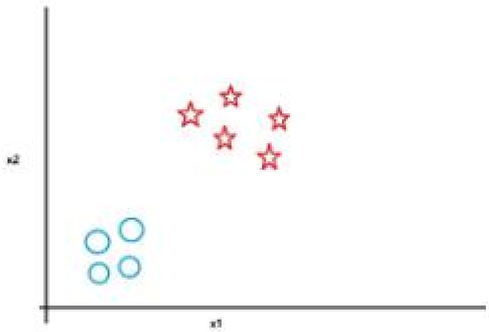
在上面，我们主要使用逻辑回归解决二分类的问题，那对于多分类的问题，也可以用逻辑回归来解决？

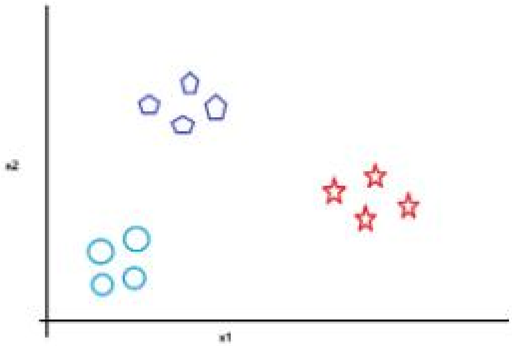
多分类问题

将邮件分为不同类别/标签：工作（y=1），朋友（y=2），家庭（y=3），爱好（y=4）

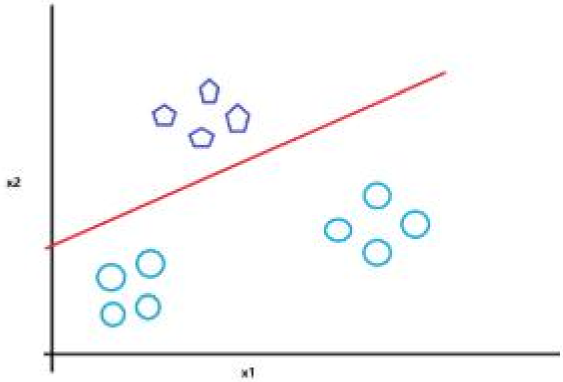
天气分类：晴天（y=1），多云天（y=2），下雨天（y=3），下雪天（y=4）

医学图示（Medical diagrams）：没生病（y=1），感冒（y=2），流感（y=3）





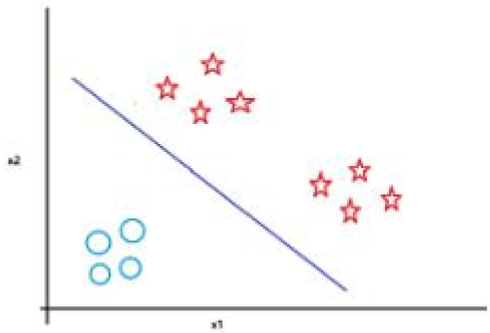
One-vs-all(one-vs-rest)，生成三个假的数据集

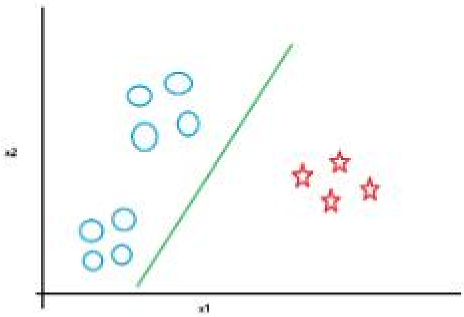


处理过的数据集就是二分类问题，通过逻辑回归可能得到红线区分不同类别

$$h_\theta^{(1)}(x) = P(y = 1|x;\theta)$$

同理





$$h_\theta^{(2)}(x) = P(y = 2|x;\theta) \quad h_\theta^{(3)}(x) = P(y = 3|x;\theta)$$

$$h_\theta^{(i)}(x) = P(y = i|x;\theta)$$

使用不同的函数去预测输入 x，分别计算不同 h(x)的值，然后取其中的最大值。哪个 i 对应的 h(x)越大，就认为属于哪个类。

$$\max_i h_\theta^{(i)}(x)$$

对鸢尾花数据集做多分类

```
y = iris['target']
```

```
log_reg = LogisticRegression(C=1000, multi_class='ovr', solver='sag', max_iter=1000)
```

