机器学习笔记

冯哲*

西安电子科技大学计算机学院

2018年6月

目录

第一部分　监督学习 **3**

1　分类 **3**

1.1　逻辑回归 3

1.1.1　指数分布簇 3

1.1.2　广义线性模型 4

1.1.3　理论：利用梯度上升做逻辑回归 4

1.1.4　实现：利用梯度上升做逻辑回归 6

1.1.5　实战：利用逻辑回归，从疝气病症预测病马的死亡率 10

1.1.6　小结 14

1.2　高斯判别算法 15

1.2.1　生成学习算法 15

1.2.2　多项正太分布 15

1.2.3　高斯判别分析模型 18

*电子邮件：1194585271@qq.com

1.2.4 GDA最大似然估计最佳参数详细推导 19
1.2.5 高斯判别分析与逻辑回归对比 21
1.3 朴素贝叶斯算法 24
1.3.1 概率论基础 24
1.3.2 算法数学原理流程 25
1.3.3 多元变量伯努利事件模型（词集模型） 25
1.3.4 多项式事件模型（词袋模型） 29
1.3.5 拉普拉斯平滑 29
1.3.6 实战：python实现朴素贝叶斯分类器分类文本 30
1.3.7 示例：使用朴素贝叶斯过滤垃圾邮件 32
1.4 支持向量机 35
1.4.1 逻辑回归与支持向量机 35
1.4.2 函数间隔与几何间隔 36
1.4.3 最优间隔分类器的产生 37
1.4.4 拉格朗日对偶 38
1.4.5 利用对偶问题求解最优间隔分类器 40

2 回归 **41**

2.1 线性回归 41
2.1.1 对高斯分布进行广义线性建模 41
2.1.2 最小二乘法的概率解释：最大似然估计 41
2.1.3 正规方程法找最佳回归系数 43
2.1.4 实战：利用线性回归寻找最佳拟合直线 44
2.1.5 利用局部加权线性回归寻找最佳拟合直线 47
2.1.6 示例：利用线性回归预测鲍鱼年龄 50

第一部分　监督学习

监督学习是从标记的训练数据来推断一个功能的机器学习任务。训练数据包括一套训练示例。在监督学习中，每个实例都是由一个输入对象（通常为矢量）和一个期望的输出值（也称为监督信号）组成。监督学习算法是分析该训练数据，并产生一个推断的功能，其可以用于映射出新的实例。一个最佳的方案将允许该算法来正确地决定那些看不见的实例的类标签。这就要求学习算法是在一种“合理”的方式从一种从训练数据到看不见的情况下形成。

当采用了监督学习后，进一步确定目标变量若为离散值（标称量），则采用分类算法进行学习；若为连续值，则采用回归算法进行学习。

1　分类

监督学习的分类算法有很多，最简单的k-邻近算法，还有决策树，朴素贝叶斯，逻辑回归，支持向量机，Adaboost算法等。这些算法都有其特性，或多或少的公式推理，我都要去熟悉，学习，加实战。

1.1　逻辑回归

逻辑回归（Logistics Regression）是一种通过画出训练样本的决策边界，解决某种数据拟合二分类问题的有效途径。对于怎样画出决策边界则为此算法核心。我从逻辑函数（Logistic Function）的由来入手，学习了指数分布族，广义线性模型。

1.1.1　指数分布簇

介绍指数分布族（Exponential Family）为下一小节的广义线性模型（GLM）做铺垫。指数分布簇抽象统一形式为：

$$p(y;\eta) = b(y)exp(\eta^T T(y) - a(\eta)) \tag{1}$$

其中参数η是自然常量；$T(y)$充分统计量（通常情况下$T(y) = y$）；$a(\eta)$为对数划分函数；给定三个函数$a(\eta), b(y), T(y)$就能确定一组概率分布；如在给定η值，便可以确定唯一一个概率分布。那么有哪些常见分布属于指数分布族：

1. 正态分布
2. 二项分布
3. 多项式分布

1.1.2 广义线性模型

广义线性模型通过对指数分布族内分布进行建模得到响应函数从而得到目标函数，再通过某种最优化的算法（牛顿法/梯度上升法）得到最优系数，从而得到目标连接模型。构建广义线性模型的步骤为，也就是广义线性模型的形式化定义为：

1. $y|x;\theta \sim ExponentialFamily(\eta)$
   即就是在给定x,θ后,y满足的概率分布是以η为自然常数的指数分布族的一员。
2. 给定x，输出 $h(x) = E[T(y)|x]$ 即目标得到y的期望值。
3. 最后令 $\eta = \theta^T x$ （线性模型）得到目标连接函数。

现在以Bernoulli分布为例，构建广义线性模型，得到逻辑函数（也叫Sigmoid函数）。现知道二项分布的概率密度函数为：

$$\begin{aligned} p(y;\phi) &= \phi^y(1-\phi)^{1-y} \\ &= exp\left(yln\phi + (1-y)ln(1-\phi)\right) \\ &= exp\left((ln(\frac{\phi}{1-\phi}))y + ln(1-\phi)\right) \end{aligned}$$

对照指数分布族的抽象式(1)得到（这里 η 是标量）：

$$\begin{aligned} \eta &= ln(\frac{\phi}{1-\phi}) \\ T(y) &= y \\ a(\eta) &= -ln(1-\phi) \\ &= ln(1+e^{\eta}) \\ b(y) &= 1 \end{aligned}$$

得到关联函数：

$$\begin{aligned} h_\theta(x) &= E(y|x;\theta) \\ &= \phi \quad (0-1\ distribution's\ mean) \\ &= \frac{1}{1+e^{-\eta}} \quad (linkfunction) \\ &= \frac{1}{1+e^{-\theta^T x}} \end{aligned}$$

1.1.3 理论：利用梯度上升做逻辑回归

首先知道逻辑函数，所以令：

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

其中，

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

可以看出当 $z \to \infty$ 时，$g(z) \to 1$；当 $z \to -\infty$ 时，$g(z) \to 0$; 再令 $x_0 = 1$,则：

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

$g(z)$ 对z求导：

$$\begin{aligned} g'(z) &= \frac{\partial}{\partial z}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{1+e^{-z}} \cdot (1 - \frac{1}{(1+e^{-z})}) \\ &= g(z)(1-g(z)) \end{aligned} \tag{2}$$

既然逻辑函数值表示概率，则：

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

即就是，

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

之后在对其进行极大似然估计：

$$\begin{aligned} L(\theta) &= p(\overrightarrow{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)})^{1-y^{(i)}} \\ l(\theta) &= lnL(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} lnh(x^{(i)}) + (1-y^{(i)}) ln(1-h(x^{(i)})) \end{aligned}$$

最终为了得到最大的 $l(\theta)$,利用一种最优化算法梯度上升来求取，要求梯度，即求导：

$$\begin{aligned} \frac{\partial}{\partial \theta_j} l(\theta) &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})\frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}) g(\theta^T x)(1-g(\theta^T x))\frac{\partial}{\partial \theta_j}\theta^T x \\ &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x))x_j \\ &= (y - h_\theta(x))x_j \end{aligned}$$

得到梯度后，再利用随机梯度上升（stochastic gradient ascent）迭代更新系数：

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

1.1.4 实现：利用梯度上升做逻辑回归

实战中利用的是梯度上升的最优化算法，逻辑回归就是利用最优化算法来训练出一个非线性函数用于分类。期间核心是利用最优化算法来寻找最佳拟合参数。逻辑回归的优点是计算代价不高，易于理解与实现；其缺点是容易欠拟合，分类精度不高；适用数据类型是数值型与标称型数据。逻辑回归的一般过程为：

1. 收集数据：任意方法收集。
2. 准备数据：最好为结构化数据。
3. 分析数据：任意方法分析数据。
4. 训练数据：大部分时间用来训练数据。
5. 测试算法：测试会很快完成。
6. 使用算法：输入数据，转结构化，进行回归计算，判定类别。

梯度上升找最佳参数 θ

梯度上升算法的迭代公式如下,利用python实现。

$$w := w + \alpha\nabla_w f(w)$$

程序核心函数代码：

```
def sigmoid(inX):
    return 1.0/(1+exp(-inX))

def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)             #convert to NumPy matrix
    labelMat = mat(classLabels).transpose() #convert to NumPy matrix
    m,n = shape(dataMatrix)
    alpha = 0.001
    maxCycles = 600                         #Number of iterations
    weights = ones((n,1))
    for k in range(maxCycles):              #heavy on matrix operations
```

```
        h = sigmoid(dataMatrix*weights)  #matrix mult
        #print(dataMatrix*weights)       #$\eta$
        error = (labelMat - h)           #vector subtraction
        weights = weights + alpha * dataMatrix.transpose()* error #matrix mult
    return weights

def plotBestFit(weights):
    import matplotlib.pyplot as plt
    dataMat,labelMat=loadDataSet()
    dataArr = array(dataMat)
    n = shape(dataArr)[0]
    xcord1 = []; ycord1 = []
    xcord2 = []; ycord2 = []
    for i in range(n):
        if int(labelMat[i])== 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
    ax.scatter(xcord2, ycord2, s=30, c='green')
    x = arange(-3.0, 3.0, 0.1)
    y = (-weights[0]-weights[1]*x)/weights[2]
    ax.plot(x, y)
    plt.xlabel('X1'); plt.ylabel('X2');
    plt.show()
    plt.savefig('LogRegres_GradAscent.eps',dpi=2000)
```

利用测试数据得到初步的分类结果，并可视化表示结果如下图：

分析图1：

分类结果相当不错，从图中看上去只有四个点分类错误，但是知道一共需要600次迭代，也就是说至少要有600次的全训练数据的矩阵乘运算。因此，不能将其作用于真实数据集的主要原因有二：其一时间复杂度缺陷；其二每一次迭代运算都需要作用于全体训练集，不利用扩充训练集。

接下来学习的随机梯度上升算法解决上述两类问题。

随机梯度上升找最佳参数 θ

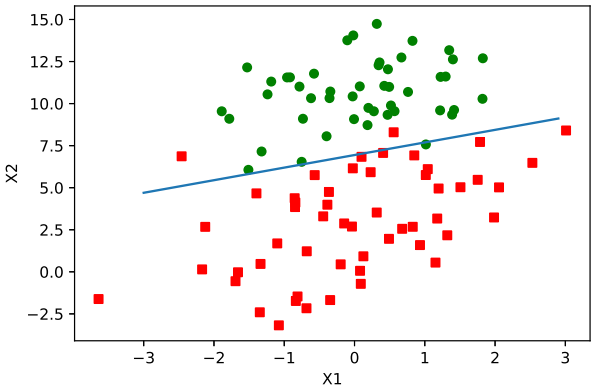

图 1: 利用梯度上升做逻辑回归的分类结果

随机梯度上升算法（Stochastic Gradient Ascent Algorithm）是一个在线学习算法，由于其可以在新样本到来时对分类器进行增量式的更新。与在线学习算法相对应的是，一次处理所有的数据被称为是批处理。

而随机梯度上升的迭代公式为：

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \quad one\ coefficient$$
$$\Longrightarrow w := w + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \quad all\ coefficient$$

程序核心函数代码：

```
def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    # print (m,n)
    alpha = 0.01
    weights = ones(n) #initialize to all ones
    for i in range(m):
        h = sigmoid(sum(dataMatrix[i]*weights))
        error = classLabels[i] - h
        weights = weights + alpha * error * dataMatrix[i] #Non matrix operation
    return weights
```

