# 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



*电子邮件：1194585271@qq.com

# 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$\eta$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$\eta$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$x,\theta$后,$y$满足的概率分布是以$\eta$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 $\eta$ 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 $\theta$

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 $\theta$**

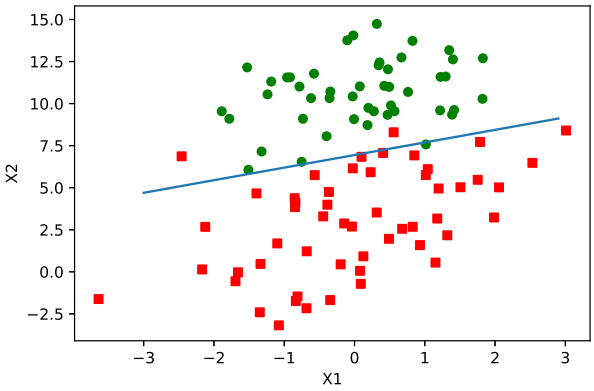



图 1: 利用梯度上升做逻辑回归的分类结果

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。

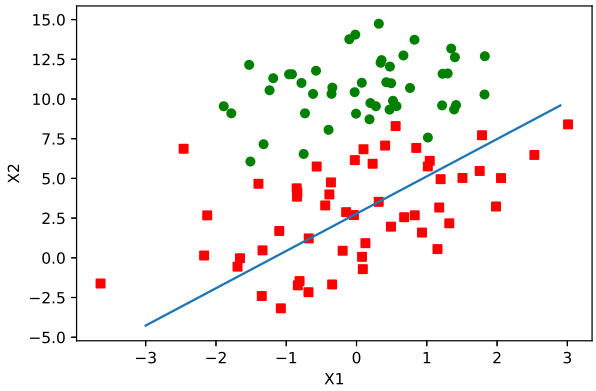



图 2: 利用随机梯度上升做逻辑回归的分类结果，并非最佳分类线

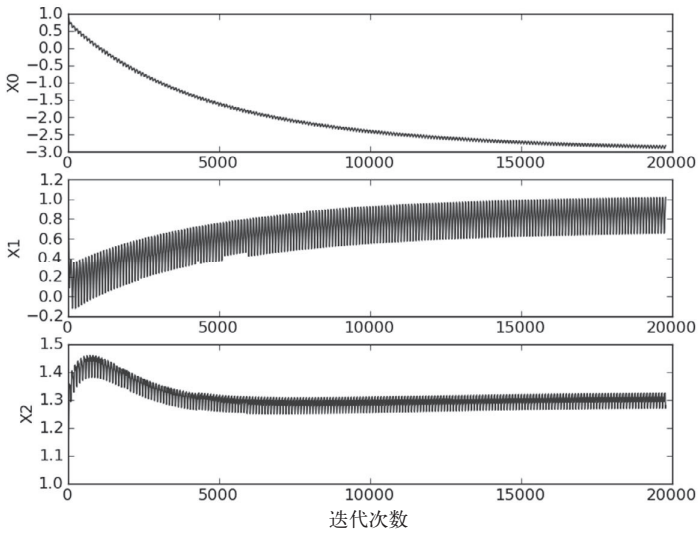



图 3: 回归系数与迭代次数的关系图

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 $\theta$

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:

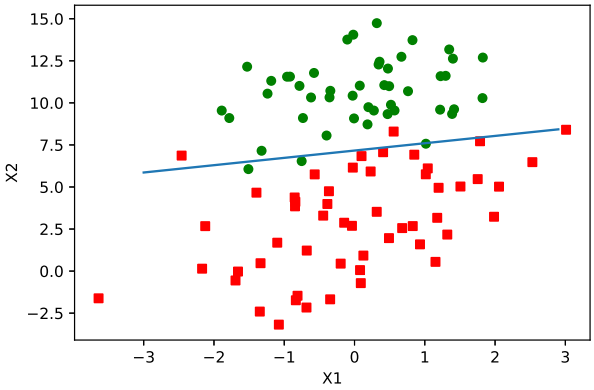



图 4: 利用改进的随机梯度上升做逻辑回归的分类结果

算法参数分析:

1.设置动态步长 $\alpha$ ，缓解上节中数据的波动或者高频波动，比固定的 $\alpha$ 收敛速度更快。当 $j << max(i)$ 时，$\alpha$ 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：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

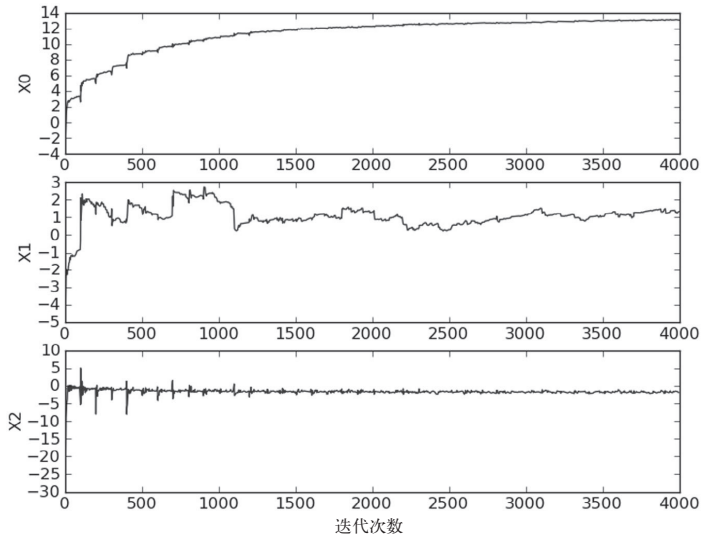



图 5: 改进算法后回归系数与迭代次数关系图

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 k-近邻算法

k-近邻法（K-NN）是一种基本分类与回归方法，那么我现只接受了分类问题。此算法是一个多分类的机器学习算法，但是它是一个在线学习的算法，每一次做判别都要用到所有的训练集，寻找最小距离的训练样本。作为一个分类的“模型”，k 值的选择、距离度量及分类决策规则是k近邻法的三个基本要素。

### 1.2.1 k近邻模型

关于k近邻模型，我理解它是传统机器学习算法最简单的学习模型之一。k近邻法中，当训练集、距离度量（如欧氏距离〉、k值及分类决策规则（如多数表决〕确定后，对于任何一个新的输入实例，那么所属的类唯一地确定．这相当于根据上述要素将特征空间划分为一些子空间，确定于空间里的每个点所属的类。

### 1.2.2 距离度量

特征空间中两个实例点的距离是两个实例点相似程度的反映。k近邻模型的特征空间一般是n维实数向量空间 $R^n$ ，使用的距离是欧氏距离，但也可以是其他距离，如更一般的 $L_p$ 距离 ($L_p$ *distance*) 或 Minkowski 距离(Minkowski distance)。

$L_p$距离：

$$L_p(x_i, x_j) = \Big(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p\Big)^{\frac{1}{p}}$$

$p = 1$ 曼哈顿距离：

$$L_1(x_i, x_j) = \Big(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|\Big)$$

$p = 2$ 欧氏距离：

$$L_2(x_i, x_j) = \Big(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^2\Big)^{\frac{1}{2}}$$

$p = \infty$ ：

$$L_\infty(x_i, x_j) = \max_l |x_i^{(l)} - x_j^{(l)}|$$

下图表示二维空间中p取不同值时，与原点的 $L_p$ 距离为1 ($L_p = 1$) 的点的图形， 那么显然不同的距离度量所确定的最近邻点是不同的。

### 1.2.3 k值的选择

k 值的选择会对k近邻法的结果产生重大影响。

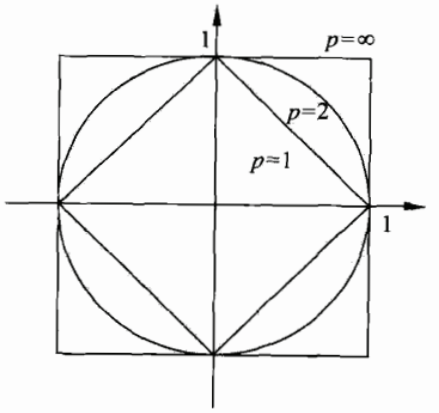



图 6: 距离度量

如果选择较小的k值，对相近的实例点非常敏感,容易产生过拟合；相反地如果选择较大的k值,不相近的点也会起预测作用，使预测发生错误。

在应用中k值一般取一个比较小的数值．通常采用交叉验证法来选取最优的k值。通常k值是不大于20的整数。

### 1.2.4 分类决策规则

k近邻法中的分类决策规则往往是多数表决，即由输入实例的k个邻近的训练实例中的多数类决定输入实例的类。事实上多数表决规则等价于经验风险最小化。

### 1.2.5 kd树

实现k近邻法时，主要考虑的问题是如何对训练数掘进行快速k近邻搜索，这点在特征空间的维数大及训练数据容量大时尤其必要。

k近邻法最简单的实现方法是线性扫描，这时要计算输入实例与每一个训练实例的距离当训练集很大时，计算非常娓时，这种方法是不可行的．为了提高k近邻搜索的效率，可以考虑使用特殊的数据结构存储训练数据，以减少计算距离的次数。kd树就是一种较好的实现。