章节 5：从广义线性回归推导出 Softmax 回归

Softmax 回归

softmax 回归是另一种做多分类的算法。从名字中大家是不是可以联想到广义线性回归，softmax 回归是假设多项分布的，多项分布可以理解为二项分布的扩展。

公式的由来

我们知道，对于伯努利分布，我们采用 Logistic 回归建模。那么我们应该如何处理多分类问题？（比如要进行邮件分类；预测病情属于哪一类等等）。对于这种多项式分布我们使用 softmax 回归建模。

多项式分布的目标值 yε{1,2,3,...,k}；（其中是类别种数）其概率分布为：

$$P(y=i)=\varphi_i$$ 并且，$$\sum_{i=1}^{k}\phi_i=1$$

联合分布的概率密度函数为：于是，多项分布转变为指数分布族的推导如下：

$$\begin{aligned}P(y;\varphi)&=\varphi_1^{I\{y=1\}}\varphi_2^{I\{y=2\}}\ldots\varphi_{k-1}^{I\{y=k-1\}}\varphi_k^{I\{y=k\}}\\&=\varphi_1^{I\{y=1\}}\varphi_2^{I\{y=2\}}\ldots\varphi_{k-1}^{I\{y=k-1\}}\varphi_k^{1-\sum_{i=1}^{k-1}I(y=i)}\\&=\exp(\log\varphi_1^{I\{y=1\}}\varphi_2^{I\{y=2\}}\ldots\varphi_{k-1}^{I\{y=k-1\}}\varphi_k^{1-\sum_{i=1}^{k-1}I(y=i)})\end{aligned}$$

$$= \exp(\sum_{i=1}^{k-1} I(y=i) log\varphi_i + (1-\sum_{i=1}^{k-1} I(y=i)) log\varphi_k)$$

$$= \exp(\sum_{i=1}^{k-1} I(y=i)\log\left(\frac{\varphi_i}{\varphi_k}\right) + log\varphi_k)$$

$$= \exp(\sum_{i=1}^{k-1} T(y)_i\log\left(\frac{\varphi_i}{\varphi_k}\right) + log\varphi_k)$$

$$= \exp(\eta^T \mathrm{T(y)} - \mathrm{a}(\eta))$$

以上推导既是说明多项分布是指数族分布，那么

$$\eta = \begin{bmatrix} log\ \varphi_1/\varphi_k \\ log\ \varphi_2/\varphi_k \\ \vdots \\ log\ \varphi_{k-1}/\varphi_k \end{bmatrix}$$

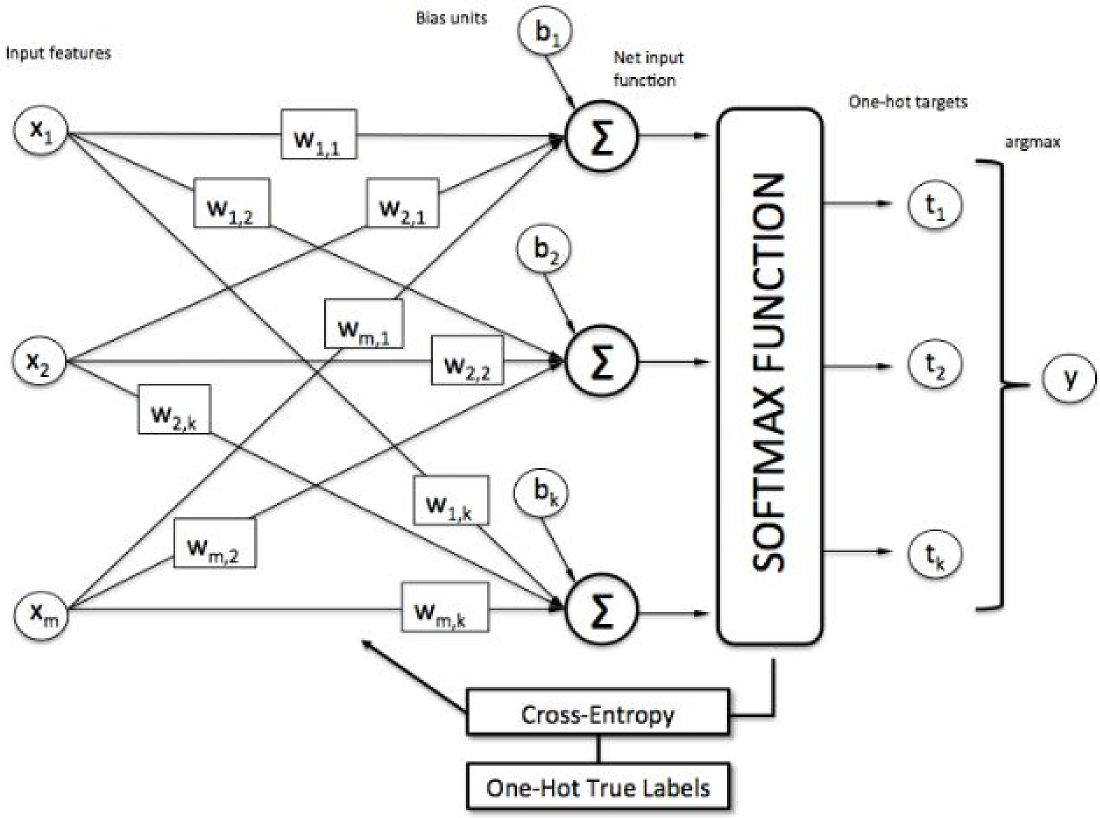
$$\eta_i = log\ \varphi_i/\varphi_k \Longrightarrow \varphi_i = \varphi_k e^{\eta_i}$$