还是利用之前的测试数据得到初步的分类结果，并可视化表示结果如下图：

首先看回归系数与迭代次数的关系图：

分析：

分类器分错了三分之一的样本，但是只进行了类似于上节批量梯度下降的一次矩阵运算，所以结果并不公平。

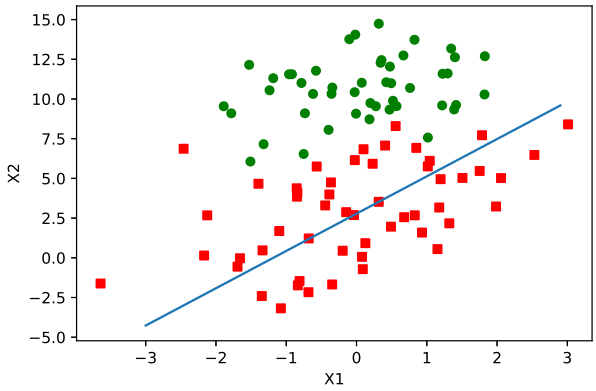

图 2: 利用随机梯度上升做逻辑回归的分类结果，并非最佳分类线

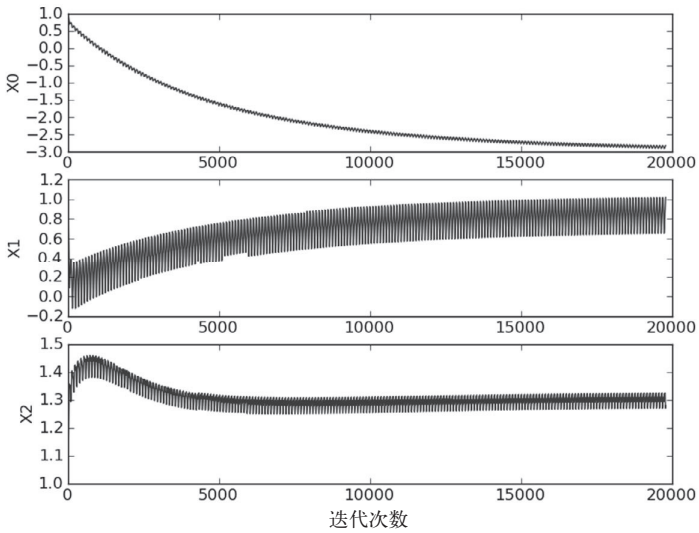

图 3: 回归系数与迭代次数的关系图

此外看到，存在一些不能正确分类的样本点，导致了在每次迭代的时，会引发系数的剧烈变化。于是期望算法能避免来回波动，从而收敛到某个值；以及收敛的速度也应加快。只要对其进行改进，效果则立竿见影。

改进的随机梯度上升找最佳参数 θ

程序核心函数代码:

```
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n) #initialize to all ones
    for j in range(numIter):
        dataIndex = list(range(m))
```

```
    for i in range(m):
        alpha = 4/(1.0+j+i)+0.0001 #alpha decreases with iteration, does not
        randIndex = int(random.uniform(0,len(dataIndex)))#go to 0 because
            of the constant
        h = sigmoid(sum(dataMatrix[randIndex]*weights))
        error = classLabels[randIndex] - h
        weights = weights + alpha * error * dataMatrix[randIndex]
        del(dataIndex[randIndex])
        # print(dataIndex)
return weights
```

还是利用之前的测试数据得到初步的分类结果，并可视化表示结果如下图:

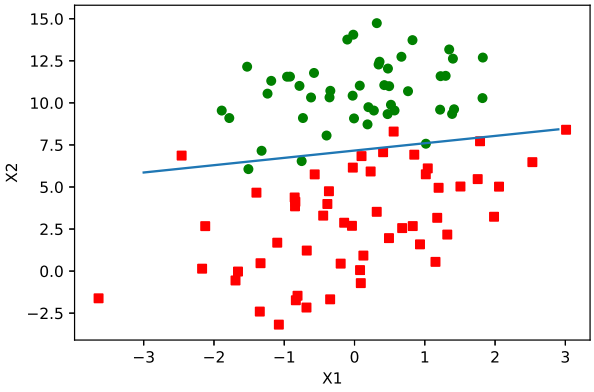

图 4: 利用改进的随机梯度上升做逻辑回归的分类结果

算法参数分析:

1.设置动态步长 α ，缓解上节中数据的波动或者高频波动，比固定的 α 收敛速度更快。当 $j << max(i)$ 时，α 就不是严格下降，避免参数的严格下降也常见于模拟退火算法等其他优化算法当中。

2.训练样本随机选取更新回归系数，为了减少上节的周期性波动。

分析:

改进的随机梯度下降优化算法与批量梯度下降优化算法相比，分类结果的效果差不多，但是所使用的计算量更少，再利用小量的训练集时间差别不大，但是如果处理数以十亿计的训练样本和成千上万的特征时，两者的优越性则显露无疑。

1.1.5 实战：利用逻辑回归，从疝气病症预测病马的死亡率

将使用Logistic回归来预测患有疝病的马的存活问题。这里数据包含368个样本，与28个

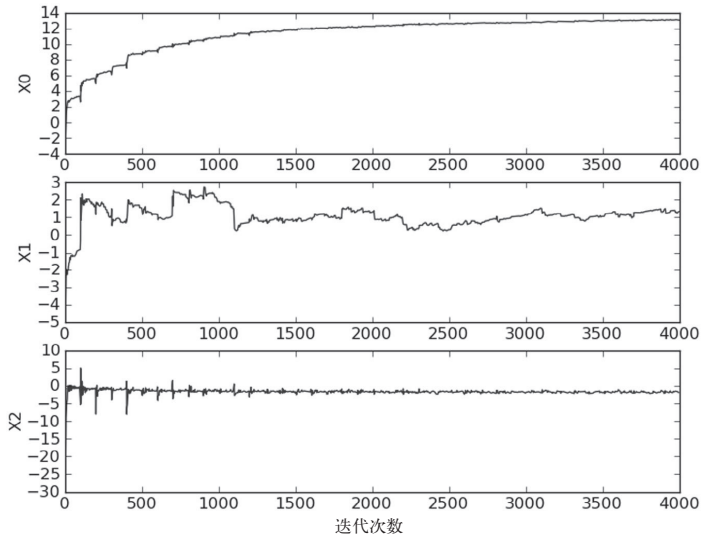

图 5: 改进算法后回归系数与迭代次数关系图

特征；但有的指标比较主观，有的指标难以测量，例如马的疼痛级别。

另外，该数据有30%的数据缺失用0填充，一条样本只有21个feature和1个label。实战的步骤:

1. 收集数据：给定数据文件。
2. 准备数据：用python解析文本文件并填充缺失值。
3. 分析数据：可视化并观察数据。
4. 训练算法：使用优化算法，找到最佳的系数。
5. 测试算法：观测错误率，回退调参，得到更好的回归系数。
6. 使用算法：预测病马死亡率。

准备数据

数据无价，数据中的缺失值，不能轻易的丢弃掉，有时也不能轻易的重新获取，必须采用一些方法来处理数据中的缺失值。这里采用的是使用特殊值来填充，用0填充，其对结果的预测不具有任何倾向性，因为$sigmoid(0) = 0.5$。

1. 使用特征均值填补缺失值。
2. 使用特殊值来填补缺失值，如-1。
3. 忽略有缺失值的样本。

4. 使用相似样本的均值填补缺失值。

5. 使用另外的机器学习算法预测缺失值。

测试算法

将训练集中的数据进行逻辑回归分析，得出回归系数向量。在与特征向量相乘输入到sigmoid函数当中，结果大于0.5，预测为1；否则为0。

程序核心函数代码：

```
def sigmoid(inX):
    if inX<-700:
        inX = -700   #avoid exp_function overflow
    return 1.0/(1+exp(-inX))

def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n) #initialize to all ones
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.0001 #apha decreases with iteration, does not
            randIndex = int(random.uniform(0,len(dataIndex)))#go to 0 because
                of the constant
            h = sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            weights = weights + alpha * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    # print("weight=",weights)
    return weights

def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0

def colicTest():
    frTrain = open('horseColicTraining.txt'); frTest =
        open('horseColicTest.txt')
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
```

```
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    trainWeights = stocGradAscent1(array(trainingSet), trainingLabels, 1000)
    errorCount = 0; numTestVec = 0.0
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(array(lineArr), trainWeights))!=
            int(currLine[21]):
            errorCount += 1
    errorRate = (float(errorCount)/numTestVec)
    print ("the error rate of this test is: %f" % errorRate)
    return errorRate

def multiTest():
    numTests = 10; errorSum=0.0
    for k in range(numTests):
        errorSum += colicTest()
    print ("after %d iterations the average error rate is: %f" % (numTests,
        errorSum/float(numTests)))
```

预测结果

```
the error rate of this test is: 0.343284
the error rate of this test is: 0.328358
the error rate of this test is: 0.283582
the error rate of this test is: 0.298507
the error rate of this test is: 0.417910
the error rate of this test is: 0.417910
the error rate of this test is: 0.298507
the error rate of this test is: 0.358209
the error rate of this test is: 0.313433
the error rate of this test is: 0.343284
after 10 iterations the average error rate is: 0.340299
```

利用改进的随机梯度下降算法迭代求得回归系数，最终在作用于测试集，得到预测的平均错误率为34%。这个结果相对不错，因为有30%的数据缺失。事实上错误率还可以通过调参往下降。

1.1.6 小结

逻辑回归这一章从逻辑函数由来入手，学习了指数分布族，广义线性模型。知道逻辑函数是由bernoulli分布广义线性建模的结果。还给了必要的理论证明。

逻辑回归目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优化算法来完成。最常用的就是梯度上升算法，而梯度上升算法可以改进为随机梯度上升算法。

随机梯度上升算法的效果相当，但占用更少的计算机资源。此外，随机梯度上升是一个在线算法，它可以在新的数据到来时就可以完成更新，而不需要重新读取整个数据集来进行批处理运算。

机器学习的一个重要的问题就是如何处理缺失数据。这个问题没有标准答案，取决于实际应用中的需求。现有的解决方案都各自有优缺点。

1.2 高斯判别算法

1.2.1 生成学习算法

逻辑回归或者感知机算法是在给定 x 的情况下直接对 $p(y|x;\theta)$ 进行建模。其都是试图做分类边界，来进行分类；那换个思路就是对各个分类目分别建立模型，最终将新样本输入到各个分类目模型当中去试图分类。

判别学习算法

判别学习算法（discriminative learning algorithm）：直接学习 $p(y|x)$ 或者是从输入直接映射到输出的方法；逻辑回归与感知机算法就是这一类算法的代表。

生成学习算法

生成学习算法（generative learning algorithm）：对 $p(x|y)$ (也包括 $p(y)$)进行建模。建模方式:

输出两类: $y \in \{0,1\}$

$p(x|y=0)$: 对0类特征进行建模

$p(x|y=1)$: 对1类特征进行建模

完成对 $p(x|y)$,以及 $p(y)$ 的建模后。利用 Bayes 公式求得再给定 x 情况下 y 的概率，如下:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

$$p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$$

最后再对 $p(x,y)$ 进行最大似然估计，得到最佳参数。

最终将新样本输入到各个模型中得到概率值，判定类目。

事实上。可以不用完全算出概率值，比较不同类目输出结果大小即可。

$$\begin{aligned} arg\ \max_y\ p(y|x) &= arg\ \max_y \frac{p(x|y)p(y)}{p(x)} \\ &= arg\ \max_y\ p(x|y)p(y) \end{aligned}$$

常见的生成模型有：高斯判别分析（特征值为连续），隐马尔可夫模型HMM，朴素贝叶斯模型（特征值为离散），高斯混合模型GMM，LDA等。

1.2.2 多项正太分布

n维多项分布也称多项高斯分布，均值向量 $\mu \in R^n$,协方差矩阵 $\Sigma \in R^{n\times n}$,记为 $N(\mu,\Sigma)$ ，其概率密度表示为:

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp\Big(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\Big)$$