关于kd树的实现不详细记录，但要知道如果实例点是随机分布的， kd树搜索的平均计算复杂度是 $O(logn)$ 的，这里N是训练实例数。kd树更适用于训练实例数远大于空间维数时的k近邻搜索,当空间维数接近训练实例数时，它的效率会迅速下降，几乎成线性。

虽然 kd树的方法对于低维度 ($D < 20$) 近邻搜索非常快，当D增长到很大时，效率变低：这就是所谓的“维度灾难” 的一种体现。

### 1.2.6 实战：实现k近邻

程序核心函数代码:

```
def createDataSet():
    group = np.array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return group, labels

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0] #training sample
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet #difference
    sqDiffMat = diffMat**2 #square
    sqDistances = sqDiffMat.sum(axis=1) #axis=1 row; 0 column
    distances = sqDistances**0.5 #root
    sortedDistIndicies = distances.argsort() #sort to get index
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1 # value
            default 0
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
        reverse=True) #iterator,Specified sort field,Descending order
    return sortedClassCount
```

结果

```
k=3
[('A', 2), ('B', 1)]
```

### 1.2.7 实战：改进约会网站

程序核心函数代码:

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0] #training sample
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet #difference
    sqDiffMat = diffMat**2 #square
    sqDistances = sqDiffMat.sum(axis=1) #axis=1 row; 0 column
    distances = sqDistances**0.5 #root
    sortedDistIndicies = distances.argsort() #sort to get index
```

```
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1 # value
            default 0
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
        reverse=True) #iterator,Specified sort field,Descending order
    return sortedClassCount[0][0]

def file2matrix(filename):
    fr = open(filename)
    numberOfLines = len(fr.readlines())     #get the number of lines in the file
    returnMat = np.zeros((numberOfLines,3))    #prepare matrix to return
    classLabelVector = []                   #prepare labels return
    fr = open(filename)
    index = 0
    for line in fr.readlines():
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector

def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = np.zeros(np.shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - np.tile(minVals, (m,1))
    normDataSet = normDataSet/np.tile(ranges, (m,1)) #element wise divide
    return normDataSet, ranges, minVals

def datingClassTest():
    hoRatio = 0.50    #hold out 10%
    datingDataMat,datingLabels = file2matrix('datingTestSet2.txt') #load data
        setfrom file
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
```

```
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:]\
        ,datingLabels[numTestVecs:m],3)
        #print ("the classifier came back with: %d, the real answer is: %d" %
            (classifierResult, datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print ("the total error rate is: %f" % (errorCount/float(numTestVecs)))
    print (errorCount)


if __name__=="__main__":
        datingClassTest()
```

结果:

```
k=3
the total error rate is: 0.066000
```

### 1.2.8 实战：手写字体的识别

程序核心函数代码:

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0] #training sample
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet #difference
    sqDiffMat = diffMat**2 #square
    sqDistances = sqDiffMat.sum(axis=1) #axis=1 row; 0 column
    distances = sqDistances**0.5 #root
    sortedDistIndicies = distances.argsort() #sort to get index
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1 # value
            default 0
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
        reverse=True) #iterator,Specified sort field,Descending order
    return sortedClassCount[0][0]

def handwritingClassTest():
```

```
    hwLabels = []
    trainingFileList = listdir('trainingDigits')      #load the training set
    m = len(trainingFileList)
    trainingMat = np.zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0] #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
    testFileList = listdir('testDigits')    #iterate through the test set
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0] #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print ("the classifier came back with: %d, the real answer is: %d" %
            (classifierResult, classNumStr))
        if (classifierResult != classNumStr): errorCount += 1.0
    print ("\nthe total number of errors is: %d" % errorCount)
    print ("\nthe total error rate is: %f" % (errorCount/float(mTest)))
```

结果:

```
the total number of errors is: 10
the total error rate is: 0.010571
Time For Run CompleteKNN:34.74240867341453s compared with SVM 9s
```

## 1.3 决策树

### 1.3.1 决策树的模型与学习

分类决策树模型是一种描述对实例进行分类的树形结构，决策树由结点和有向边组成．结点查两种类型：内部结点和叶结点；内部结点表示一个特证或属性，叶结点表示一个类。

用决策树分类，从根结点开始，对实例的某一特征进行测试，根据测试结果，将实例分配到其于结点。这时，每一个子结点对应着该特征的一个取值如此递归地对实例进行测试并分配，直至达到叶结点最后将实例分到叶结点的类中。 **事实上，决策树可以转换为if-then规则集合。并且决策树将将特征空间块状划分。**

决策树学习算法包含特征选择、决策树的生成与决策树的剪枝过程。由于决策树表示一个条件概率分布，所以深浅不同的决策树对应着不同复杂度的概率模型。决策树的生成对应于模型的局部选择，决策树的剪枝对应于模型的全局选择。决策树的生成只考虑局部最优，相对地，决策树的剪枝则考虑全局最优。

决策树学习常用的算法有ID3，C4.5，CART（Regression）。

### 1.3.2 特征选择

特征选择在于选取对训练数据具有分类能力的特征．这样可以提高决策树学习的效率．如果利用一个特征进行分类的结果与随机分类的结果没有很大差别，则称这个特征是没有分类能力的经验上扔掉这样的特征对决策树学习的精度影响不大．通常特征选择的准则是信息增益或信息增益比。

**熵** 在信息论与概率统计中，熵（entropy）是表示随机变量不确定性的度量。设X是一个取有限个值的离散随机变量，其概率分布为

$$P(X = x_i) = p_i$$

则随机变量 X 的熵定义为：

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i$$

从定义可以看出熵越大，随机变量的不确定性就越大。

**条件熵** 设有随机变量 $(X, Y)$ 其联合概率分布为

$$P(X = x_i, Y = y_j) = p_{ij}, \quad i, j = 1, 2, ..., m$$

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．随机变量 X 给定的条件下随机变量 Y 的条件熵(conditional entropy) $H(Y|X)$ ，定义为 X 给定条件

下 Y 的条件概率分布的条件熵对 X 的数学期望

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i)$$

这里， $p_i = P(X = x_i), \quad i = 1, 2..., n$

当熵和条件熵中的概率由数据估计〈特别是极大似然估计〉得到时，所对应的熵与条件熵分别称为经验熵（empirical entropy）和经验条件熵（empirical conditional entropy）.

**信息增益** 特征 A 对训练数据集 D 的信息增益 $g(D, A\Psi$ ，定义为集合D的经验熵 $H(D\Psi$ 与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即

$$g(D|A) = H(D) - H(D|A)$$

一般地，熵 $H(Y)$ 与条件熵 $H(Y|X)$ 之差称为互信息（mutual information）.决策树学习中的信息增益等价于训练数据集中类与特征的互信息，

**信息增益比** 信息增益值的大小是相对于训练数据集而言的，并没有绝对意义在分类问题困难时，也就是说在训练数据集的经验熵大的时候，信息增益值会偏大．反之，信息增益值会偏小，使用信息增益比＜information gain ratio）可以对这一问题进行校正．这是特征选择的另一准则。

特征 A 对训练数据集D的信息增益比 $g_R(D, A\Psi$ 定义为其信息增益 $g(D, A\Psi$ 与训练数据集 D 的经验熵H(D）之比:

$$g_R(D, A) = \frac{g(D, A)}{H(D)}$$

### 1.3.3 决策树的生成

**ID3** ID3算法的核心是在决策树各个结点上应用信息增益准则选择特征，递归地构建决策树．具体方法是：从根结点（root node）开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立于子节点；再对于结点递归地调用以上方法，构建决策树：直到所有特征的信息增益均很小或没有特征可以选择为止最后得到一个决策树，ID3相当于用极大似然法进行概率模型的选择。

