线性模型(Linear Model)

对于给定样本$\vec{\mathbf{x}}$，假定其有n维特征，则，$\vec{\mathbf{x}} = (x_1, x_2, x_3, \ldots, x_n)^{\mathsf{T}}$。线性模型可以表示为：

$$\hat{f}(\vec{\mathbf{x}}) = \vec{\mathbf{w}}^{\mathsf{T}}\vec{\mathbf{x}} + b$$

其中$\vec{w} = (w_1, w_2, w_3, \ldots, w_n)^{\mathsf{T}}$是权重参数，表征了$\vec{x}$中每个维度特征的重要性；$b$是偏置项；$\hat{f}(\vec{\mathbf{x}})$是预测值。

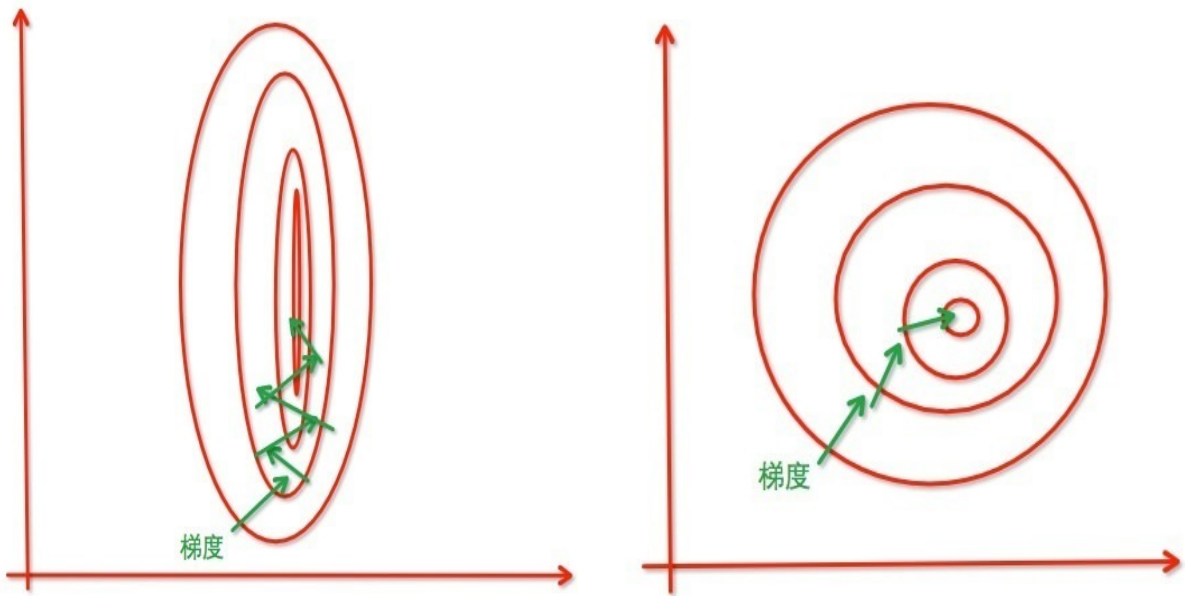
线性回归就是根据给定数据集$\{(\overrightarrow{\mathrm{x}_1}, \mathrm{y}_1), (\overrightarrow{\mathrm{x}_2}, \mathrm{y}_2), \ldots, (\overrightarrow{\mathrm{x}_n}, y_n)\}$，来确定权重系数$\vec{\mathbf{w}}$和偏置项$b$。其中，$(y_1, y_2, \ldots, y_n)$是每个样本对应的真实值。通过最小化真实值和预测值之间的误差来确定最优的权重系数和偏置项。可以构造损失函数，通过最小化损失函数来求取权重系数和偏置项，如常用的均方误差(mean squared error)损失函数：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

可以用梯度下降法来求解上述最优化问题（注意：此处不是随机梯度下降法，因为均方误差损失函数是凸函数）。在求解最优化问题的时候要注意特征归一化，这在许多机器学习模型中都需要注意的问题。

特征归一化的优点：

1. 归一化后加快了梯度下降求最优解的速度。例如对于不在同一尺度的两个特征，在进行优化时梯度等高线是椭圆形的，导致在梯度下降时，优化的方向沿着垂直等高线的方向走之字路线，也就是说变量的权重在优化的过程会出现震荡，而归一化之后梯度等高线是圆形，梯度的方向指向圆心，迭代就会很快。



2. 归一化有可能提高精度。如果一个特征值域范围非常大，那么距离计算就主要取决于这个特征，如果这时值域范围小的特征更重要，就会导致精度损失。归一化可以让每个维度的特征对结果做出的贡献相同。

特征归一化的方法： 1. 线性归一化； 2. 标准差归一化； 3. 非线性归一化。

在求解线性回归模型时，特征组合问题也是需要注意的问题，比如房子的长度和宽度作为两个特征参与模型的构造，不如把其乘积面积作为一个特征来进行求解，这样在特征选择上就起到了降维的作用。

广义线性模型(Generalized Linear Model)