其中 $|\Sigma|$ 表示矩阵的行列式（determinant）.
均值为 $E[X] = \int_x xp(x,;\mu,\Sigma)dx = \mu$.
对于多元随机变量Z,

$$Cov(Z) = E[(Z - E[Z])(Z - E[Z])^T] = E[ZZ^T] - (E[Z])(E[Z])^T.$$

则,

$$If:\ X \sim N(\mu,\Sigma)$$

$$So:\ Cov(X) = \Sigma$$

二维正太分布图： $\mu = [0,0], \Sigma = I$（单位阵）

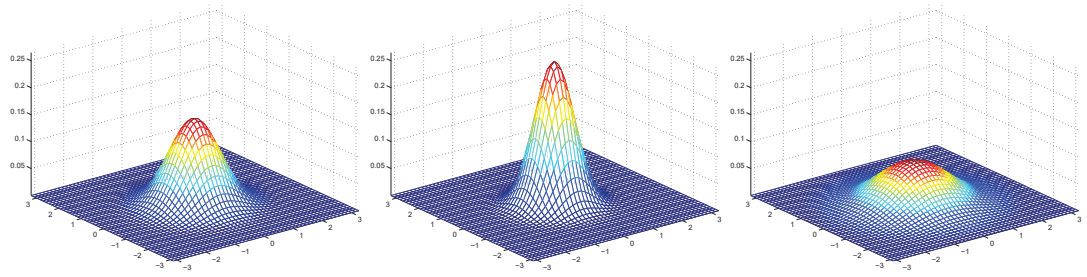

图 6: $\Sigma = I$ $\Sigma = 0.6I$ $\Sigma = 2I$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad 0.6I = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix} \quad 2I = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

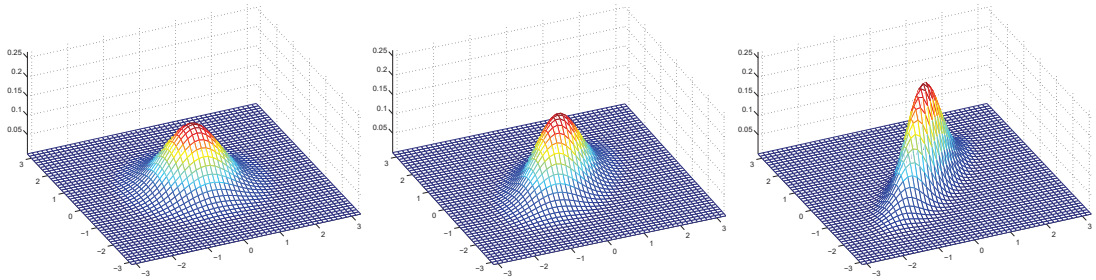

图 7: $\Sigma = I$ $\Sigma = I_1$ $\Sigma = I_2$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad I_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \quad I_2 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

上图的等高线形式更能清晰可见：

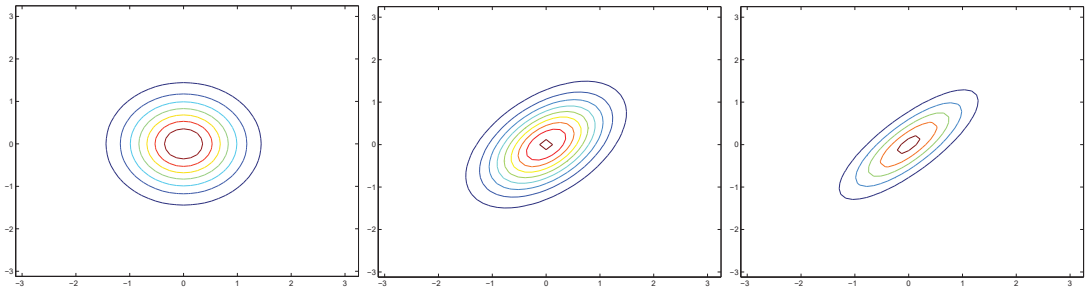

图 8: $\Sigma = I$ $\Sigma = I_1$ $\Sigma = I_2$

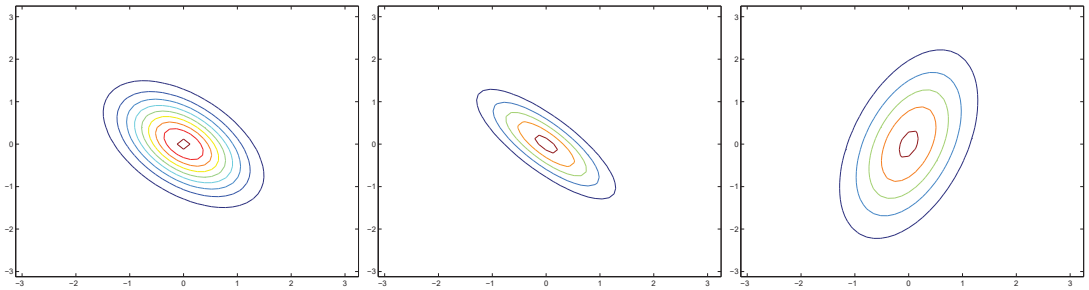

图 9: $\Sigma = I_3$ $\Sigma = I_4$ $\Sigma = I_5$

$$I_3 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix} \quad I_4 = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix} \quad I_5 = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

变均值，而不变协方差：$\Sigma = I$

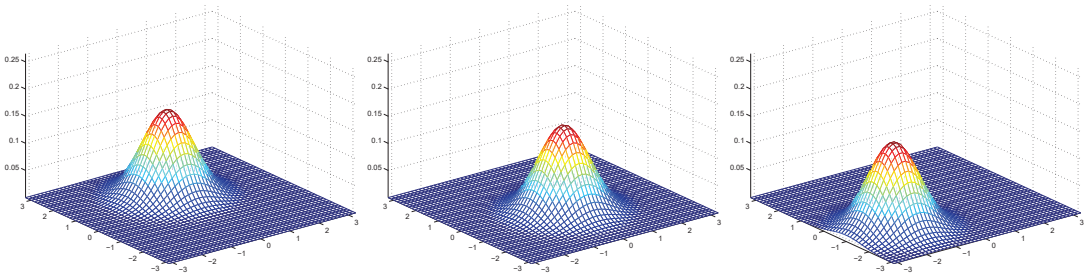

图 10: $\mu = \mu_1$ $\mu = \mu_2$ $\mu = \mu_3$

$$\mu_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} -1 \\ -0.5 \end{bmatrix}$$

因此， μ 决定中心位置，而 Σ 决定投影椭圆的朝向和大小。

1.2.3 高斯判别分析模型

现有一个分类问题，训练集的特征值X都是随机的连续值，便可利用高斯判别模型（The Gaussian Discriminant Analysis model）,假设:

$$
\begin{aligned}
y &\sim Bernoulli(\phi) \\
x|y=0 &\sim N(\mu_0, \Sigma) \\
x|y=1 &\sim N(\mu_1, \Sigma)
\end{aligned}
$$

因此就有:

$$
\begin{aligned}
p(y) &= \phi^y(1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp\bigg(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\bigg) \\
p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp\bigg(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\bigg)
\end{aligned}
$$

最大似然估计 $l(\phi, \mu_0, \mu_1, \Sigma)$ 得到最佳参数值:

$$
\begin{aligned}
l(\phi, \mu_0, \mu_1, \Sigma) &= log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\
&= log \prod_{i=1}^{m} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi)
\end{aligned}
$$

$$
\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^{m} y^i = 1 \\
\mu_0 &= \frac{\sum_{i=1}^{m} \mathbf{1}\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}\{y^{(i)}=0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} \mathbf{1}\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}\{y^{(i)}=1\}} \\
\Sigma &= \frac{1}{m}\sum_{i=1}^{m} (x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T
\end{aligned}
$$

理解公式(似然估计的结果非常简洁下节给出推导过程):

ϕ : 训练集中分类结果为1所占的比例。

$\mu_0 : y=0$类样本中的特征均值。

$\mu_1 : y=1$类样本中的特征均值。

Σ: 是样本特征方差均值。

通过上述的理论及描述，可以得到下面图像:

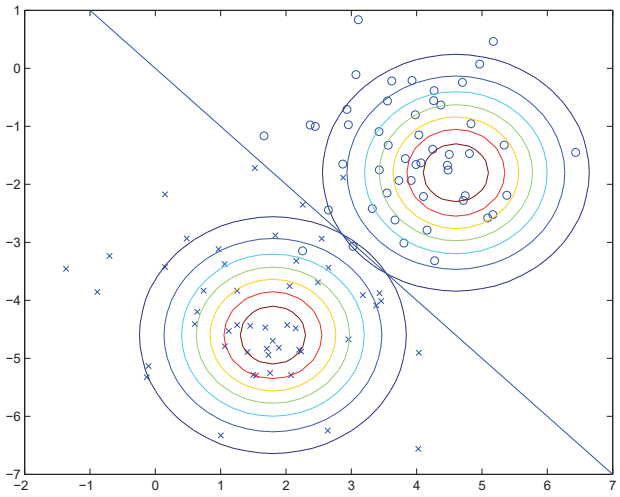

图 11: 高斯判别算法分类结果

分析:

直线两边的y值不同，但协方差矩阵相同，因此形状相同。 μ 值不同，所以位置不同。

1.2.4 GDA最大似然估计最佳参数详细推导

对数似然函数:

$$
\begin{aligned}
l(\phi,\mu_0,\mu_1,\Sigma) &= log\prod_{i=1}^{m}p(x^{(i)},y^{(i)}) = log\prod_{i=1}^{m}p(x^{(i)}|y^{(i)})p(y^{(i)}) = \sum_{i=1}^{m}log\ p(x^{(i)}|y^{(i)}) + log\ p(y^{(i)}) \\
&= \sum_{i=1}^{m}log\ (p(x^{(i)}|y^{(i)}=0)^{1-y^{(i)}}\cdot p(x^{(i)}|y^{(i)}=1)^{y^{(i)}}) + \sum_{i=1}^{m}log\ p(y^{(i)}) \\
&= \sum_{i=1}^{m}(1-y^{(i)})log\ (p(x^{(i)}|y^{(i)}=0) + \sum_{i=1}^{m}y^{(i)}log\ (p(x^{(i)}|y^{(i)}=1) + \sum_{i=1}^{m}log\ p(y^{(i)})
\end{aligned}
$$

注意此函数分为三个部分，第一部分只与 μ_0 有关；第二部分只与 μ_1 有关；第三部分只 ϕ 有关。最大化该函数，1.首先先求 ϕ ，则求第三部分的偏导。

$$
\begin{aligned}
\frac{\partial l(\phi,\mu_0,\mu_1,\Sigma)}{\partial\phi} &= \frac{\sum_{i=1}^{m}log\ p(y^{(i)})}{\partial\phi} \\
&= \frac{\partial\sum_{i=1}^{m}log\ \phi^{y^{(i)}}(1-\phi)^{(1-y^{(i)})}}{\partial\phi} \\
&= \frac{\partial\sum_{i=1}^{m}y^{(i)}log\ \phi + (1-y^{(i)})log\ (1-\phi)}{\partial\phi} \\
&= \sum_{i=1}^{m}(y^{(i)}\frac{1}{\phi} - (1-y^{(i)})\frac{1}{1-\phi}) \\
&= \sum_{i=1}^{m}(I(y^{(1)}=1)\frac{1}{\phi} - I(y^{(i)}=0)\frac{1}{1-\phi})
\end{aligned}
$$

令其为零，求得 ϕ，（其中 I 是指示函数），

$$\phi = \frac{\sum_{i=1}^{m} I(y^{(i)} = 1)}{m}$$

2.同样地，对 μ_0 求偏导，

$$\begin{aligned}
\frac{\partial l(\phi, \mu_0, \mu_1, \Sigma)}{\partial \mu_0} &= \frac{\partial \sum_{i=1}^{m} (1 - y^{(i)}) log\, p(x^{(i)}|y^{(i)} = 0)}{\partial \mu_0} \\
&= \frac{\partial \sum_{i=1}^{m} (1 - y^{(i)}) (log \frac{1}{\sqrt{(2\pi)^2 |\Sigma|}} - \frac{1}{2}(x^{(i)} - \mu_0)^T \Sigma^{-1} (x^{(i)} - \mu_0))}{\partial \mu_0} \\
&= \sum_{i=1}^{m} (1 - y^{(i)}) \Sigma^{-1} (x^{(i)} - \mu_0) \\
&= \sum_{i=1}^{m} I(y^{(i)} = 0) \Sigma^{-1} (x^{(i)} - \mu_0)
\end{aligned}$$