**C4.5** C4.5算法与ID3算法相似， C4.5算法对ID3算法进行了改进，C4.S在生成的过程中，用信息增益比来选择特征。

### 1.3.4 决策树的剪枝

决策树生成算法递归地产生决策树，直到不能继续下去为止这样产生的树往往对训

练数据的分类很准确，但对未知的测试数据的分类却没有那么准确，即出现过拟告现象．过拟合的原因在于学习时过多地考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树解决这个问题的办法是考虑诀策树的复杂度，对已生成的决策树进行简化．

在决策树学习中将已生成的树进行简化的过程称为剪枝（pruning）.具体地，剪枝从已生成的树上载掉一些子树或叶结点，并将其根结点就父结点作为新的叶结点，从而简化分类树模型。

**损失函数** 决策树的剪枝往往通过极小化决策树整体的损失函数（loss function）或代价函数（cost function）来实现．设树 T 的叶结点个数为 $|T|$ , t是树T的叶结点，该叶节点有 $N_t$ 个样本点，其中k类样本点有 $N_{tk}$ 个，$k = 1, 2, ..., K$，$H_t(T)$ )为叶节点t上的经验熵，$\alpha$ 为参数，则决策树的损失函数可以定义为

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T|$$

其中经验熵为:

$$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

在损失函数中，将右端的第1项记作:

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

这时有

$$C_\alpha(T) = C(T) + \alpha|T|$$

$C(T)$表示模型对训练数据的预测误差，即模型与训练数据的拟合程度，$|T|$ 表示模型复杂度，参数 $\alpha \geq 0$ 控制两者之间的影响．较大的 $\alpha$ 促使选择较简单的模型（树〉，较小的 $\alpha$ 促使选择较复杂的模型（树〉。$\alpha = 0$意味着只考虑模型与训练数据的拟合程度，不考虑模型的复杂度剪枝，就是当 $\alpha$ 确定时，选择损失函数量小的模型，即损失函数最小的于树．当 $\alpha$ 值确定时，子树越大，往往与训练数据的拟合越好，但是模型的复杂度就越高，相反，子树越小，模型的复杂度就越低，但是往往与训练数据的拟合不好，损失函数正好表示了对两者的平衡可以看出，决策树生成只考虑了通过提高信息增益〈或信息增益比〉对训练数据进行更好的拟合．而决策树剪枝通过优化损失函数还考虑了减小模型复杂度。决策树生成学习局部的模型，而决策树剪枝学习整体的模型。

### 1.3.5 实战：决策树生成与绘图

**ID3** 利用ID3算法进行决策树的构建，与可视化。

程序核心函数代码:

```
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet: #the the number of unique elements and their
        occurance
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2) #log base 2
    return shannonEnt


def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]  #chop out axis used for splitting
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet


def chooseBestFeatureToSplit(dataSet): #data must be list and same length
    numFeatures = len(dataSet[0]) - 1  #the last column is used for the labels
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):     #iterate over all the features
        featList = [example[i] for example in dataSet]#create a list of all the
            examples of this feature
        uniqueVals = set(featList)    #get a set of unique values
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy #calculate the info gain; ie
            reduction in entropy
```

```
        if (infoGain > bestInfoGain):    #compare this to the best gain so far
            bestInfoGain = infoGain      #if better than current best, set to
                best
            bestFeature = i
    return bestFeature


def majorityCnt(classList): #exit majority voter
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(),
        key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def createTree(dataSet,label):
    labels = label.copy()
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList): #Recursive exit
        return classList[0]#stop splitting when all of the classes are equal
    if len(dataSet[0]) == 1: #stop splitting when there are no more features
        in dataSet
        return majorityCnt(classList) #Recursive exit
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]    #copy all of labels, so trees don't mess up
            existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,
            bestFeat, value),subLabels)
    return myTree


def classify(inputTree,featLabels,testVec):
    firstStr = list(inputTree.keys())[0]
```

```
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel
```

结果:

```
Data [[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
Tree {'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
```

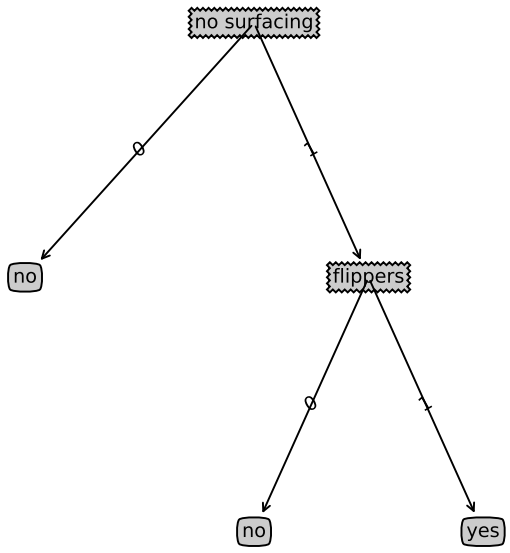
可视化

```
#visualization
#define textframe and arrow type
decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")

def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',
            xytext=centerPt, textcoords='axes fraction',
            va="center", ha="center", bbox=nodeType, arrowprops=arrow_args )


def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops) #no ticks
    #createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo puropses
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0;
    plotTree(inTree, (0.5,1.0), '')
    plt.show()
```

```
def plotTree(myTree, parentPt, nodeTxt):#if the first key tells you what feat
    was split on
    numLeafs = getNumLeafs(myTree) #this determines the x width of this tree
#    depth = getTreeDepth(myTree)
    firstStr = list(myTree.keys())[0] #the text label for this node should be
        this
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW,
        plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are
            dictonaires, if not they are leaf nodes
            plotTree(secondDict[key],cntrPt,str(key))    #recursion
        else:  #it's a leaf node print the leaf node
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt,
                leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD


def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are
            dictonaires, if not they are leaf nodes
            numLeafs += getNumLeafs(secondDict[key])
        else:  numLeafs +=1
    return numLeafs


def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
```

```
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are
            dictonaires, if not they are leaf nodes
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    return maxDepth


def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center",
        rotation=30)
```

结果:




### 1.3.6 实战：预测隐形眼镜类型

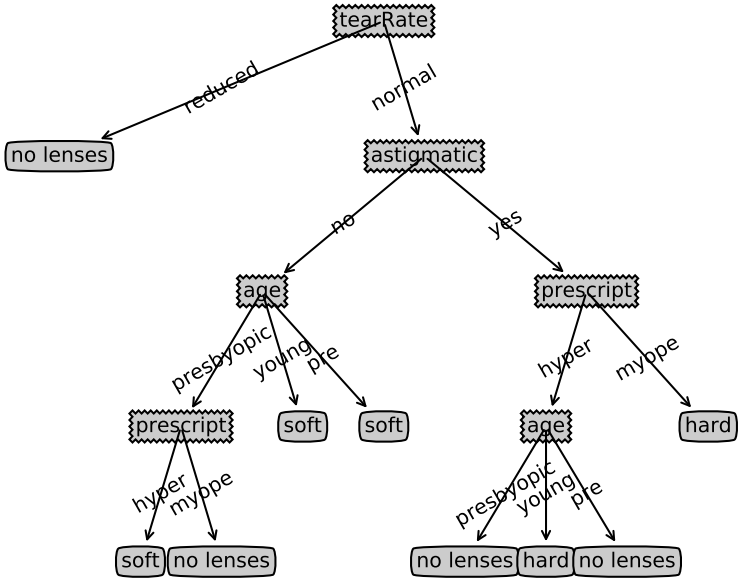
利用决策树构建一个预测隐形眼镜的专家系统。

数据集

```
young    myope   no reduced no lenses
young    myope   no normal  soft
```

```
young    myope  yes   reduced no lenses
young    myope  yes   normal  hard
young    hyper  no reduced no lenses
young    hyper  no normal  soft
young    hyper  yes   reduced no lenses
young    hyper  yes   normal  hard
pre          myope  no reduced no lenses
pre          myope  no normal  soft
pre          myope  yes   reduced no lenses
pre          myope  yes   normal  hard
pre      hyper  no reduced no lenses
pre          hyper  no normal  soft
pre      hyper  yes   reduced no lenses
pre     hyper yes  normal  no lenses
presbyopic myope no reduced no lenses
presbyopic myope no normal no lenses
presbyopic myope yes  reduced no lenses
presbyopic myope yes  normal  hard
presbyopic hyper no reduced no lenses
presbyopic hyper no normal soft
presbyopic hyper yes  reduced no lenses
presbyopic hyper yes  normal  no lenses
```