对于可导函数h，令$h(y) = \vec{\mathbf{w}}^{\mathsf{T}}\vec{\mathbf{x}} + b$，就得到了广义线性模型。
例如，对数线性回归：

$$ln(y) = \vec{\mathbf{w}}^{\mathsf{T}}\vec{\mathbf{x}} + b$$

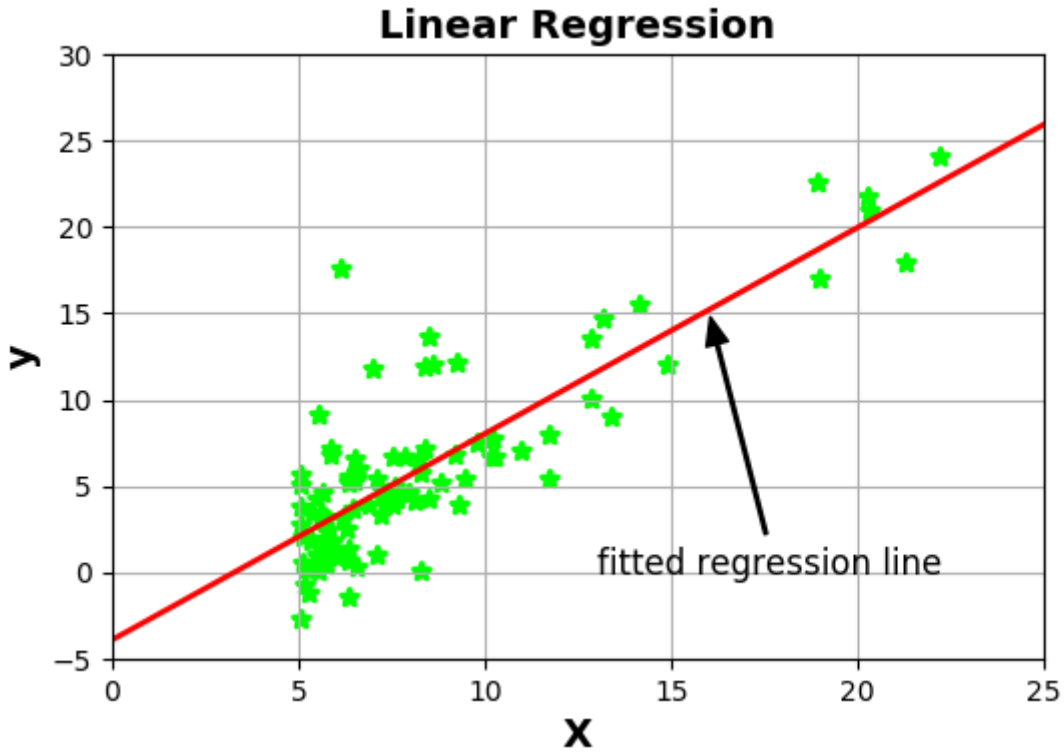
sklearn实现线性回归：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet

data = np.loadtxt('LinearRegressionData.txt', delimiter=',')
X = data[:,0]
X = X.reshape(-1,1)
y = data[:,1]

#sklearn中线性回归有多种正则化方法，下面的LinearRegression可换成Lasso, Ridge, ElasticNet
regr = LinearRegression()
regr.fit(X,y)

#显示结果
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_xlim(0,25)
ax.set_ylim(-5,30)
ax.set_title('Linear Regression',fontsize=14,fontweight='bold')
ax.set_xlabel('X',fontsize=14,fontweight='bold')
ax.set_ylabel('y',fontsize=14,fontweight='bold')
ax.annotate('fitted regression line', xy=(16, 16*regr.coef_+regr.intercept_),
            xytext=(13,1.5), arrowprops=dict(facecolor='black',shrink=0.03,width=1,headwidth=
            horizontalalignment='left',verticalalignment='top',fontsize=12)
ax.grid()
plt.scatter(X,y,color='lime',marker='*',linewidth=2)
fit_x = np.linspace(0,25)
fit_y = regr.coef_ * fit_x + regr.intercept_
plt.plot(fit_x, fit_y,color='r',linewidth=2)
plt.show()
```



sklearn实现逻辑回归：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

data = np.loadtxt('LogisticRegressionData.txt',delimiter=',')
X = data[:,0:2]
X = X.reshape(-1,2)
y = data[:,2]

#分别获取两类样本的索引
negative = data[:,2] == 0
positive = data[:,2] == 1

#sklearn中逻辑回归除了线性模型中的LogisticRegression外还可以用判别分析中的LinearDiscriminantAnalys
#即下面的LogisticRegression可换位LinearDiscriminantAnalysis
regr = LogisticRegression()
regr.fit(X,y)

#显示结果
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_xlim(25,105)
ax.set_ylim(25,105)
ax.set_xlabel("X",fontsize=14,fontweight='bold')
ax.set_ylabel("y",fontsize=14,fontweight='bold')
ax.set_title('Logistic Regression',fontsize=14,fontweight='bold')
ax.grid()
plt.scatter(data[negative][:,0], data[negative][:,1],marker='x',c='r')
plt.scatter(data[positive][:,0], data[positive][:,1],marker='+',c='b')
```

```
x_min,x_max = X[:,0].min(),X[:,0].max()
y_min,y_max = X[:,1].min(),X[:,1].max()
h = 0.01
xx,yy = np.meshgrid(np.arange(x_min,x_max,h), np.arange(y_min,y_max,h))
Z = regr.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx,yy,Z,[0.5],colors='g')
plt.savefig('Logistic Regression',dpi=100)
plt.show()
```