令其为0，得，

$$\mu_0 = \frac{\sum_{i=1}^{m} I\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} I\{y^{(i)} = 0\}}$$

3.同理得 μ_1 ，

$$\mu_1 = \frac{\sum_{i=1}^{m} I\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} I\{y^{(i)} = 1\}}$$

4.对 Σ 求偏导以求 Σ (先改写初式前两部分)，

$$\begin{aligned}
&\sum_{i=1}^{m} (1 - y^{(i)}) log\, p(x^{(i)}|y^{(i)=0}) + \sum_{i=1}^{m} y^{(i)} log\, p(x^{(i)}|y^{(i)=1}) \\
=&\sum_{i=1}^{m} (1 - y^{(i)}) (log \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} - \frac{1}{2}(x^{(i)} - \mu_0)^T \Sigma^{-1} (x^{(i)} - \mu_0)) + \\
&\sum_{i=1}^{m} y^{(i)} (log \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} - \frac{1}{2}(x^{(i)} - \mu_1)^T \Sigma^{-1} (x^{(i)} - \mu_1)) \\
=&\sum_{i=1}^{m} log \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} - \frac{1}{2} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})^T \Sigma^{-1} (x^{(i)} - \mu_{y^{(i)}}) \\
=&\sum_{i=1}^{m} (-\frac{n}{2} log(2\pi) - \frac{1}{2} log(|\Sigma|)) - \frac{1}{2} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})^T (x^{(i)} - \mu_{y^{(i)}})
\end{aligned}$$

进而有，

$$\begin{aligned}
\frac{\partial l(\phi, \mu_0, \mu_1, \Sigma)}{\partial \Sigma} &= -\frac{1}{2} \sum_{i=1}^{m} (\frac{1}{|\Sigma|} |\Sigma| \Sigma^{-1}) - \frac{1}{2} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}}) (x^{(i)} - \mu_{y^{(i)}})^T \frac{\partial \Sigma^{-1}}{\partial \Sigma} \\
&= -\frac{m}{2} \Sigma^{-1} - \frac{1}{2} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}}) (x^{(i)} - \mu_{y^{(i)}})^T (-\Sigma^{-2})
\end{aligned}$$

这里用到的公式有，

$$\frac{\partial|\Sigma|}{\partial\Sigma} = |\Sigma|\Sigma^{-1}$$
$$\frac{\partial\Sigma^{-1}}{\partial\Sigma} = -\Sigma^{-2}$$

令其为零，得，

$$\Sigma = \frac{1}{m}\sum_{i=0}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

将参数全部求出后，可以看出公式推导相当复杂，但是结果是非常精简。要判断一个新样本x时，可分别使用贝叶斯求出 $p(y=0|x)$ 和 $p(y=1|x)$ ，取概率更大的那个类。

实际计算时，我们只需要比大小，那么贝叶斯公式中分母项可以不计算，由于2个高斯函数协方差矩阵相同，则高斯分布前面那相同部分也可以忽略。实际上，GDA算法也是一个线性分类器，根据上面推导可以知道，GDA的分界线(面)的方程为：

$$(1-\phi)exp((x-\mu_0)^T\Sigma^{-1}(x-\mu_0)) = \phi exp((x-\mu_1)^T\Sigma(x-\mu_1))$$

取对数展开后化解，可得：

$$2x^T\Sigma^{-1}(\mu_1-\mu_0) = \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0 + log\phi - log(1-\phi)$$

若，

$$A = 2\Sigma^{-1}(\mu_1-\mu_0) = (a_1, a_2, \ldots, a_n)$$
$$B = \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0 + log\phi - log(1-\phi)$$

则，

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$$

这就是GDA算法的线性分界面。

1.2.5 高斯判别分析与逻辑回归对比

似然公式对比：

GDA:

$$\begin{aligned} l(\phi,\mu_0,\mu_1,\Sigma) &= log\prod_{i=1}^{m}p(x^{(i)},y^{(i)};\phi,\mu_0,\mu_1,\Sigma) \\ &= log\prod_{i=1}^{m}p(x^{(i)}|y^{(i)};\mu_0,\mu_1,\Sigma)p(y^{(i)};\phi) \end{aligned}$$

LR:

$$\begin{aligned} L(\theta) &= p(\overrightarrow{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)})^{1-y^{(i)}} \\ l(\theta) &= lnL(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} lnh(x^{(i)}) + (1-y^{(i)}) ln(1-h(x^{(i)})) \end{aligned}$$

结论一:

如果 $x|y \sim Gaussian$ 即其服从正太分布，那么它的后验公式 $p(y=1|x)$ 就是逻辑函数(sigmoid function)。

证明:

由贝叶斯公式可知:

$$\begin{aligned} p(y=1|x) &= \frac{p(x|y=1)p(y=1)}{p(x)} \\ &= \frac{N(\mu_1,\Sigma)\phi}{N(\mu_0,\Sigma)(1-\phi)+N(\mu_1,\Sigma)\phi} \\ &= 1/(1+\frac{N(\mu_0,\Sigma)}{N(\mu_1,\Sigma)}\frac{1-\phi}{\phi}) \end{aligned}$$

而:

$$\begin{aligned} \frac{N(\mu_0,\Sigma)}{N(\mu_1,\Sigma)} &= exp\{(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) - (x-\mu_1)^T\Sigma^{-1}(x-\mu)\} \\ &= exp\{2(\mu_1-\mu_0)^T\Sigma^{-1}x + (\mu_0^T\Sigma\mu_0 - \mu_1^T\Sigma\mu_1)\} \end{aligned}$$

那么，令:

$$\begin{aligned} 2\Sigma^{-1}(\mu_1-\mu_0) &= (\theta_1,\theta_2,.....,\theta_n)^T \\ \theta_0 &= \mu_0^T\Sigma\mu_0 - \mu_1^T\Sigma\mu_1 + log\frac{1-\phi}{\phi} \end{aligned}$$

则:

$$p(y=1|x) = \frac{1}{1+exp(\theta_0+\theta_1x_1+\theta_2x_2+...+\theta_nx_n)}$$

结论一得证。

结论二:

$$\begin{cases} \dot{x}|y=1 \sim Possion(\lambda_1) \\ x|y=0 \sim Possion(\lambda_0) \end{cases}$$

$$\Rightarrow p(y=1|x) \rightarrow logistic function.$$

在推导逻辑回归的时候，我们并没有假设类内样本是服从高斯分布的，因而GDA只是逻辑回归的一个特例，其建立在更强的假设条件下。故两者效果比较：

1.逻辑回归是基于弱假设推导的，则其效果更稳定，适用范围更广。

2.数据服从高斯分布时，GDA效果更好。

3.当训练样本数很大时，根据中心极限定理，数据将无限逼近于高斯分布，则此时GDA的表现效果会非常好。

为何要假设两类内部高斯分布的协方差矩阵相同?

从直观上讲，假设两个类的高斯分布协方差矩阵不同，会更加合理（在混合高斯模型中就是如此假设的），而且可推导出类似上面简洁的结果。

假定两个类有相同协方差矩阵，分析具有以下几点影响：

1.当样本不充分时，使用不同协方差矩阵会导致算法稳定性不够；过少的样本甚至导致协方差矩阵不可逆，那么GDA算法就没法进行

2.使用不同协方差矩阵，最终GDA的分界面不是线性的，同样也推导不出GDA的逻辑回归形式

使用GDA时对训练样本有何要求?

1.正负样本数的比例需要符合其先验概率。若是预先明确知道两类的先验概率，那么可使用此概率来代替GDA计算的先验概率；若是完全不知道，则可以公平地认为先验概率为50%.

2.样本数必须不小于样本特征维数，否则会导致协方差矩阵不可逆，按照前面分析应该是多多益善。

1.3 朴素贝叶斯算法

朴素贝叶斯（naive Bayes）法是基于**贝叶斯定理**与**特征条件独立假设**的分类方法。对于给定的训练数据集，首先基于特征条件独立假设学习输入/输出的联合概率分布；然后基于此模型，对于给定的输入 x，利用贝叶斯定理求出后验概率最大的输出 y。当然，朴素贝叶斯算法与高斯判别法一样也是一种典型的生成学习算法。

1.3.1 概率论基础

贝叶斯学派的思想可以概括为先验概率+数据=后验概率。

条件独立公式

$$P(X,Y) = P(X)P(Y)$$

条件概率公式

$$P(Y|X) = P(X,Y)/P(X)$$

$$P(X|Y) = P(Y,X)/P(Y)$$

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

全概率公式

$$P(X) = \sum_{i=1}^{m} P(X|Y=Y_i)P(Y_i) \ , \ \sum_{i=1}^{m} Y_i = 1$$

贝叶斯公式

$$P(Y_j|X) = \frac{P(X|Y_j)P(Y_j)}{\sum_{i=1}^{m} P(X|Y=Y_i)P(Y_i)}$$

其中，

$P(Y_j)$ 为先验概率；

$P(X|Y_j)$ 为似然函数；

$P(X)$ 为归一化项；

$P(Y_j|X)$ 为后验概率；

那么朴素贝叶斯算法的核心就是，利用最大似然估计，求取 $P(X|Y_j)$,最终比较后验概率大小来判别类别。

1.3.2 算法数学原理流程

利用上述概率论基础，回到我们的数据分析，来构我们的建朴素贝叶斯分类器。假设我们**有m个样本**，每个样本有n个特征，特征输出有两个类别，定义为1,0.（多分类同理）。

$$(x_1^{(1)}, x_2^{(1)}, ..., x_n^{(1)}, y_1), (x_1^{(2)}, x_2^{(2)}, ..., x_n^{(2)}, y_2), ..., (x_1^{(m)}, x_2^{(m)}, ..., x_n^{(m)}, y_m)$$

从这些样本中就可以学习到**朴素贝叶斯的先验分布** $P(Y=1), P(Y=0)$;接着可以学习**条件概率分布** $P(X|Y=0), P(X|Y=1)$;这里就出现了一个问题，多元变量n个维度的条件分布很难求出。所以朴素贝叶斯就用到了一个大胆的**强独立假设（Naive Bayes Assumption）**，样本各个特征之间相互独立则有：

$$\begin{aligned} P(X|Y_j) &= P(x_1, x_2, ..., x_n|Y_j) \\ &= P(x_1|Y_j)P(x_2|Y_j)...P(x_n|Y_j) \\ &= \prod_{i=1}^{n} P(x_i|Y_j) \end{aligned}$$

从上式可以看出，这个很难的条件分布大大的简化了，但是这也可能带来预测的不准确性。你会说如果我的特征之间非常不独立怎么办？如果真是非常不独立的话，那就尽量不要使用朴素贝叶斯模型了，考虑使用其他的分类方法比较好。但是一般情况下，样本的特征之间独立这个条件的确是弱成立的，尤其是数据量非常大的时候。虽然我们牺牲了准确性，但是得到的好处是模型的条件分布的计算大大简化了，这就是贝叶斯模型的选择。

那么现在朴素贝叶斯可可将贝叶斯公式重写为：

$$\begin{aligned} P(Y_j|X) &= \frac{P(X|Y_j)P(Y_j)}{\sum_{i=1}^{m} P(X|Y=Y_i)P(Y_i)} \\ &= \frac{\prod_{i=1}^{n} P(x_i|Y_j)P(Y_j)}{\sum_{i=1}^{m} P(X|Y=Y_i)P(Y_i)} \\ &= \frac{\prod_{i=1}^{n} P(x_i|Y_j)P(Y_j)}{...} \end{aligned}$$

将分母省略实际上，是在朴素贝叶斯算法判别时是比较大小时，各类的后验概率都不用去除以相同的归一化项，可直接用来比较。事实上，此模型还用到了一个假设就是**特征同等重要假设**。

1.3.3 多元变量伯努利事件模型（词集模型）

1.词集模型：

多元变量伯努利事件模型（multi-variate Bernoulli event model）常用于文本分类，也称为**词集模型（set-of-words model）**就是不考虑同一文档中同一特征出现的频率，