$$\sum_{j=1}^{k} \varphi_i = \sum_{j=1}^{k} \varphi_k e^{\eta_i} = 1 \Longrightarrow \varphi_k = \frac{1}{\sum_{j=1}^{k} e^{\eta_i}}$$

$$\varphi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_i}}$$

至此，我们就得到了 softmax 回归的公式：

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)}=1|x^{(i)};\theta) \\ p(y^{(i)}=2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)}=k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$



Softmax Regression

章节 6：逻辑回归和 Softmax 回归的关系

交叉熵损失函数

$$L(\theta)=\prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta)=\prod_{i=1}^{m}\prod_{j=1}^{k}\varphi_j^{\ I\{y^{(i)}=j\}}$$

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^{m}\log p(y^{(i)}|x^{(i)};\theta)\\ &= \sum_{i=1}^{m}\log\prod_{l=1}^{k}\left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}\right)^{1\{y^{(i)}=l\}}\end{aligned}$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right]$$

逻辑回归和 Softmax 回归

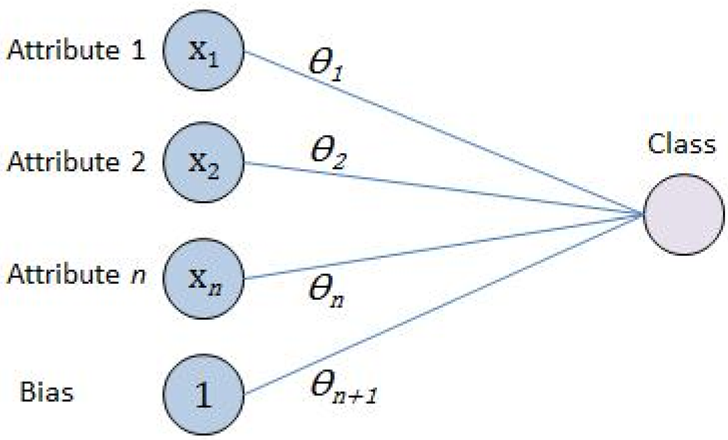
逻辑回归可以看成是 Softmax 回归的特例，就是 k=2 时候的 softmax 回归，因为当我们把 softmax 回归公式 k=2 带入的话