结果如下:



## 1.4 高斯判别算法

### 1.4.1 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。

### 1.4.2 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）

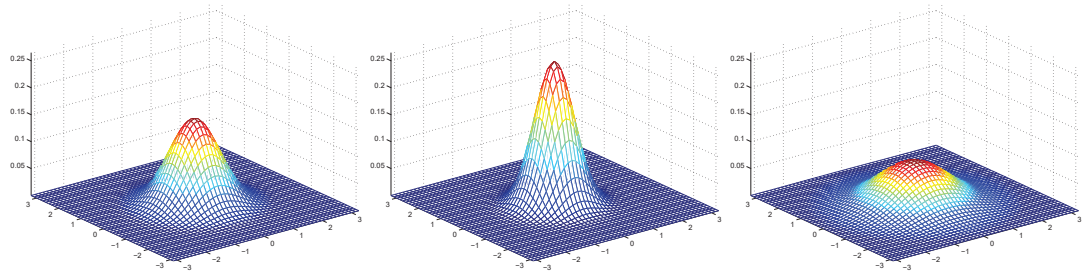


图 7: $\Sigma = I$ $\Sigma = 0.6I$ $\Sigma = 2I$

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

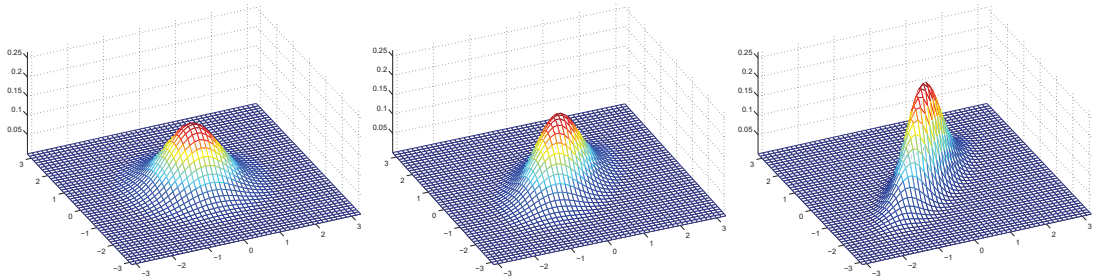


图 8: $\Sigma = I$ $\Sigma = I_1$ $\Sigma = I_2$

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

上图的等高线形式更能清晰可见：

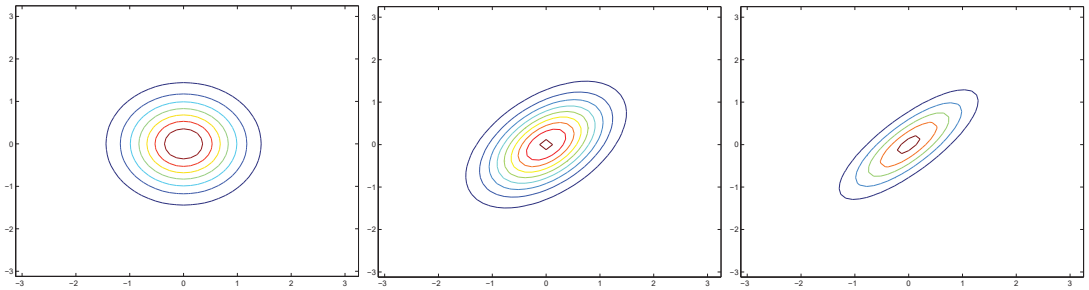


图 9: $\Sigma = I$ $\Sigma = I_1$ $\Sigma = I_2$

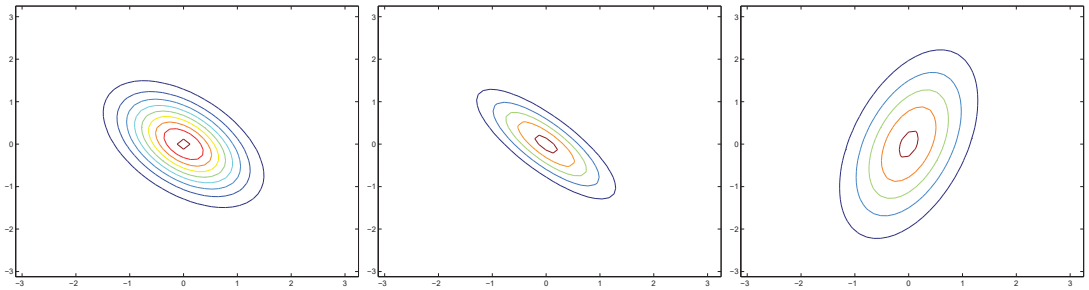


图 10: $\Sigma = I_3$ $\Sigma = I_4$ $\Sigma = I_5$

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

变均值，而不变协方差：$\Sigma = I$

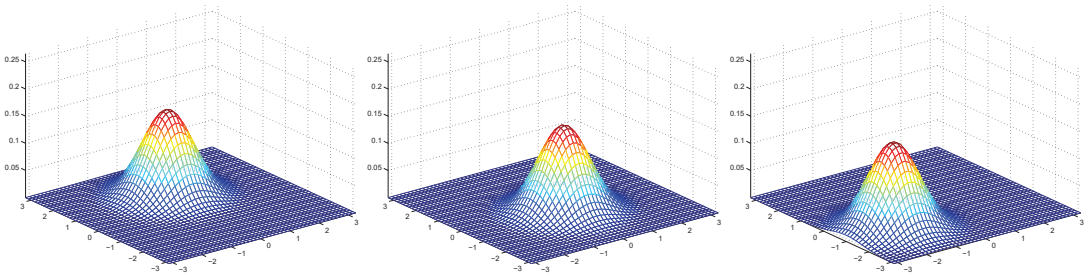


图 11: $\mu = \mu_1$ $\mu = \mu_2$ $\mu = \mu_3$

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

因此， $\mu$ 决定中心位置，而 $\Sigma$ 决定投影椭圆的朝向和大小。

### 1.4.3 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):

$\phi$ : 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。

$\Sigma$: 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:

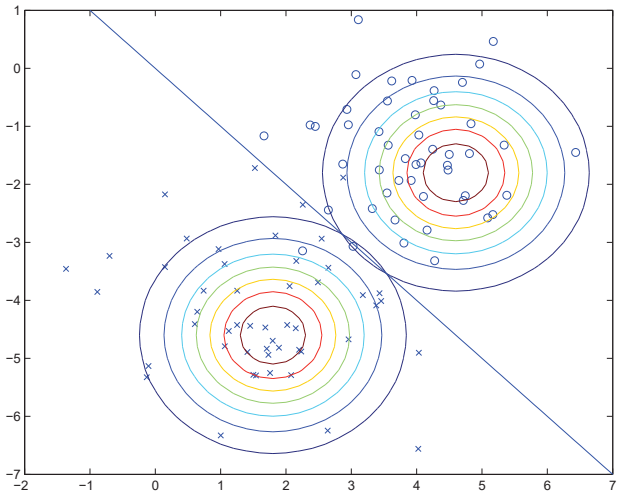


图 12: 高斯判别算法分类结果

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。 $\mu$ 值不同，所以位置不同。

### 1.4.4 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 $\mu_0$ 有关；第二部分只与 $\mu_1$ 有关；第三部分只 $\phi$ 有关。最大化该函数，1.首先先求 $\phi$ ，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 $\phi$，（其中 I 是指示函数），

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

2.同样地，对 $\mu_0$ 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 $\mu_1$ ，

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

4.对 $\Sigma$ 求偏导以求 $\Sigma$ (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。

### 1.4.5 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。

## 1.5 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。

### 1.5.1 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**

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

**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。

### 1.5.2 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**。

### 1.5.3 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 $\phi_y$ 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$\theta_0$)

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

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。

### 1.5.4 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。

### 1.5.5 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。

### 1.5.6 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。

### 1.5.7 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%。

## 1.6 感知机

感知机(perceptron)是二分类的线性分类模型，其输入为实例的特征向量,输出为实例的类别,取＋l和－1二值;感知机1957年Rosenbla提出,是神经网络与支持向量机的基础。