只要出现就将其置1。具体的建模过程为：

①模型总词表：（长度为n=5000）

$$Vocabulary = [word_1, word_2, ..., word_{5000}]$$

②文档词向量：（长度为n=5000,与总词表对应，存在文档总词表的单词相应位置置1）

$$WordVector = [0, 0, 1, 0, 1, 0, 0, ...1, 0]$$

③特征为离散二值变化，输出也是二值{0,1}。因此有：

$$\phi_y = p(y = 1)$$
$$\phi_{j|y=0} = p(x_j = 1|y = 1)$$
$$\phi_{j|y=1} = p(x_j = 0|y = 0)$$

④将 y 和 x_j, y 建模成Bernoulli分布（这是朴素贝叶斯最简单的特例之一），

$$p(y) = (\phi_y)^y(1 - \phi_y)^{1-y}$$
$$p(x|y = 0) = \prod_{j=1}^{n} p(x_j|y = 0) = \prod_{j=1}^{n}(\phi_{j|y=0})^{x_j}(1 - \phi_{j|y=0})^{1-x_j}$$
$$p(x|y = 1) = \prod_{j=1}^{n} p(x_j|y = 1) = \prod_{j=1}^{n}(\phi_{j|y=1})^{x_j}(1 - \phi_{j|y=1})^{1-x_j}$$

⑤强独立假设下的似然函数(有m个样本)：

$$L(\phi_y, \phi_{i|y=1}, \phi_{i|y=0}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi_y, \phi_{i|y=1}, \phi_{i|y=0})$$

⑥最大似然求参数：

$$
\begin{aligned}
l(\phi_y, \phi_{i|y=1}, \phi_{i|y=0}) &= log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi_y, \phi_{i|y=1}, \phi_{i|y=0}) \\
&= log \prod_{i=1}^{m} p(y^{(i)}; \phi_y) p(x^{(i)}|y^{(i)}; \phi_{j|y=0}, \phi_{j|y=1}) \\
&= \sum_{i=1}^{m} \Big(log(\phi_y^{y^{(i)}}(1-\phi_y)^{1-y^{(i)}}) + log \prod_{j=1}^{n} p(x_j^{(i)}|y^{(i)}) \Big) \\
&= \sum_{i=1}^{m} \Big((y^{(i)} log\phi_y + (1-y^{(i)}) log(1-\phi_y)) \\
&\quad + log \prod_{j=1}^{n} (p(x_j^{(i)}|y^{(i)}=1))^{I\{y^{(i)}=1\}} (p(x_j^{(i)}|y^{(i)}=0))^{I\{y^{(i)}=0\}} \Big) \\
&= \sum_{i=1}^{m} \Big((y^{(i)} log\phi_y + (1-y^{(i)}) log(1-\phi_y)) \\
&\quad + \sum_{j=1}^{n} I\{y^{(i)}=1\} log\, p(x_j^{(i)}|y^{(i)}=1) \\
&\quad\quad + \sum_{j=1}^{n} I\{y^{(i)}=0\} log\, p(x_j^{(i)}|y^{(i)}=0) \Big) \\
&= \sum_{i=1}^{m} \Big((y^{(i)} log\phi_y + (1-y^{(i)}) log(1-\phi_y)) \\
&\quad + \sum_{j=1}^{n} I\{y^{(i)}=1\} log\, (\phi_{j|y=1})^{x_j^{(i)}} (1-\phi_{j|y=1})^{1-x_j^{(i)}} \\
&\quad\quad + \sum_{j=1}^{n} I\{y^{(i)}=0\} log\, (\phi_{j|y=0})^{x_j^{(i)}} (1-\phi_{j|y=0})^{1-x_j^{(i)}} \Big)
\end{aligned}
$$

全式分为三部分，为求 ϕ_y 对第一部分求导：

$$
\begin{aligned}
\frac{\partial l}{\partial \phi_y} &= \sum_{i=1}^{m} \frac{y^{(i)}}{\phi_y} - \frac{1-y^{(i)}}{1-\phi_y} = 0 \\
&\Rightarrow \phi_y = \frac{\sum_{i=1}^{m} I\{y^{(i)}=1\}}{m}
\end{aligned}
$$

为求$\phi_{j|y=1}$对第二部分求导：

$$
\begin{aligned}
\frac{\partial l}{\partial \phi_{j|y=1}} &= \frac{\partial \sum_{i=1}^{m} I\{y^{(i)}=1\} log\, (\phi_{j|y=1})^{x_j^{(i)}} (1-\phi_{j|y=1})^{1-x_j^{(i)}}}{\partial \phi_{j|y=1}} \\
&= \sum_{i=1}^{m} I\{y^{(i)}=1\} (\frac{x_j^{(i)}}{\phi_{j|y=1}} - \frac{1-x_j^{(i)}}{1-\phi_{j|y=1}}) = 0 \\
\Rightarrow \phi_{j|y=1} &= \frac{\sum_{i=1}^{m} I\{y^{(i)}=1 \wedge x_j^{(i)}=1\}}{\sum_{i=1}^{m} I\{y^{(i)}=1\}}
\end{aligned}
$$

同理，可得$\phi_{j|y=0}$：

$$\phi_{j|y=0}=\frac{\sum_{i=1}^{m}I\{y^{(i)}=0\wedge x_j^{(i)}=1\}}{\sum_{i=1}^{m}I\{y^{(i)}=0\}}$$

所以，我们得到了模型所有的参数值（进行拉普拉斯平滑后的结果）;可以利用贝叶斯公式判别新样本类别：

$$\phi_y=\frac{\sum_{i=1}^{m}I\{y^{(i)}=1\}+1}{m+2}$$
$$\phi_{j|y=1}=\frac{\sum_{i=1}^{m}I\{y^{(i)}=1\wedge x_j^{(i)}=1\}+1}{\sum_{i=1}^{m}I\{y^{(i)}=1\}+2}$$
$$\phi_{j|y=0}=\frac{\sum_{i=1}^{m}I\{y^{(i)}=0\wedge x_j^{(i)}=1\}+1}{\sum_{i=1}^{m}I\{y^{(i)}=0\}+2}$$

最后，**上述建模的贝叶斯分类器，是一个线性分类器**；
即存在某个$\theta\in R^{(n+1)}$,(特征第0个位置，对应截距项θ_0)

$$p(y=1|x)>=p(y=0|x)\ \Leftrightarrow\ \theta^T\begin{bmatrix}1\\x\end{bmatrix}>=0$$

证明：

$$\begin{aligned}
&p(y=1|x)>=p(y=0|x)\\
\Leftrightarrow&\frac{p(y=1|x)}{p(y=0|x)}>=1\\
\Leftrightarrow&\frac{(\prod_{j=1}^{n}p(x_j|y=1))p(y=1)}{(\prod_{j=1}^{n}p(x_j|y=0))p(y=0)}>=1\\
\Leftrightarrow&\frac{(\prod_{j=1}^{n}(\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{(1-x_j)})\phi_y}{(\prod_{j=1}^{n}(\phi_{j|y=0})^{x_j}(1-\phi_{j|y=0})^{(1-x_j)})(1-\phi_y)}>=1\\
\Leftrightarrow&\sum_{j=0}^{n}x_jlog(\phi_{j|y=1})+(1-x_j)log(1-\phi_{j|y=1})+log\phi_y\\
&-\sum_{j=0}^{n}x_jlog(\phi_{j|y=0})+(1-x_j)log(1-\phi_{j|y=0})+(1-log\phi_y)>=0\\
\Leftrightarrow&\sum_{j=1}^{n}x_jlog\frac{\phi_{j|y=1}}{(1-\phi_{j|y=1})}+log(1-\phi_{j|y=1})\\
&-\sum_{j=1}^{n}x_jlog\frac{\phi_{j|y=0}}{(1-\phi_{j|y=0})}-log(1-\phi_{j|y=0})+log\frac{\phi_y}{1-\phi_y}>=0\\
\Leftrightarrow&\sum_{j=1}^{n}x_jlog\frac{\phi_{j|y=1}(1-\phi_{j|y=0})}{(1-\phi_{j|y=1})\phi_{j|y=0}}+log\frac{\phi_y}{1-\phi_y}>=0
\end{aligned}$$

此时：

$$\theta_0=\sum_{j=1}^{m}log\frac{1-\phi_{j|y=1}}{1-\phi_{j|y=0}}+log\frac{\phi_y}{1-\phi_y}$$

$$\theta_j = \frac{\phi_{j|y=1}(1-\phi_{j|y=0})}{(1-\phi_{j|y=1})\phi_{j|y=0}}$$

也就是说，我们拿一个新数据x代入模型测试，利用我们上面得到的线性分类器$\theta^T x$，与利在朴素贝叶斯比较$p(y=1|x)$与$p(y=0|x)$是等效的。**事实上，离散特征的朴素贝叶斯分类器都是线性分类器；方差相同的连续的朴素贝叶斯分类器也是线性分类器。**进一步讲，只有某些具有特定属性的朴素贝叶斯分类器才是线性分类器。

1.3.4 多项式事件模型（词袋模型）

多项式事件模型（multinomial event model）又称为朴素贝叶斯的词袋模型（多用于文档分类），**词袋模型（bag-of-words model）**就是考虑同一文档中重复出现的词以累加，显然词袋模型更加贴合实际；其建模过程，与伯努利事件模型相像，只是特征不在只取二值，而是多值$\{1,2,3,4,...\}$。这就与伯努利事件模型出现了差异，**其每一个特征变量不在服从伯努利分布而是多项式分布。**最大似然的过程相似，但结果不同，其似然结果为：

$$\phi_y = \frac{\sum_{i=1}^{m} I\{y^{(i)}=1\}+1}{m+2}$$

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{y^{(i)}=1 \wedge x_j^{(i)}=k\}+1}{\sum_{i=1}^{m} I\{y^{(i)}=1\}n_i+|V|}$$

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{y^{(i)}=0 \wedge x_j^{(i)}=k\}+1}{\sum_{i=1}^{m} I\{y^{(i)}=0\}n_i+|V|}$$

k ,为模型总词表中的一个个单词；
n_i ,为一个个样本文档的有效长度；
$|V|$ ，是文档总词表的长度。

1.3.5 拉普拉斯平滑

做拉普拉斯平滑处理的原因很显然，也就是零概率问题，即就是在计算实例的概率时，如果某个量x，在观察样本库（训练集）中没有出现过，会导致整个实例的概率结果是0。在文本分类的问题中，当一个词语没有在训练样本中出现，该词语调概率为0，使用连乘计算文本出现概率时也为0。这是不合理的，不能因为一个事件没有观察到就武断的认为该事件的概率是0。

为了解决零概率的问题，法国数学家拉普拉斯最早提出用加1的方法估计没有出现过的现象的概率，所以加法平滑也叫做拉普拉斯平滑。假定训练样本很大时，每个分量x的计数加1造成的估计概率变化可以忽略不计，但可以方便有效的避免零概率问题。上述两种朴素贝叶斯模型的参数估计值，最终结果式都做了拉普拉斯平滑。

1.3.6 实战：python实现朴素贝叶斯分类器分类文本

在自然语言处理中，朴素贝叶斯的作用非常广泛；利用朴素贝叶斯进行文本分类，是因为它模型简易，且准确率较高。经过上面的学习知道了朴素贝叶斯之所以朴素是因为它存在两个假设，一是特征变量之间独立性假设；其二就是特征之间同等重要假设。这两个假设分明在现实的文本分类中不成立，尽管如此，朴素贝叶斯模型还会有一个很好的精度。这就使得朴素贝叶斯的优缺点显露无疑。

优点：数据较少仍然有效；适用于多分类。

缺点：对于输入数据的准备方式比较敏感。

适用数据类型：标称量数据。

因为词袋模型更符合实际情况，且包含了简单的词集模型，那我利用词袋模型进行实战；实战的流程分为**五步**为：

①生成文档总词表：

```
def loadDataSet(): #import text data
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
                 ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                 ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                 ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop',
                     'him'],
                 ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1] #1 is abusive, 0 not
    return postingList,classVec

def createVocabList(dataSet):#create word vector list to contain all text
    information
    vocabSet = set([]) #create empty set
    for document in dataSet:
        vocabSet = vocabSet | set(document) #union of the two sets
    return list(vocabSet)
```