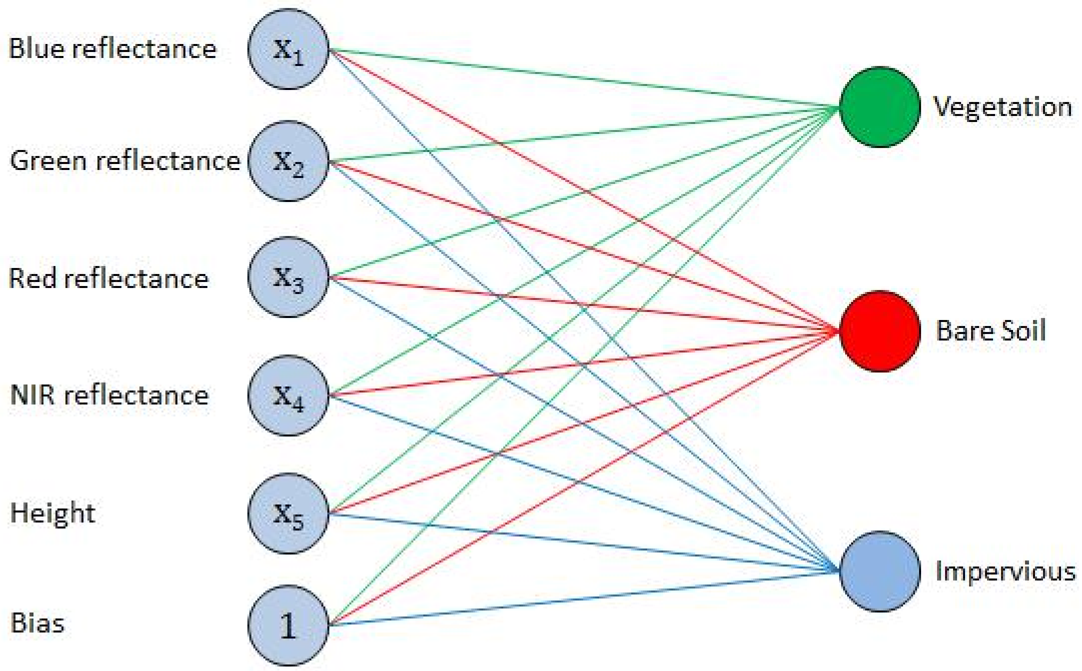
$$h_\theta(x) = \frac{1}{e^{\theta_1^T x} + e^{\theta_2^T x^{(i)}}}\begin{bmatrix} e^{\theta_1^T x} \\ e^{\theta_2^T x} \end{bmatrix}$$

$$\begin{aligned}
h(x) &= \frac{1}{e^{\vec{0}^T x} + e^{(\theta_2-\theta_1)^T x^{(i)}}}\begin{bmatrix} e^{\vec{0}^T x} \\ e^{(\theta_2-\theta_1)^T x} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{1+e^{(\theta_2-\theta_1)^T x^{(i)}}} \\ \frac{e^{(\theta_2-\theta_1)^T x}}{1+e^{(\theta_2-\theta_1)^T x^{(i)}}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{1+e^{(\theta_2-\theta_1)^T x^{(i)}}} \\ 1-\frac{1}{1+e^{(\theta_2-\theta_1)^T x^{(i)}}} \end{bmatrix}
\end{aligned}$$

此时，softmax 回归就是参数为θ2-θ1 的逻辑回归！

同样，k=2 时 softmax 回归的损失函数就是逻辑回归的损失函数！





鸢尾花 Softmax 多分类

```
multi_class : str, {'ovr', 'multinomial', 'auto'}, default: 'ovr'
    If the option chosen is 'ovr', then a binary problem is fit for each
    label. For 'multinomial' the loss minimised is the multinomial loss fit
    across the entire probability distribution, *even when the data is
    binary*. 'multinomial' is unavailable when solver='liblinear'.
    'auto' selects 'ovr' if the data is binary, or if solver='liblinear',
    and otherwise selects 'multinomial'.
```

总结，由 GLM 可知，LinearRegression，LogisticRegression，SoftmaxClassification 三者都可以通过广义线性模型的指数分布族来解释，其参数求解优化过程都可以通过极大似然函数法来实现。当采用极大似然函数求解时，LogisticRegression 等价于内积层+Sigmoid 函数+BinraryCrossEntropy 损失；SoftmaxClassification 等价于内积层+softmax 函数+CategoricalCrossEntropy 损失（在 caffe 里叫做 MultinomialLoss）；LinearRegression 等价于内积层+均方差损失（caffe 里叫做 EuclideanLoss）

章节 7：代码实战音乐分类器

数据集(音乐数据)

名称	修改日期	类型
blues	2018/1/30 13:56	文件夹
classical	2018/1/30 13:56	文件夹
country	2018/1/30 13:56	文件夹
disco	2018/1/30 13:56	文件夹
hiphop	2018/1/30 13:56	文件夹
jazz	2018/1/30 13:56	文件夹
metal	2018/1/30 13:56	文件夹
pop	2018/1/30 13:56	文件夹
reggae	2018/1/30 13:57	文件夹
rock	2018/1/30 13:57	文件夹