### 1.6.1 感知机模型，策略，学习算法

**感知机模型**的**输入到输出空间的函数**： $f(x) = SIGN(w \cdot x + b)$
感知机模型通过上述分隔超平面将特征空间分成两类，实现对数据的分类，属于判别学习算法。

假设训练数据集是线性可分的，**感知机学习的目标**是求得一个能够将训练集实例点和负实例点完全正确分开的分离超平面,为了找出这样的超平面，即确定感知机模型参数 w,b ，需要确定一个**学习策略**，即定义（经验）损失函数并将损失函数极小化。损失函数定义为

$$L(w,b) = -\sum_{i=1}^{F} y_i(w \cdot x_i + b)$$

事实上损失函数是训练集所有分错的训练样本的函数间隔和的相反数，至于什么是函数间隔会在支持向量机中介绍。显然，损失函数是非负的，如果没有误分类点，损失函数值是 0 。而且，误分类点越少，误分类点离超平面越远，损失函数值就越小。一个特定的样本点的损失函数在误分类时是参数 w,b 的线性函数，在正确分类时是 0 .因此，给定训练数据集 T ，损失函数L的是 w,b 的连续可导函数。

**感知机学习算法** 感知机学习问题转化为求解损失函数式的最优化问题，最优化的方法是随机梯度降法，学习算法包括原始形式和对偶形式。最后给出了在训练数据钱性可分条件下感知机学习算法的收敛性定理。

感知机的学习算法是由错误分点驱动的。具体的算法采用随机梯度下降算法。关于随机梯度算法不做关注。假设误分类点集合 M 是固定的，那么损失函数的梯度是

$$\nabla_w L(w,b) = -\sum_{x_i \in M} y_i x_i$$

$$\nabla_b L(w,b) = -\sum_{x_i \in M} y_i$$

随机选取一个误分类点 $(x_i, y_i)$ 对 $w, b$ 进行更新

$$w \leftarrow w + \eta y_i x_i$$

$$b \leftarrow b + \eta y_i$$

式中 $\eta$ 是步长，在统计学习中又称为学习率(learning rate)。

**对偶形式** 逐步修改 $w, b$ ,设修改 n 次,则 $w, b$ 关于 $(x_i, y_i)$ 的增量分别是 $\alpha_i y_i x_i$ 和 $\alpha_i y_i$ ，这里 $\alpha_i = n\eta_i$ 这样，从学习过程不难看出，最后学习到的 $w, b$ 可以分别表示为

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i$$
$$b = \sum_{i=1}^{m} \alpha_i y_i$$

这里， $\alpha_i \geq 0, i = 1, 2, 3, ..., m$ ,当 $\eta = 1$ 时，表示第 i 个实例点由于误分而进行更新的次数．实例点更新次数越多，意味着它距离分离超平面越近，也就越难正确分类，换句话说，这样的实例对学习结果影响最大。

### 1.6.2 算法的收敛性定理

**定理** 设训练集数据集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ 实现线性可分的，其中 $y_i \in \chi = R^n, y_i \in \{-1, +1\}, i = 1, 2, ..., m$ ，则

① 存在满足条件 $||\hat{w}_{opt}|| = 1$ 的超平面 $\hat{w}_{opt} \cdot \hat{x}_i = w_{opt} x_i + b = 0$ 将训练数据集完全正确分开;且存在 $\gamma > 0$ ，对所有 $i = 1, 2, 3, .., m$

$$y_i(\hat{w}_{opt} \cdot \hat{x}_i) = y_i(w_{opt} x_i + b) \geq \gamma$$

② 令 $R = \max\limits_{1 \leq i \leq m} ||x_i||$ ,则原始问题感知机算法在训练集上的误差分类次数k满足不等式

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

定理表明，误分类的次数 k 是有上界的，经过有限次搜索可以找到将训练数据完全正确分开的分离超平面，也就是说，当训练数据集线性可分时，感知机学习算法原始形式法代是收敛的。但是感知机学习算法存在许多解，这些解既依赖于初值的选择和误分类点的选择顺序。为了得到唯一的超平面，需要对分离超平面增加约束条件，这就是线性支持向量机的想法。当训练集线性不可分时，感知机学习算法不收敛，迭代结果会发生震荡。

### 1.6.3 实战：实现两种形式的感知机算法

**原始问题的实现**

程序核心函数代码:

```
def trainPerceptron(dataMat, labelMat, eta):
```

```
    m, n = np.shape(np.mat(dataMat))
    weight = np.zeros(n)
    bias = 0
    flag = True
    while flag:
        for i in range(m):
            if np.any(labelMat[i] * (np.dot(weight, dataMat[i]) + bias) <= 0):
                weight = weight + eta * np.dot(labelMat[i],dataMat[i])
                bias = bias + eta * labelMat[i]
                print("weight, bias: ", end="")
                print(weight, end=" ")
                print(bias)
                flag = True
                break
            else:
                flag = False
    return weight, bias
```

运行结果可视化图:

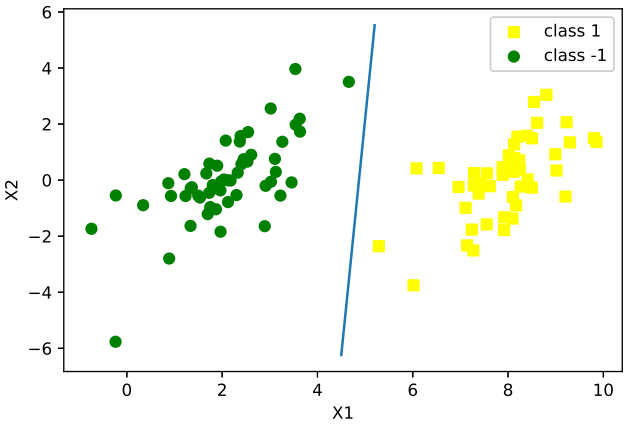



图 13: 原始优化运行结果图

**对偶问题的实现**

程序核心函数代码:

```
def trainModel(dataMatIn,labelMatIn,eta):
    dataMat = np.mat(dataMatIn); labelMat = np.mat(labelMatIn).transpose()
    b = 0; m,n = np.shape(dataMat)
    alpha = np.mat(np.zeros((m,1)))
```

```
    flag = True
    while flag:
        for i in range(m):
            if  labelMatIn[i]*(float(np.multiply(alpha,labelMat).\
            T*(dataMat*dataMat[i,:].T)) + b)<= 0:
                alpha[i] = alpha[i] + eta
                b = b + eta * labelMat[i]
                w = np.multiply(labelMat,alpha).T*dataMat
                print (i,alpha[i],w,b)
                flag = True
                break
            else:
                flag = False
    w = (np.multiply(labelMat,alpha).T*dataMat).T
    return w,b
```

运行结果可视化图:

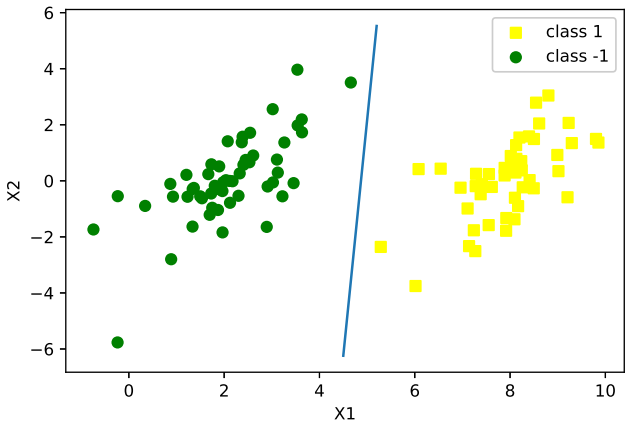



图 14: 对偶优化运行结果图

因为原始问题与对偶问题具有相同的初值以及相同训练集样本选择顺序，因此具有相同的结果。经过对感知机原始问题与对偶问题的实现我们可以有对支持向量机更好的理解。

## 1.7 支持向量机

### 1.7.1 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。

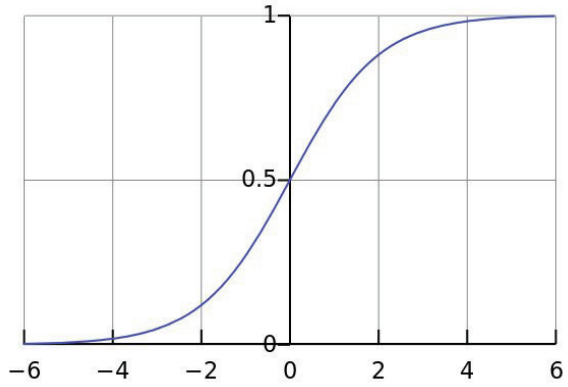



图 15: 逻辑函数

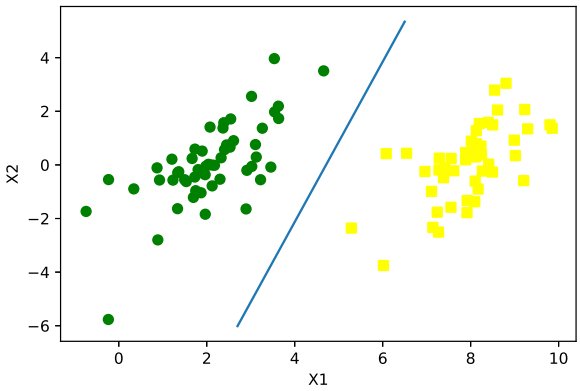



图 16: 线性可分模型构想

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）。

### 1.7.2 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

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

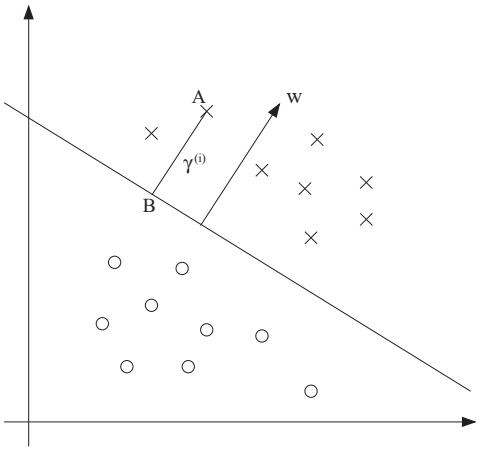



图 17: 几何间隔

### 1.7.3 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

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

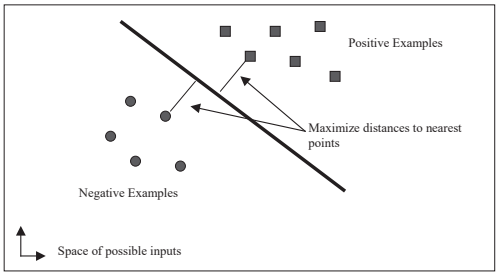



图 18: 支持向量机

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。

### 1.7.4 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:

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

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,

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

**原始问题**

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

若 $g_i(w) > 0$ 或者 $h_i(w) \ne 0$ ,有

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

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

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

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

**定理** $d^* = \max_{\alpha \ge 0, \beta} \min L(w, \alpha, \beta) \le \min_{w} \max_{\alpha, \beta, \alpha_i \ge 0} L(w, \alpha, \beta) = p^*$

即就是: $d^* \le p^*$

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. $g_i(w^*)\le 0,\ \ i=1,...,k$
5. $\alpha_i\ge 0,\ \ i=1,...,k$

由3可知，　当 $\alpha_i>0$ 时，$g_i(w^*)=0$

　　当 $\alpha_i\ne 0$ 时，$g_i(w^*)=0$

### 1.7.5 利用对偶问题求解最优间隔分类器

利用拉格朗日对偶求得原始问题的对偶问题，再解决对偶问题实现对原始问题的求解。这样做的好处是求解容易以及便于向非线性扩展。

**原始问题**

$$\min_{w,b}\ \frac{1}{2}||w||^2$$

$$s.t.\ y_i(w\cdot x_i+b)-1\ge 0,\ \ i=1,2,...,m$$

**对偶问题**

$$L(w,b,\alpha)\ =\ \frac{1}{2}||w||^2-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}(w^Tx^{(i)}+b)-1\right]$$

$$\theta_D(\alpha)\ =\ \min_{w,b}L(w,b,\alpha)$$

求解：① 先最小化lagrangian. $L(w,b,\alpha)$

$$\nabla_w L(w,b,\alpha)=w-\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}=0$$

$$w=\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}$$

$$\frac{\partial}{\partial b}L(w,b,\alpha)=\sum_{i=1}^{m}\alpha_i y^{(i)}=0$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)}=0$$

② 代入到原式:

$L(w,b,\alpha)$

$$
\begin{aligned}
&= \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[ y^{(i)}(w^T x^{(i)} + b) - 1 \right] \\
&= \frac{1}{2} w w^T - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} w^T - \sum_{i=1}^{m} \alpha_i y^{(i)} b + \sum_{i=1}^{m} \alpha_i \\
&= \frac{1}{2} w w^T - w w^T - \sum_{i=1}^{m} \alpha_i y^{(i)} b + \sum_{i=1}^{m} \alpha_i \\
&= -\frac{1}{2} \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} (\sum_{j=1}^{m} \alpha_i y^{(j)} x^{(j)})^T - b \sum_{i=1}^{m} \alpha_i y^{(i)} + \sum_{i=1}^{m} \alpha_i \\
W(\alpha) &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} < x^{(i)}, x^{(j)} >
\end{aligned}
$$

得到最终的对偶优化问题:

$$
\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} < x^{(i)}, x^{(j)} > - \sum_{i=1}^{m} \alpha_i
$$

$$
s.t. \ \ \alpha_i \geq 0, \ \ i = 1, ..., m
$$

$$
\sum_{i=1}^{m} \alpha_i y^{(i)} = 0
$$

最终要解决此对偶问题，1998年John.C.platt发表了一篇论文 "Sequential Minimal Optimization:A Fast Algorithm For Training Support Vector Machines" or SMO algorithm 很好的解决了这个优化问题。

### 1.7.6 软间隔支持向量机

在此之前所有的讨论都是建立在数据明显线性可分下进行的，接下来要介绍一种 SVM 的变化形式来解决允许少量错误分点的情况，就是引入 L1 范式的正则化项（松弛变量）。这样做的好处是防止模型过拟合，使模型更稳定。

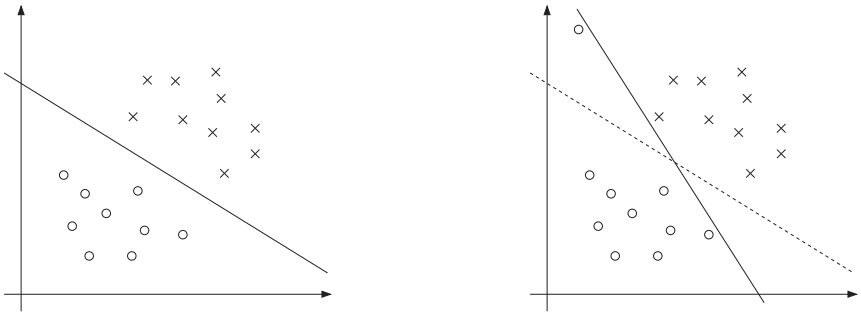


图 19: 软间隔支持向量机

**原始问题加入正则化项**

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

**得到对偶问题**

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)} < x^{(i)}, x^{(j)} > -\sum_{i=1}^{m}\alpha_i$$
$$s.t. \ \ 0 \leq \alpha_i \leq C, \ \ i = 1, ..., m$$
$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

**由KKT条件可知**

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 \leq \alpha_i \leq C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1 \end{aligned}$$

### 1.7.7 引入核函数

分类数据从线性可分，有少量早噪声点，我们可以利用 SVM 或者 SVM 的变形来解决；但大多数的分类数据是线性不可分的，此时我们就要引入核函数，利用核技巧来解决这一线性不可分的情况。

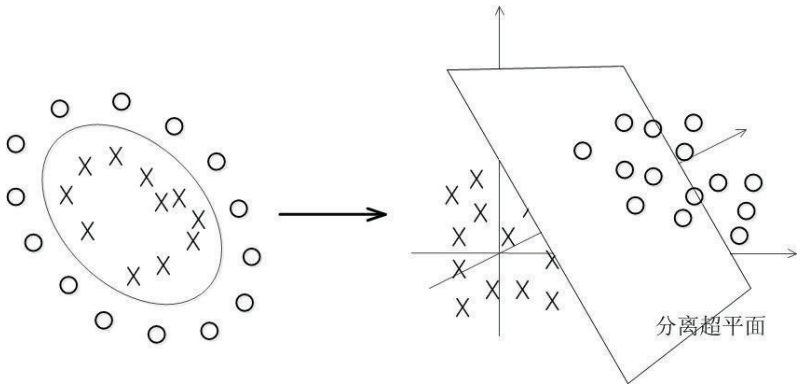



图 20: 核技巧

**核**

特征映射 feature mapping.