这里文档总词表手动生成便于测试算法；示例实战中的总词表是作用于文档，经过切分文本等相对复杂的文本解析函数生成的。

②构建文档词向量：

```
def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
```

```
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

将文档列表转换为文档词向量，由于是词袋模型，所以目标词未出现置0，出现则置出现的次数。

③训练模型得到参数：

```
def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix) # #sample
    numWords = len(trainMatrix[0]) # #vocabulary
    pAbusive = (sum(trainCategory))+1)/(float(numTrainDocs)+2) #laplace
        smoothing
    p0Num = np.ones(numWords); p1Num = np.ones(numWords) #change to ones() #1
        equal #vocabulary. Laplace smoothing
#    print (p0Num,p1Num)
    p0Denom = numWords; p1Denom = numWords                    #change to
        numWords-#vacabulary laplace smoothing
    for i in range(numTrainDocs):#6
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i]) #bag of words model (multinomial
                event model(Andrew Ng))
#           p1Denom += 1 #set of words model (multi-variate Bernoulli event
    model(Andrew Ng))
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i]) #bag of words model (multinomial
                event model(Andrew Ng))
#           p0Denom += 1 #set of words model (multi-variate Bernoulli event
    model(Andrew Ng))
#       print (p0Num,p1Num,p0Denom,p1Denom)
    p1Vect = np.log(p1Num/p1Denom)       #change to log() avoid underflow
    p0Vect = np.log(p0Num/p0Denom)       #change to log()
#    print (p1Vect,p0Vect)
    return p0Vect,p1Vect,pAbusive
```

此函数为整个朴素贝叶斯算法的核心，其中有三个重要问题；第一，利用python实

现的是朴素贝叶斯词袋模型也就是多项式事件模型，但与Andrew Ng所讲的多项式事件模型有些区别，这里没有将单个文档的词向量设为变长，但是实质是一样的；第二，拉普拉斯平滑，将分子分母加上相应的数字，注意p1Denom和p0Denom，这两个数字的初始值是文档总词表的长度。第三，由于判定分类时，要经过概率求积，但是在实现时，多个很小的概率值求积可能会产生下越界，因此在这里取自然对数，将求积运算转化成求和运算来避免下越界。

④模型用于分类：

```
import naiveBayes

listOPosts , listClasses = naiveBayes.loadDataSet()
myVocabList = naiveBayes.createVocabList(listOPosts)
wordVec = naiveBayes.setOfWords2Vec(myVocabList,listOPosts[0])
trainMat = []
for postinDoc in listOPosts:
    trainMat.append(naiveBayes.setOfWords2Vec(myVocabList,postinDoc))
p0V,p1V,PAb = naiveBayes.trainNB0(trainMat,listClasses)

print ("set of words models:========================================")
naiveBayes.testingNBsetOfwords()
print ("============================================================")
print ("\nbag of words models:========================================")
naiveBayes.testingNBbagofwords()
print ("============================================================")

output:
['love', 'my', 'dalmation', 'to', 'dog', 'part', 'yes'] classified as: 0
['stupid', 'garbage', 'conveninence'] classified as: 1
```

最终将两个测试文档成功分类。

⑤测试算法准确度：

算法准确度在下节应用朴素贝叶斯算法进行实战时，在进行测算。之前已经确保了算法模型的正确性以及可行性验证。

1.3.7 示例：使用朴素贝叶斯过滤垃圾邮件

利用朴素贝叶斯词袋模型（多项式事件模型），进行垃圾邮件的过滤。实战的过程与上节一致，只有两部分差异；第一部分为，输入数据为许多邮件文本，先要经过文本

解析转变为邮件词列表，再生成文档总词表；第二部分是测试算法时利用交叉验证进行的，最终得到准确度。

①生成文档总词表：

```
def textParse(bigString): #input is big string, #output is word list
    import re
    pattern = re.compile('\\W+') #one or more word
    listOfTokens = pattern.split(bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2] #return list

def createVocabList(dataSet):#create word vector list to contain all text
    information
    vocabSet = set([]) #create empty set
    for document in dataSet:
        vocabSet = vocabSet | set(document) #union of the two sets
    return list(vocabSet)
```

python利用正则表达式模块很方便的实现文本解析，生成目标总词表及相应文档词列表。

⑥完整的测试函数：

```
def spamTest():#process function
    docList=[]; classList = []; fullText =[]
    for i in range(1,26):
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)     #list of list
        fullText.extend(wordList)    #list of word
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)    #create vocabulary
    trainingSet = list(range(50)); testSet=[]      #create test set

    print ("all valid words number:",len(fullText))
    print ("The length of word vector:",len(vocabList))

    for i in range(10):#select randomly 10 mails as test set
```

```
    randIndex = int(np.random.uniform(0,len(trainingSet)))
    testSet.append(trainingSet[randIndex])
    del(trainingSet[randIndex])

trainMat=[]; trainClasses = [] #train naive bayes classifier in trainSet
for docIndex in trainingSet:#train the classifier (get probs) trainNB0
    trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
    trainClasses.append(classList[docIndex])
p0V,p1V,pSpam = trainNB0(np.array(trainMat),np.array(trainClasses))