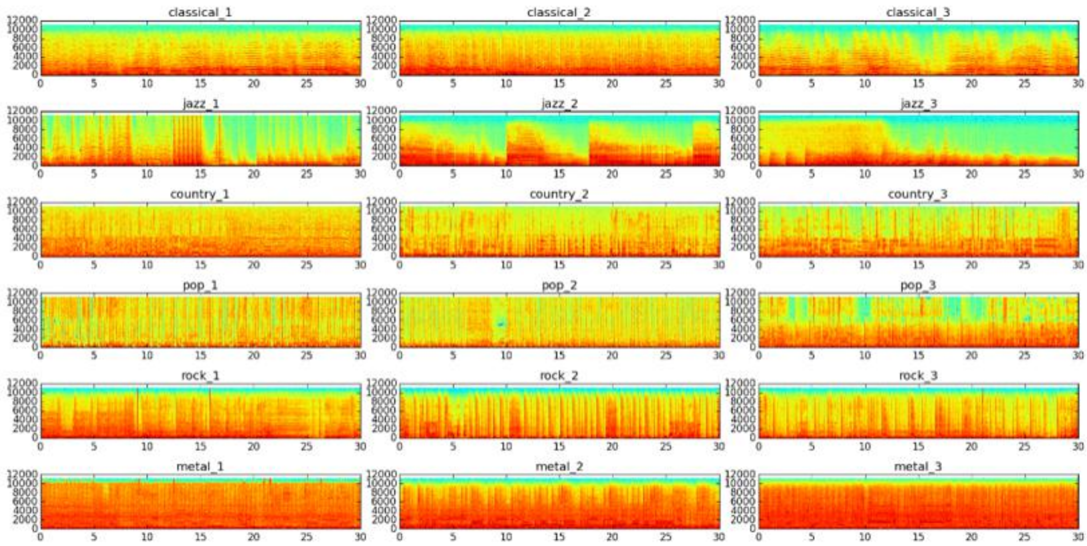
读取数据

可以先把一个 wav 文件读入 python,然后绘制它的频谱图(spectrogram)来看看是什么样的

```
from scipy import fft
from scipy.io import wavfile
from matplotlib.pyplot import specgram
import matplotlib.pyplot as plt

(sample_rate, X) = wavfile.read("D:/genres/metal/converted/metal.00090.au.wav")
print(sample_rate, X.shape)
specgram(X, Fs=sample_rate, xextent=(0, 30))
plt.figure(figsize=(10, 4), dpi=80)
plt.xlabel("time")
plt.ylabel("frequency")
plt.grid(True, linestyle='-', color='0.75')
plt.savefig("D:/metal.00090.au.wav.png", bbox_inches="tight")
```

数据分析

当然,我们也可以把每一种的音乐都抽一些出来打印频谱图以便比较,如下图:

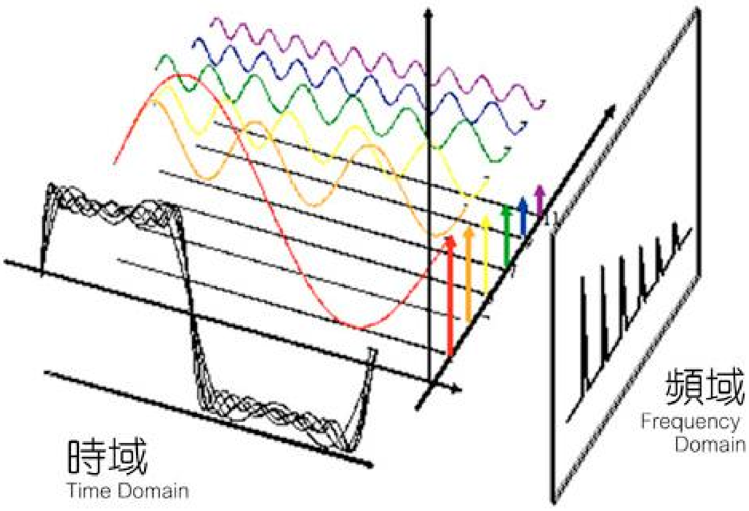



```
def plotSpec(g, n):
    sample_rate, X = wavfile.read("D:/genres/"+g+"/converted/"+g+"."+n+".au.wav")
    specgram(X, Fs=sample_rate, xextent=(0, 30))
    plt.title(g+"_"+n[-1])

plt.figure(num=None, figsize=(18, 9), dpi=80, facecolor='w', edgecolor='k')
plt.subplot(6,3,1);plotSpec("classical","00001");plt.subplot(6,3,2);plotSpec("classical","00002")
plt.subplot(6,3,3);plotSpec("classical","00003");plt.subplot(6,3,4);plotSpec("jazz","00001")
plt.subplot(6,3,5);plotSpec("jazz","00002");plt.subplot(6,3,6);plotSpec("jazz","00003")
plt.subplot(6,3,7);plotSpec("country","00001");plt.subplot(6,3,8);plotSpec("country","00002")
plt.subplot(6,3,9);plotSpec("country","00003");plt.subplot(6,3,10);plotSpec("pop","00001")
plt.subplot(6,3,11);plotSpec("pop","00002");plt.subplot(6,3,12);plotSpec("pop","00003")
plt.subplot(6,3,13);plotSpec("rock","00001");plt.subplot(6,3,14);plotSpec("rock","00002")
plt.subplot(6,3,15);plotSpec("rock","00003");plt.subplot(6,3,16);plotSpec("metal","00001")
plt.subplot(6,3,17);plotSpec("metal","00002");plt.subplot(6,3,18);plotSpec("metal","00003")
```

```
plt.tight_layout(pad=0.4, w_pad=0, h_pad=1.0)
plt.savefig("D:/compare.au.wav.png", bbox_inches="tight")
```

数据预处理

对于一个信号来说，信号强度随时间的变化规律就是时域特性，信号是由哪些单一频率的信号合成的就是频域特性。时域分析与频域分析是对信号的两个观察面。贯穿时域与频域的方法之一，就是传说中的傅里叶变换(Fourier Transformation)。傅里叶原理表明：任何连续测量的时序或信号，都可以表示为不同频率的正弦波信号的无限叠加。

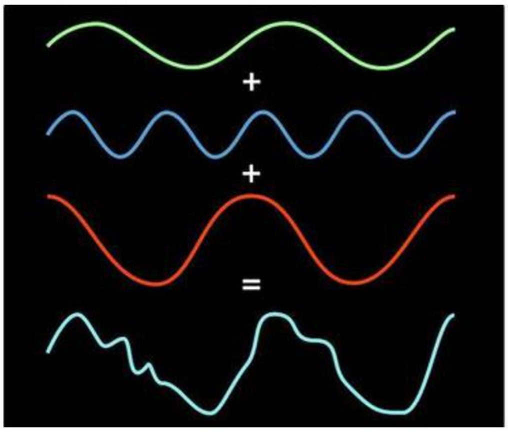


创建混音数据并进行傅里叶变换

"""
C:\sox-14-4-2\sox.exe --null -r 22050 d:\sine_a.wav synth 0.2 sine 400
C:\sox-14-4-2\sox.exe --null -r 22050 d:\sine_b.wav synth 0.2 sine 3000
C:\sox-14-4-2\sox.exe --combine mix --volume 1 sine_b.wav --volume 0.5 sine_a.wav sine_mix.wav
"""

```
plt.figure(num=None, figsize=(12, 8), dpi=80, facecolor='w', edgecolor='k')
plt.subplot(3,2,1)
sample_rate, a = wavfile.read("D:/StudyMaterials/python/python-sklearn/sine_a.wav")
specgram(a, Fs=sample_rate, xextent=(0,30))
plt.xlabel("time")
```

```
plt.ylabel("frequency")
plt.title("400 HZ sine wave")
plt.subplot(3,2,2)
fft_a = abs(fft(a))
specgram(fft_a, Fs=sample_rate)
plt.xlabel("frequency")
plt.ylabel("amplitude")
plt.title("FFT of 400 HZ sine wave")
plt.subplot(3,2,3)
sample_rate, b =
wavfile.read("D:/StudyMaterials/python/python-sklearn/sine_b.wav")
specgram(b, Fs=sample_rate, xextent=(0,30))
plt.xlabel("time")
plt.ylabel("frequency")
plt.title("3000 HZ sine wave")
plt.subplot(3,2,4)
fft_b = abs(fft(b))
specgram(fft_b, Fs=sample_rate)
plt.xlabel("frequency")
plt.ylabel("amplitude")
plt.title("FFT of 3000 HZ sine wave")
plt.subplot(3,2,5)
sample_rate, c =
wavfile.read("D:/StudyMaterials/python/python-sklearn/sine_mix.wav")
specgram(c, Fs=sample_rate, xextent=(0,30))
plt.xlabel("time")
plt.ylabel("frequency")
plt.title("Mixed sine wave")
plt.subplot(3,2,6)
fft_c = abs(fft(c))
specgram(fft_c, Fs=sample_rate)
plt.xlabel("frequency")
plt.ylabel("amplitude")
plt.title("FFT of mixed sine wave")
plt.tight_layout(pad=0.4, w_pad=0, h_pad=1.0)
plt.savefig("D:/compare.sina.wave.png", bbox_inches="tight")
```