一维特征 $x$ $\rightarrow$ 三维特征 $\Phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$

利用特征映射来代替对偶问题中的内积，是一种好的方法，但是 $\Phi(x)$ 一般维度很高甚至是无限维，这就导致利用 $< \Phi(x^{(i)}), \Phi(x^{(j)}) >$ 代替 $< x^{(i)}, x^{(j)} >$ 代价非常大。如此，便引入了核函数。

**核函数**

举例，核函数及对应的核：

$$
\begin{aligned}
& ① \ x, z \in R^n \\
K(x,z) &= (x^T z)^2 \\
&= (\sum_{i=1}^{n} x_i z_i)(\sum_{j=1}^{n} x_j z_j) \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \qquad O(n)
\end{aligned}
$$

$$
\Phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix} \qquad O(n^2)
$$

② $K(x,z) = (x^T z + c)^d \qquad O(n)$

$\Phi(x)$ 的维度与的相关。 $\qquad O(n^d)$

③ 径向基函数(radial basis function)

$$
K(x,y) = \exp\left(\frac{-||x-y||^2}{2\sigma^2}\right) \qquad O(n)
$$

$\Phi(x)$ 无限维 $\qquad O(\infty)$

事实上，对于任意给定的核函数，是否一定存在原始的核，答案显然是不一定的。那其中的关系由 Mercer 定理给出。

**Mercer 定理** 任何半正定对称函数都可以作为核函数。

K 是一个 Mercer 核，其充要条件是 $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ 所对应的核矩阵是对称半正定矩阵。

**引入高斯核得到新的对偶优化问题**

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}K < x^{(i)}, x^{(j)} > - \sum_{i=1}^{m}\alpha_i$$

$$s.t.\ 0 \le \alpha_i \le C,\ \ i = 1, ..., m$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

得到 , $\alpha^*, w^*$ , $b^*$

构造决策函数: $f(x) = sign\left(\sum_{i=1}^{m}\alpha_i^* y_i K(x, x_i) + b^*\right)$

最终的问题在于如何求得最终优化问题的最优解，这便是利用 SMO 算法。

### 1.7.8 SMO 算法

SOM 算法全称 Sequential Minimal Optimization. 是1998年4月 John.C.platt 发表的关于训练支持向量机一种快速的算法。它相较于其他训练支持向量机的算法有明显的速度优势。它的灵感来源于坐标上升算法 (Coordinate ascent) .

**坐标上升**

*Repeat*{

$for\ i = 1\ to\ m${

$\alpha_i := arg\ \max_{\alpha_i} W(\alpha_1, .., \alpha_{i-1}, \alpha_i, \alpha_{i+1}, ..., \alpha_m)$

}

}

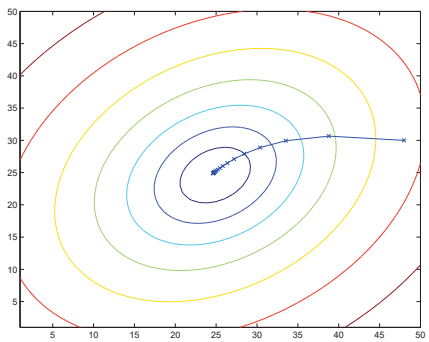


图 21: 梯度下降

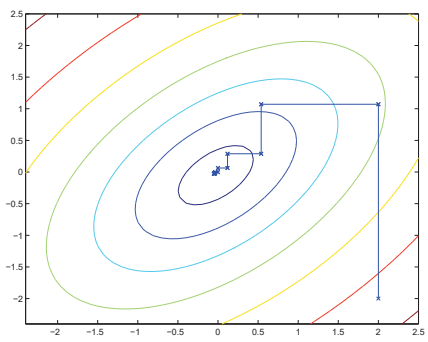


图 22: 坐标上升

坐标上升相较于其他最优化算法，如牛顿法来讲，是一种“曲线救国”的形式。虽然它收敛时的迭代次数要远多于牛顿法，但是对于单变量的优化问题它拥有明确的解析解，它的执行速度是非常快的。

利用坐标上升的思想来解决支持向量机的对偶问题就是SMO算法的实质，但是坐标上升的基本形式不能直接用于 SVM 的对偶优化问题，应为它有一个关于 $\alpha$ 的等式约束。所以 SMO 算法每次优两个 $\alpha$ 变量。

**SMO 算法框架**

框架:

选择 $\alpha_i, \alpha_j$ 。(经验法则亦或者是启发式方法)

固定其他所有 $\alpha_i$ 。

同时使得 $\alpha_i, \alpha_j$ 最优。 （核心步骤）

**核心步骤的数学推导**

对偶优化问题:

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}K < x^{(i)}, x^{(j)} > -\sum_{i=1}^{m}\alpha_i$$

$$s.t.\ 0 \le \alpha_i \le C,\ \ i = 1, ..., m$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

依据坐标上升思想，控制变量剩两个得到新的优化问题:

$$\begin{aligned}
&\min_{\alpha_1,\alpha_2} \Psi(\alpha_1, \alpha_2) \\
&= \min_{\alpha_1,\alpha_2} \frac{1}{2}K_{11}\alpha_1^2 + \frac{1}{2}K_{22}\alpha_2^2 + y_1y_2\alpha_1\alpha_2 \\
&\qquad - (\alpha_1 + \alpha_2) + y_1v_1\alpha_1 + y_2v_2\alpha_2 + Constant \qquad (1) \\
&\quad v_i = \sum_{j=3}^{m}\alpha_i\alpha_j K(x_i, x_j) \\
&\quad s.t.\ \ 0 \le \alpha_1, \alpha_2 \le C \\
&\quad \alpha_1y_1 + \alpha_2y_2 = -\sum_{i=3}^{m}\alpha_i y_i = \xi \\
&\Rightarrow \ \ \alpha_1 = (\xi - \alpha_2y_2)y_1 \qquad (2)
\end{aligned}$$

将(2)代入(1)得到一元二次函数的优化问题:

$$\begin{aligned}
\min_{\alpha_2}\phi(\alpha 2) &= \frac{1}{2}K_{11}((\xi - \alpha_2y_2)y_1)^2 + \frac{1}{2}K_{22}\alpha_2^2 + y_1y_2(\xi - \alpha_2y_2)y_1\alpha_2 \\
&- ((\xi - \alpha_2y_2)y_1 + \alpha_2) + y_1v_1(\xi - \alpha_2y_2)y_1 + y_2v_2\alpha_2 + Constant \qquad (3)
\end{aligned}$$

利用微分求这个最小化问题的最优解:

$$\frac{\partial\phi(\alpha_2)}{\partial\alpha_2} = (K_{11}+K_{22}-K_{12})\alpha_2 - K_{11}\xi y_2$$
$$+ K_{11}\xi y_2 + y_1y_2 - 1 - v_1y_2 + v_2y_1 = 0$$

令求得的 $\alpha_1, \alpha_2$ 为 $\alpha_1^{new}, \alpha_2^{new}$ . 未更新的 $\alpha_1, \alpha_2$ 为 $\alpha_1^{old}, \alpha_2^{old}$ .

由于等式约束条件 $\sum_{i=1}^{m} y^{(i)}\alpha_i = 0$ 有:

$$\alpha_1^{old}y_1 + \alpha_2^{old}y_2 = -\sum_{i=3}^{m}\alpha_i y_i = \xi = \alpha_1^{new}y_1 + \alpha_2^{new}y_2$$
$$\Rightarrow \xi = \alpha_1^{old}y_1 + \alpha_2^{old}y_2 \tag{4}$$

真实值与预测值之间的误差:

$$E_i = f(x_i) - y_i \tag{5}$$
$$f(x) = \sum_{i=1}^{N}\alpha_i y_i K(x_i, x) + b$$

由于 $v_i = \sum_{j=3}^{m}\alpha_i\alpha_j K(x_i, x_j)$ 因此:

$$v_1 = f(x_1) - \sum_{j=1}^{2}\alpha_j y_j K_{1j} - b \tag{6}$$

$$v_2 = f(x_2) - \sum_{j=1}^{2}\alpha_j y_j K_{2j} - b \tag{7}$$

将(4)(6)(7)代入求导式:

$$(K_{11}+K_{22}-2K_{12})\alpha_2 - (\alpha_1^{old}y_1 + \alpha_2^{old}y_2)y_2(K_{11}-K_{22}) + y_1y_2 - y_2^2$$
$$+ y_2(f(x_2) - f(x_1) + \alpha_1^{old}y_1k_{11} + \alpha_2^{old}y_2K_{12} - \alpha_1^{old}y_1K_{12} - \alpha_2^{old}y_2K_{22}) = 0$$
$$\Rightarrow (K_{11}+K_{22}-2K_{12})\alpha_2 = \alpha_2^{old}(K_{11}+K_{22}-2K_{12}) + y_2(f(x_1) - f(x_2) - y_1 + y_2)$$