errorCount = 0        #test classifier
for docIndex in testSet:     #classify the remaining items
    wordVector = bagOfWords2VecMN(vocabList, docList[docIndex]) #numpy array
    if classifyNB(np.array(wordVector),p0V,p1V,pSpam) !=
        classList[docIndex]:
        errorCount += 1
        print ("classification error",docIndex,docList[docIndex])
print ('the error rate is: ',float(errorCount)/len(testSet))
#return vocabList,fullText
```

此过程利用所有模块函数进行实战过程，包括**文本解析，生成模型总词表和文档词列表，随机构建训练集与测试集，训练集训练生成概率参数，测试集测试得到结果**。最终测试100次中有34次错误率为0.1，其它全为0。因此平均错误率为3.4%。

1.4 支持向量机

1.4.1 逻辑回归与支持向量机

对于支持向量机我的理解是决定模型的生成只与支持向量相关；支持向量机一度被认为**最高效**的分类算法。那么逻辑回归与支持向量机都属于**判别学习算法**，来看看他们的区别于联系。

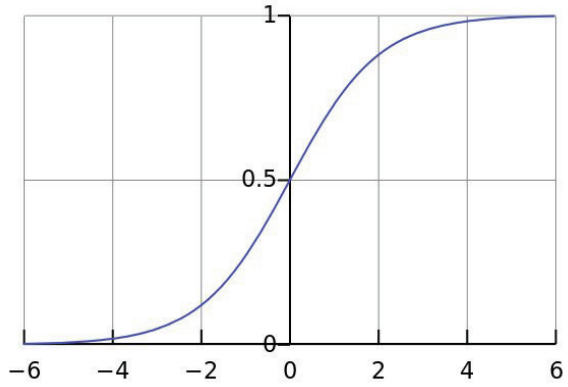

图 12: 逻辑函数

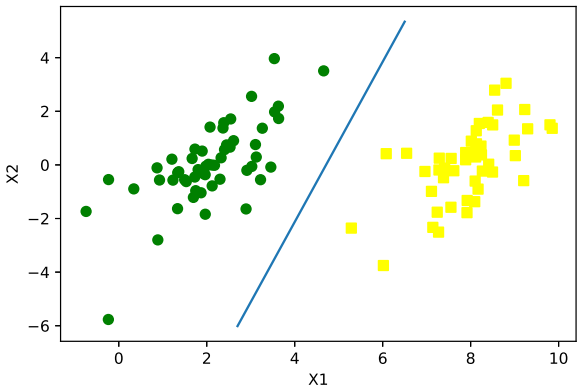

图 13: 线性可分模型构想

逻辑函数:

$$y = h(z) = \frac{1}{1 + exp(-z)}$$

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

易知:

$$z >> 0\ ,\ y = 1 \qquad z << 0\ ,\ y = 0$$

反之：

$$y = 1\ ,\ z >> 0 \qquad y = 0\ ,\ z << 0$$

那么试想若对于一个二分类问题，线性可分下最优的决策边界就是离两类数据特征向量**最远**的情况。这就是由逻辑回归以及事实情况启发而来，然后是要解决线性可分下找出这样的决策边界，即就是分割（超）平面。

那么从下面只有一两个特征线性可分的数据来看，这样的决策边界事实上只与少量的特征向量相关，就是离这个超平面最近的这几个点，把这些点称为**支持向量**。这也是支持向量机高效的原因。

那么接下来要做的工作便是利用数学方法将求出这个显然存在的分割最优超平面。为此定义了两种间隔，**函数间隔**（functional margin）与**几何间隔**（geometric margin）。

1.4.2 函数间隔与几何间隔

一般来说，一个点的离分离超平面的远近可用 $|w \cdot x + b|$ 的大小来衡量，而分类的正确性可用 $(w \cdot x + b)$ 与类标记 y 的符号是否一致来判定。所以可以用 $y(w \cdot x + b)$ 来表示分类的正确性与确信度。这就是**函数间隔**。

函数间隔 对于给定的训练集数据集 T 和超平面 (w, b) , T 中所有的样本点 (x_i, y_i) 的函数间隔为

$$\hat{\gamma_i} = y_i(w \cdot x_i + b)$$

定义超平面 (w, b) 关于训练集 T 的函数间隔为超平面 (w, b) 关于 T 中所有样本点 (x_i, y_i) 的函数间隔之最小值，即

$$\hat{\gamma} = \min_{i=1,\ldots,N} \hat{\gamma_i}$$

函数间隔可以表示分类预测的正确性以及确信度，但是选择分离超平面时，只有函数间隔还不够。原因是如果成比例的改变 w,b ,超平面没有变而函数间隔为原来的二倍。因此我们需要规范化，如 $||w|| = 1$,使得函数间隔变为**几何间隔**。

几何间隔 对于给定的数据集 T 和超平面 (w, b) ，定义超平面 (w, b) 关于样本点 (x_i, y_i) 的几何间隔为

$$\gamma_i = y_i(\frac{w}{||w||} \cdot x_i + \frac{b}{||w||})$$

定义超平面 (w, b) 关于训练集 T 的几何间隔为超平面 (w, b) 关于 T 中所有样本

点 (x_i, y_i) 的几何间隔的最小值，即

$$\gamma = \min_{i=1,...N} \gamma_i$$

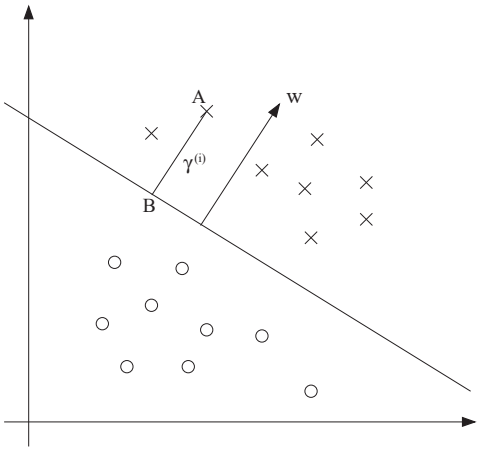

图 14: 几何间隔

1.4.3 最优间隔分类器的产生

定义了函数间隔与几何间隔之后，然后利用数学优化方法，显式表达这个求最优分割超平面的优化问题。

优化问题的转变

$$\begin{aligned} &\max_{\gamma,w,b} \ \gamma \\ s.t. \ &y^{(i)}(w^T x^{(i)} + b) \ \geq \ \gamma, \ \ i = 1, ..., m \\ &||w|| = 1 \end{aligned}$$

由于一号优化问题中 $||w|| = 1$ 是一个非凸性的约束，导致此优化问题难以求解，因此改变这个优化问题。

$$\begin{aligned} &\max_{\gamma,\hat{w},b} \ \frac{\hat{\gamma}}{||w||} \\ s.t. \ &y^{(i)}(w^T x^{(i)} + b) \ \geq \ \gamma, \ \ i = 1, ..., m \end{aligned}$$

二号优化问题用了另外的放缩方式，但依然是一个非凸性的优化问题，还需进行转变利用放缩条件 $\hat{\gamma} = 1$ 产生三号也是最终的优化问题。

$$\begin{aligned} &\min_{w,b} \ \frac{1}{2}||w||^2 \\ s.t. \ &y_i(w \cdot x_i + b) - 1 \geq 0, \ \ i = 1, 2, ..., m \end{aligned}$$

最大间隔分离超平面的存在唯一性定理 若训练集 T 线性可分，则可将训练数据集中的样本点完全正确分开的最大间隔分离超平面存在且唯一。

那么由定理可知最优间隔超平面存在且唯一，那么求解这个凸二次规划问题既可以产生这个最有间隔分类器。关于求解凸二次规划问题，可以利用如梯度下降等最优化算法下的QP软件来做，现在大多数是利用拉格朗日对偶学习算法来做。不仅是在求解速度上有优势，而且便于由线性向非线性扩充。

支持向量与间隔边界 在线性可分情况下，训练数据集的样本点中与分离超平面距离最近的样本点的实例称为支持向量(support vector)，支持向量是使约束条件式等号成立的点，即

$$y_i(w \cdot x_i + b) - 1 = 0$$

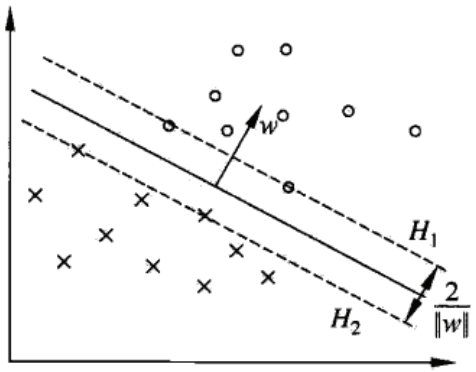

图 15: 支持向量与间隔边界

注意到 H_1 和 H_2 平行，并且没有实例点落在它们中间,在 H_1 与 H_2 ，之间形成一条长带，分离超平面与它们平行且位于它们中央长带的宽度，即 H_1 与 H_2 之间的距离称为间隔(margin)。间隔依赖于分离超平面的法向量 w ,等于 $\frac{2}{||w||}$, H_1 和 H_2 称为间隔边界。

在决定分离超平而时只有支持向量起作用，而其他实例点并不起作用如果移动支持向量将改变所求的解，但是如果在间隔边界以外移动其他实例点，甚至去掉这些点，则解是不会改变的。由于支持向量奄确定分离超平面中起着决定性作用，所以将这种分类模型称为支持向量机支持向量的个数般很少，所以支持向量机由很少的“重要的”训练样本确定。

1.4.4 拉格朗日对偶

为了求解线性可分支持向量机的最优化问题，将它作为原始最优化问题，应用拉格

朗日对偶性，通过求解对偶问题(duality problem)得到原始问题(primal problem)的最优解，这就是线性可分支持向量机的对偶算法(duality algorithm).这样做的优点,一是对偶问题往往更容易求解,二是自然引入核函数,进而推广到非线性分类问题.

拉格朗日数乘的一般形式

问题:

$$\begin{aligned} \min_{w} \quad & f(w) \\ s.t. \quad & g_i(w) \leq 0, \quad i = 1, ..., k \\ & h_i(w) = 0, \quad i = 1, ..., l \end{aligned}$$

拉格朗日对偶： Lagrangian 拉格朗日算子

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

定义,

$$\Theta_p(w) = \max_{\alpha, \beta, \alpha_i \geq 0} L(w, \alpha, \beta)$$

原始问题

$$p^* = \min_{w} \max_{\alpha, \beta, \alpha_i \geq 0} L(w, \alpha, \beta) = \min_{w} \Theta_p(w)$$

若 $g_i(w) > 0$ 或者 $h_i(w) \neq 0$,有

$$\Theta_p(w) = \max_{\alpha, \beta, \alpha_i \geq 0} f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w) = \infty$$

否则:

$$\Theta_p(w) = f(w)$$

有,

$$\Theta_p(w) = \begin{cases} f(w) & \text{w满足原始约束} \\ \infty & \text{w不满足原始约束} \end{cases}$$

最终导出

$$\min_{w} \Theta(w) = \min_{w} \ f(w) \quad s.t. \ ...$$

对偶问题

定义

$$\Theta_D(\alpha, \beta) = \min_{w} L(w, \alpha, \beta)$$

$$d^* = \max_{\alpha \geq 0, \beta} \min L(w, \alpha, \beta) = \max_{\alpha \geq 0, \beta} \Theta_D(\alpha, \beta)$$

定理 $d^* = \max\limits_{\alpha\geq 0,\beta} \min L(w,\alpha,\beta) \leq \min\limits_{w} \max\limits_{\alpha,\beta,\alpha_i\geq 0} L(w,\alpha,\beta) = p^*$

即就是: $d^* \leq p^*$

对于对偶问题有很多有用的性质；解决原始优化问题的有效途径就是解决对偶问题。那么等号在什么条件下成立，就是KKT条件。

KKT条件(对偶问题 d^* 与原始问题 p^* 等价条件)

1. $\frac{\partial}{\partial w_i} L(w^*,\alpha^*,\beta^*) = 0,\ \ i = 1,...,n$
2. $\frac{\partial}{\partial \beta_i} L(w^*,\alpha^*,\beta^*) = 0,\ \ i = 1,...,l$
3. $\alpha_i^* g_i(w^*) = 0,\ \ i = 1,...,k$
4. $g_i(w^*) \leq 0,\ \ i = 1,...,k$
5. $\alpha_i \geq 0,\ \ i = 1,...,k$

由3可知, 当 $\alpha_i > 0$ 时，$g_i(w^*) = 0$

当 $\alpha_i \neq 0$ 时，$g_i(w^*) = 0$

1.4.5 利用对偶问题求解最优间隔分类器

2 回归

回归是解决监督学习目标变量为连续值时，采用的学习方法。

回归，指研究一组随机变量$(Y_1, Y_2, \ldots, Y_i)$和另一组$(X_1, X_2, \ldots, X_k)$变量之间关系的统计分析方法，又称多重回归分析。通常$Y_1, Y_2, \ldots, Yi$是因变量，$X_1, X_2, \ldots, X_k$是自变量。

2.1 线性回归

线性回归的目的是寻找最佳拟合直线，做线性回归实战之前，要理解的东西有：线性函数是高斯分布在广义线性模型下建模；最小二乘法的概率解释；利用正规方程法求锝最佳回归系数；梯度下降法求得最佳回归系数。

2.1.1 对高斯分布进行广义线性建模

根据广义线性模型的形式化定义推导出高斯分布建模情况：

1. $N(\mu, \sigma^2) \sim ExponentialFamily(\eta)$

2.令 $\sigma = 1$

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2})(y-\mu)^2 \\ &= \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2}y^2) \cdot exp(\mu y - \frac{1}{2}\mu^2) \end{aligned}$$

对照指数分布族，得：

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ a(\eta) &= \mu^2/2 \\ &= \eta^2/2 \\ b(y) &= (1/\sqrt{2\pi}) exp(-y^2/2). \end{aligned}$$

$$So, \quad h(x) = E(y|x) = \mu = \eta$$

3.自然常数 η 与输入 X 是线性相关的，即： $\eta = \theta x$,

若 η 是向量,则 $\eta_i = \theta_i^T x$. (我理解是预测多标签数据时的情况)。

2.1.2 最小二乘法的概率解释：最大似然估计

当我们面对回归问题时，为什么会采用线性回归，最小二乘法来定义成本函数，即1/2的差的平方和。这里给出概率解释：

我们拟合的直线的函数值即预测值必然和真实值会存在误差。那么假定一个等式：

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

其中各个样本的误差项，是独立同分布且服从高斯分布（正态分布）。（可根据中心极限定理来看）

即就是：

$$\epsilon^{(i)} \sim N(0, \sigma^2)$$

$$P(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2})$$

其 ϵ 满足均值为0的正太分布易理解。因此：

$$P(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

也就是要面对在以为参数给定一个x时预测值y是真实值的概率服从正太分布，要求得概率最大时的?

则采用最大似然估计：

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} P(y^{(i)}|x^{(i)}) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}) \end{aligned}$$

$$\begin{aligned} l(\theta) &= ln(L(\sigma)) \\ &= ln \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}) \\ &= \prod_{i=1}^{m} ln \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}) \\ &= mln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \end{aligned}$$

于是有：

$$J(\theta) = \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

根据此过程，要求此 $L(\theta)$ 函数的最大值，需求上式中后项函数的最小值 $J(\theta)$ ，

函数 $J(\theta)$ 又即为最小二乘估计的成本函数。

结论：上式推导即为最小二乘的概率解释。

2.1.3 正规方程法找最佳回归系数

1.矩阵求导

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

函数自变量是矩阵，求导是对矩阵的每一个元素分别求导后，组成新的矩阵。
例：

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$
$$f(A) = \frac{2}{3}A_{11} + 5A_{12}^2 + A_{21}A_{22}$$
$$\nabla_A f(A) = \begin{bmatrix} \frac{2}{3} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

2.矩阵的迹及常用性质与相关结论

矩阵的迹：

$$trA = \sum_{i=1}^{m} A_{ii}$$

矩阵迹的常用性质：

$$trABC = trCAB = trBCA$$
$$trA = trA^T$$
$$trA + B = trA + trB$$
$$traA = atrA$$

矩阵迹与矩阵求导相关结论：

$$\nabla_A trAB = B^T \tag{1}$$
$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T \tag{2}$$
$$\nabla_A trABA^TC = CAB + C^TAB^T \tag{3}$$

$$\nabla_A |A| = |A|(A^{-1})^T \tag{4}$$

$combining\ (2)\ and\ (3)\ :$

$$\nabla_{A^T} trABA^TC = B^TA^TC^T + BA^TC \tag{5}$$

3.利用正规方程求最佳回归系数

$input:$

$$X = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$now:$

$$\begin{aligned} J(\theta) &= \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}))^2 \end{aligned}$$

$Hence:$

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2}\nabla_\theta(\theta^TX^TX\theta - \theta^TX^T\vec{y} - \vec{y}^TX\theta + \vec{y}^T\vec{y}) \\ &= \frac{1}{2}\nabla_\theta tr(\theta^TX^TX\theta - \theta^TX^T\vec{y} - \vec{y}^TX\theta + \vec{y}^T\vec{y}) \\ &= \frac{1}{2}\nabla_\theta(tr\theta^TX^TX\theta - 2tr\vec{y}^TX\theta) \\ &= \frac{1}{2}(X^TX\theta + X^TX\theta - 2X^T\vec{y}) \\ &= X^TX\theta - X^T\vec{y} = 0 \end{aligned}$$

$Normal\ equations:$

$$X^TX\theta = X^T\vec{y}$$

$if\ (X^TX)^{-1}\ exist:$

$$\theta = (X^TX)^{-1}X^T\vec{y}$$

2.1.4 实战：利用线性回归寻找最佳拟合直线

优点：结果易于理解，计算上不复杂。

缺点：对非线性的数据拟合不好。

适用数据类型：数值型和标称量型的数据。

实战是利用高斯分布广义线性建模出的连接函数，再利用最大似然估计得到最小二乘的成本函数。现，为得到最佳回归系数，有两种方式得到成本函数的最小值。
1.最优化算法。（梯度下降算法，牛顿法等）
2.正规方程式。
两种方法各有千秋，最优化算法在训练集数量庞大时，优势便可以显现出来，因为其可以使用在线的最优化算法。而正规方程的方法，有严格的理论支持，若条件满足能得到最精确的最佳拟合直线，且不需要调参。、

用到的数学理论公式在上几节中都有记录：

$$
\begin{aligned}
&link\ function\ :\\
&\qquad h(x) = E(y|x) = \mu = \eta\\
&cost\ function\ :\\
&\qquad J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y})\\
&\qquad\quad = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}))^2\\
&Normal\ equations\ :\\
&\qquad X^TX\theta = X^T\vec{y}\\
&if\ (X^TX)^{-1}\ exist\ :\\
&\qquad \theta = (X^TX)^{-1}X^T\vec{y}
\end{aligned}
$$

程序核心函数代码：