对单首音乐进行傅里叶变换

```
plt.figure(num=None, figsize=(9, 6), dpi=80, facecolor='w', edgecolor='k')
sample_rate, X = wavfile.read("D:/genres/jazz/converted/jazz.00000.au.wav")
plt.subplot(2, 1, 1)
specgram(X, Fs=sample_rate, xextent=(0, 30))
plt.xlabel("time")
plt.ylabel("frequency")
plt.subplot(2, 1, 2)
fft_X = abs(fft(X))
specgram(fft_X, Fs=sample_rate)
plt.xlabel("frequency")
plt.ylabel("amplitude")
plt.tight_layout(pad=0.4, w_pad=0, h_pad=1.0)
plt.savefig("D:/jazz.00000.au.wav.fft.png", bbox_inches="tight")
```

数据预处理并保存

准备音乐数据，把音乐文件一个个的去使用傅里叶变化，并且把结果存储起来

```
def create_fft(g, n):
    rad = "d:/genres/" + g + "/converted/" + g + "." + str(n).zfill(5) + ".au.wav"
    sample_rate, X = wavfile.read(rad)
    fft_features = abs(fft(X)[:1000])
    sad = "d:/trainset/" + g + "." + str(n).zfill(5) + ".fft"
    np.save(sad, fft_features)
```

```
genre_list = ["classical", "jazz", "country", "pop", "rock", "metal"]
for g in genre_list:
    for n in range(100):
        create_fft(g, n)
```

classical.00000.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00001.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00002.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00003.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00004.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00005.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00006.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00007.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00008.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00009.fft.npy	2015/8/7 11:42	NPY 文件	8 KB
classical.00010.fft.npy	2015/8/7 11:42	NPY 文件	8 KB

构建训练集

```
genre_list = ["classical", "jazz", "country", "pop", "rock", "metal"]
X = []
Y = []
for g in genre_list:
    for n in range(100):
        rad = "d:/trainset/" + g + "." + str(n).zfill(5) + ".fft" + ".npy"
        fft_features = np.load(rad)
        X.append(fft_features)
        Y.append(genre_list.index(g))

X = np.array(X)
Y = np.array(Y)
```

训练模型并保存

```
model = LogisticRegression()
model.fit(X, Y)

output = open('data.pkl', 'wb')
```

```
pickle.dump(model, output)
output.close()
```

加载模型进行测试

```
pkl_file = open('data.pkl', 'rb')
model_load = pickle.load(pkl_file)
pprint.pprint(model_load)
pkl_file.close()

print('Starting read wavfile...')
# sample_rate, test_images = wavfile.read("d:/trainset/sample/heibao-wudizirong-remix.wav")
# sample_rate, test_images = wavfile.read("d:/trainset/sample/small-apple.wav")
sample_rate, test = wavfile.read("d:/trainset/sample/xiaobang.wav")
print(test.shape)
# 9320535*2
test = np.reshape(test, (1, -1))[0]
print(test.shape)
print(sample_rate, test)
testdata_fft_features = abs(fft(test)[:1000])
print(sample_rate, testdata_fft_features, len(testdata_fft_features))
type_index = model_load.predict([testdata_fft_features])[0]
print(model_load.predict([testdata_fft_features]))
print(model_load.predict_proba([testdata_fft_features]))
print(type_index)
print(genre_list[type_index])
```