```
def loadDataSet(fileName):   #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) - 1 #get number of
        fields 2
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t') #['1.000000', '0.116163', '3.129283']
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr) #[[1.0, 0.52707], [1.0, 0.116163],...]
        labelMat.append(float(curLine[-1])) #[4.225236, 4.231083,...]
    return dataMat,labelMat
```

```
def standRegres(xArr,yArr):
    xMat = np.mat(xArr) #to matrix
    yMat = np.mat(yArr).T #to matrix with transform
    xTx = xMat.T*xMat #xMat.T*xMat*w - xMat.T*yMat = 0
    if np.linalg.det(xTx) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws


xArr,yArr = linear_regression.loadDataSet('ex0.txt') #feature matrix

ws = linear_regression.standRegres(xArr,yArr) #regression coefficient vector

#ouput : y = ws[0] + ws[1]*x
#drawing

xMat = np.mat(xArr)
yMat = np.mat(yArr)
yHat = xMat*ws    #prediction value

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:,1].flatten().A[0],yMat.T[:,0].flatten().A[0],color='k')

xCopy = xMat.copy()
xCopy.sort(0) #sorted
yCHat = xCopy*ws

ax.plot(xCopy[:,1],yCHat,color='k')

plt.savefig('linear_fitting.eps',dpi=2000)
plt.show()

correlation_coefficient = np.corrcoef(yHat.T,yMat)
#1 0.986474
#0.986474  1
```

利用测试数据得到拟合直线，并可视化表示结果如下图:

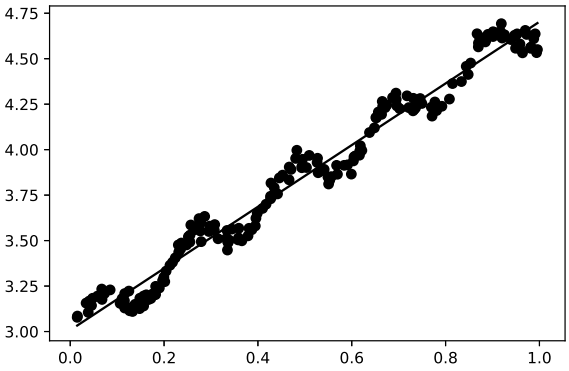

图 16: 线性回归得到拟合直线

分析:

任意数据集都可用线性进行建模，只是建模好坏有差异；程序中算得相关系数来计算预测值与真实值的匹配程度。
从拟合的效果图来看相当不错，但似乎有些欠拟合。该线性模型不能很好拟合出数据所存在的模式。

2.1.5 利用局部加权线性回归寻找最佳拟合直线

首先知道线性回归的一个问题就是欠拟合，将不能取得很好的预测效果。因为它是最小均方误差的无偏估计。解决这一问题的方法就是允许估计中存在一些偏差。其中一个比较有效的方法就是局部加权线性回归（Locally Weighted Linear Regression）。
算法思想:

比较线性回归与局部加权线性回归:

$$LR:$$

$$(1).Fit\ \theta\ to\ minimize\ \sum_{i=1}^{m}(y^{(i)}-\theta^T x^{(i)})^2$$

$$(2).Output:\ \theta^T x$$

$$LWLR:$$

$$(1).Fit\ \theta\ to\ minimize\ \sum_{i=1}^{m}w_{(i)}(y^{(i)}-\theta^T x^{(i)})^2$$

$$(2).Output:\ \theta^T x$$

$$weighted:$$

$$w_{(i)}=exp(-\frac{(x_{(i)}-x)^2}{2\tau^2})$$

解释:

当样本点 $x^{(i)}$ 接近预测点x时，权值大。 $w^{(i)} \sim 1$.

当样本点 $x^{(i)}$ 远离预测点x时，权值小。 $w^{(i)} \sim 0$.

权值系数 $w^{(i)}$ 指数衰减，其中参数 τ 为衰减因子，即权重衰减的速率。

τ 越小权重衰减越快。(依据权重函数易得知)

从后面实战中我们可以更好的理解参数 τ .

最小二乘法，求解最佳回归系数。

$$\begin{aligned} J(\theta) &= \frac{1}{2}(X\theta - \vec{y})^T W(X\theta - \vec{y}) \\ &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)}(h_\theta(x^{(i)} - y^{(i)}))^2 \end{aligned}$$

$weighted:$

$$w_{(i)} = exp(-\frac{(x_{(i)} - x)^2}{2\tau^2})$$

$order:$

$$\begin{aligned} &\nabla_\theta J(\theta) = 0 \\ \Rightarrow\ &X^T W X\theta = X^T W\vec{y} \\ \Rightarrow\ &\theta = (X^T W X)^{-1} X^T W\vec{y} \end{aligned}$$

程序核心函数代码:

```
def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = np.mat(xArr); yMat = np.mat(yArr).T
    m = np.shape(xMat)[0]
    weights = np.mat(np.eye((m)))
    for j in range(m):                    #next 2 lines create weights matrix
        diffMat = testPoint - xMat[j,:] #difference matrix
        weights[j,j] = np.exp(diffMat*diffMat.T/(-2.0*k**2)) #weighted matrix
    xTx = xMat.T * (weights * xMat)
    if np.linalg.det(xTx) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    ws = xTx.I * (xMat.T * (weights * yMat)) #normal equation
    return testPoint * w
```

参数 τ 的作用范围:

实验结果可视化:

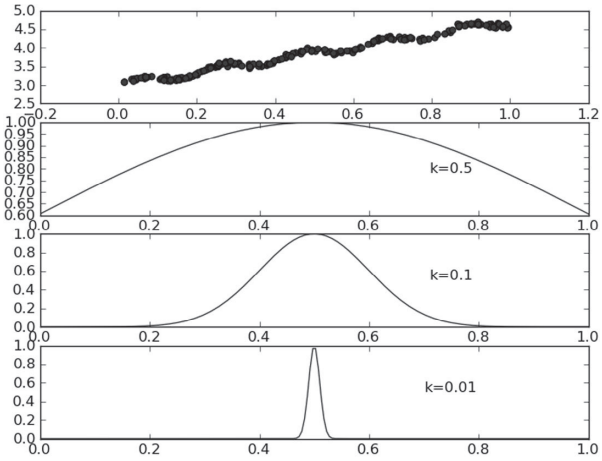

图 17: τ 的作用范围

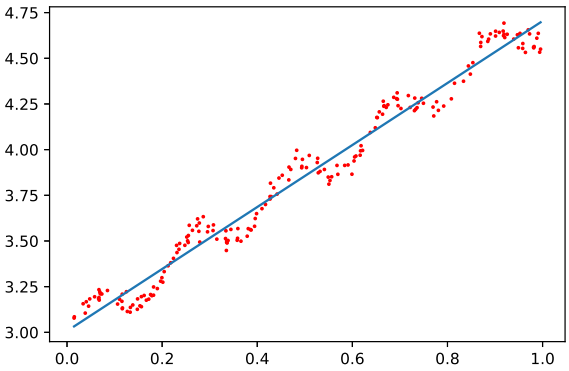

图 18: $\tau = 1$

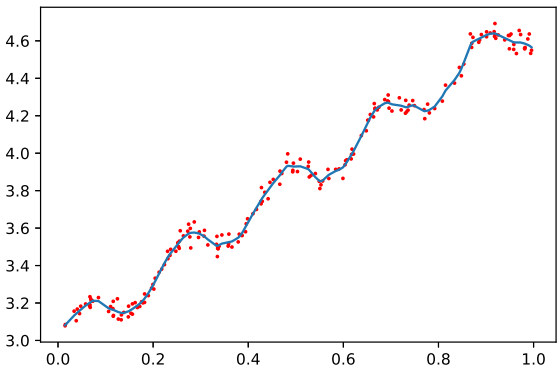

图 19: $\tau = 0.01$

分析：

从结果来看局部线性回归能很好地解决线性回归欠拟合的问题，但又可能出现过拟合。

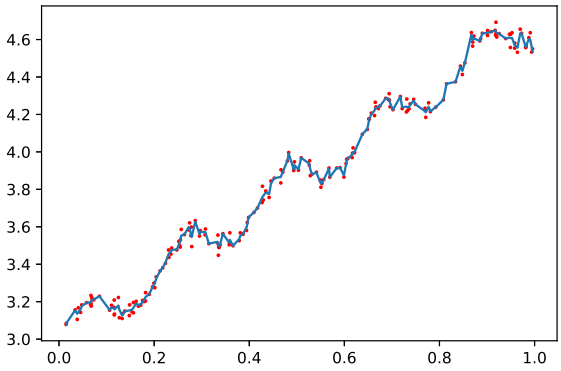

图 20: $\tau = 0.003$

所以参数调整影响了模型的泛化能力。选取合适参数至关重要。
虽然局部线性回归能增强模型的泛化能力。但是它也有自己的缺陷。就是对每个点的预测都必须使用整个数据集。这样大大增加了计算量。

存在问题:
考虑一个问题，当数据特征比训练集样本点还多时，也就是说不可逆，矩阵求导无计可施。此时就要用缩减样本来“理解”数据，求得回归系数矩阵。即就是:

$$
\begin{aligned}
&R(X^TX) = R(X) = m \\
&if\ \#feature > \#sample\ \ (n > m) \\
&X^TX \in R^{n\times n} \\
So,&(X^TX)^{-1}\ not\ exist.
\end{aligned}
$$

2.1.6 示例：利用线性回归预测鲍鱼年龄

实战代码以及结果见注释:
程序核心函数代码:

```
def standRegres(xArr,yArr):
    xMat = np.mat(xArr)
    yMat = np.mat(yArr).T
    xTx = xMat.T*xMat #xMat.T*xMat*w - xMat.T*yMat = 0
    if np.linalg.det(xTx) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws
```

```
def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = np.mat(xArr); yMat = np.mat(yArr).T
    m = np.shape(xMat)[0]
    weights = np.mat(np.eye((m)))
    for j in range(m):                    #next 2 lines create weights matrix
        diffMat = testPoint - xMat[j,:] #difference matrix
        weights[j,j] = np.exp(diffMat*diffMat.T/(-2.0*k**2)) #weighted matrix
    xTx = xMat.T * (weights * xMat)
    if np.linalg.det(xTx) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    ws = xTx.I * (xMat.T * (weights * yMat)) #normal equation
    #ws #7 feature,and 1
    return testPoint * ws

#locally weighted linear regression
yHat01 = function.lwlrTest(abX[0:99],abX[0:99],abY[0:99],0.1) #training set
    #0-99
yHat1 = function.lwlrTest(abX[0:99],abX[0:99],abY[0:99],1)
yHat10 = function.lwlrTest(abX[0:99],abX[0:99],abY[0:99],10)

#error
error01 = function.rssError(abY[0:99],yHat01.T) #56.820227823572182
error1 = function.rssError(abY[0:99],yHat1.T) #429.89056187016683
error10 = function.rssError(abY[0:99],yHat10.T) #549.1181708825128

#generalization
yHat01g = function.lwlrTest(abX[100:199],abX[100:199],abY[100:199],0.1) #test
    set #100-199
yHat1g = function.lwlrTest(abX[100:199],abX[100:199],abY[100:199],1)
yHat10g = function.lwlrTest(abX[100:199],abX[100:199],abY[100:199],10)

#error
error01g = function.rssError(abY[100:199],yHat01g.T) #36199.797699875046
error1g = function.rssError(abY[100:199],yHat1g.T) #231.81344796874004
error10g = function.rssError(abY[100:199],yHat10g.T) #291.87996390562728

#compare
#k = 0.1 overfitting
```

```
#linear regression
ws = function.standRegres(abX[0:99],abY[0:99])
yHat = np.mat(abX[100:199])*ws
errorlr = function.rssError(yHat.T.A,abY[100:199]) #518.63631532510897

#compare
#lwlr is better than lr
```