令 $\eta = (K_{11}+K_{22}-2K_{12}) = ||\phi(x_1) - \phi(x_2)||^2$

$$\Rightarrow \alpha_2^{new,unclipped} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta} \tag{8}$$

由于求得的 $\alpha_2^{new,unclipped}$ 是未利用不等式约束的解析解，我们进一步依据不等式约束 $0 \le \alpha_{1,2} \le C$ 和等式 $\alpha_1^{new}y_1 + \alpha_2^{new}y_2 = \xi$ 来进行裁剪，使求得的 $\alpha_{1,2}$ 都满足约束。

裁剪 情况1:

$$y_1 \ne y_2 \Rightarrow \alpha_1 - \alpha_2 = K$$
$$L = \max(0, -K) = \max(0, \alpha_2^{old} - \alpha_1^{old})$$
$$H = \min(0, C - K) = \min(0, C + \alpha_2^{old} - \alpha_1^{old})$$

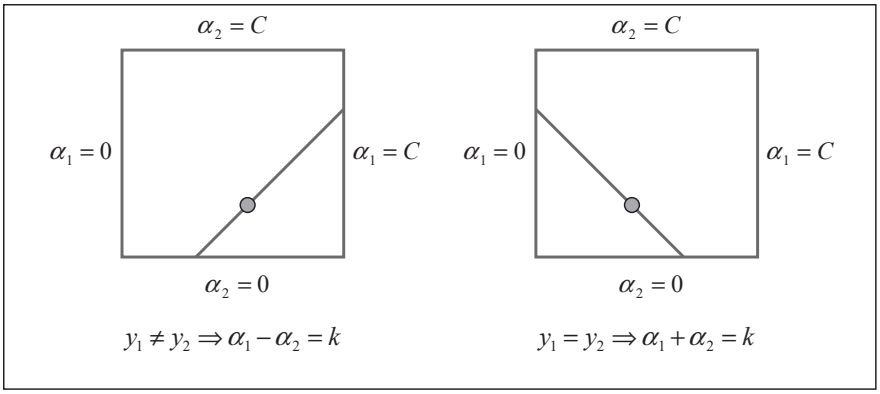



图 23: 裁剪

情况2:

$$y_1 = y_2 \Rightarrow \alpha_1 + \alpha_2 = k$$
$$L = \max(0, K - C) = \max(0, \alpha_2^{old} + \alpha_1^{old} - C)$$
$$H = \min(K, C) = \min(0, c + \alpha_2^{old} - \alpha_1^{old})$$

裁剪后:

$$\alpha_2^{new,clipped} = \begin{cases} H & , \alpha_2^{new,unclipped} > H \\ \alpha_2^{new,unclipped} & , L \leq \alpha_2^{new,unclipped} \leq H \\ L & , \alpha_2^{new,unclipped} > L \end{cases}$$

求解 $\alpha_1$ :

$$\alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \alpha_1^{new} y_1 + \alpha_2^{new} y_2$$
$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new}) \tag{9}$$

特殊情况: ① $\eta < 0$ ,核矩阵为非半正定矩阵。

② $\eta = 0$ ,样本 $x_1, x_2$ 输入特征相同。

⇒这两种情况下，都是在边界上取得极值。

则将 $\alpha_2^{new} = L$ , $\alpha_2^{new} = H$

代入(9)式得 $\alpha_1^{new} = L_1$ , $\alpha_1^{new} = H_1$ , $s = y_1 y_2$.

$\Rightarrow L_1 = \alpha_1^{old} + s(\alpha_2^{old} - L)$

$H_1 = \alpha_1^{old} + s(\alpha_2^{old} - H)$

将代入(1)式中比较 $\Psi(\alpha_1 = L_1, \alpha_2 = L)$ 与 $\Psi(\alpha_1 = H_1, \alpha_2 = H)$ 取小得特殊情况的解析解。

$$
\begin{aligned}
\Psi_L &= \Psi(L_1, L) = \Psi(\alpha_1 = L_1, \alpha_2 = L) \\
&= \frac{1}{2}L_1^2K_{11} + \frac{1}{2}L_2^2K_{22} + sK_{12}LL_1 - (L_1 + L) + y_1v_1L_1 + y_2v_2L_2 + Constant \\
&= ... + L_1(y_1v_1 - 1) + L_2(y_2v_2 - 1) + Constant \\
&= ... + L_1(y_1[f(x_1) - \alpha_1y_1K_{11} - \alpha_2y_2K_{12} - b] - 1) + L_2(y_2v_2 - 1) + Constant \\
&= ... + L_1([f(x_1) - y_1 + y_1 - b - \alpha_1y_1K_{11} - \alpha_2y_2K_{12}] - 1) + L_2(y_2v_2 - 1) + Constant \\
&= ... + L_1(y_1(E_1 - b) + 1 - \alpha_1K_{11} - y_1y_2K_{12}\alpha_2 - 1) + L_2(y_2v_2 - 1) + Constant \\
&= \frac{1}{2}L_1^2K_{11} + \frac{1}{2}L_2^2K_{22} + sK_{12}LL_1 + L_1f_1 + L_2f_2 + Constant \\
&\qquad f_1 = y_1(E_1 - b) - \alpha_1K_{11} - y_1y_2\alpha_2K_{12} \\
&\qquad f_2 = y_2(E_2 - b) - \alpha_2K_{22} - y_1y_2\alpha_1K_{12} \\
\Psi_H &= \Psi(H_1, H) = \Psi(\alpha_1 = H_1, \alpha_2 = H) \\
&= \frac{1}{2}H_1^2K_{11} + \frac{1}{2}H_2^2K_{22} + sK_{12}HH_1 + H_1f_1 + H_2f_2 + Constant
\end{aligned}
$$

计算阈值 b 和差值 $E_i$

① 若 $0 < \alpha_1^{new} < C$ , $\alpha_2^{new} = 0 \; or \; C \Rightarrow \sum_{i=1}^{N} \alpha_i y_i K_{i1} + b = y_1$

$$\Rightarrow b_1^{new} = y_1 - \sum_{i=3}^{m} \alpha_i y_i K_{i1} - \alpha_1^{new} y_1 K_{11} - \alpha_2^{new} y_2 K_{21}$$

因 $E_1 = f(x_1) - y_1 = \sum_{i=3}^{m} \alpha_i y_1 K_{i1} + \alpha_1^{old} y_1 K_{11} + \alpha_2^{old} y_2 K_{21} + b^{old} - y_1$

$$\Rightarrow b^{new} = b_1^{new} = -E_1 - y_1K_{11}(\alpha_1^{new} - \alpha_1^{old}) - y_2K_{21}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

② 同理 $0 < \alpha_2^{new} < C$ , $\alpha_1^{new} = 0 \; or \; C$

$$\Rightarrow b^{new} = b_2^{new} = -E_2 - y_1K_{12}(\alpha_1^{new} - \alpha_1^{old}) - y_2K_{22}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

③ 若 $0 < \alpha_1^{new} < C$ , $0 < \alpha_2^{new} < C \Rightarrow b^{new} = b_1^{new} = b_2^{new}$

④ 若 $\alpha_1^{new} = 0 \; or \; C, \; \alpha_2^{new} = 0 \; or \; C \Rightarrow b^{new} = \frac{1}{2}(b_1^{new} + b_2^{new})$

变量的选取方法

SMO 算法在每个子问题中选择两个变量优化，其中至少一个变量是违反 KKT条件的。

①第一个变量的选取方法:

SMO 称选择第1个变量的过程为外层循环外层循环在训练样本中选取违反 KKT 条件最严重的样本点,并将其对应的变量作为第 1个变量.具体地,检验训练样本点 $(x_i, y_i)$ 是否满足 KKT 条件，即

$$\begin{aligned} \alpha_i = 0 \quad &\Leftrightarrow \quad y_i g(x_i) \geq 1 \\ \alpha_i = C \quad &\Leftrightarrow \quad y_i g(x_i) \leq 1 \\ 0 \leq \alpha_i \leq C \quad &\Leftrightarrow \quad y_i g(x_i) = 1 \end{aligned}$$

其中， $g(x_i) = \sum_{j=1}^{m} \alpha_j y_j K(x_i, x_j) + b$ .

该检验是在 $\epsilon$ 范围内进行的。在检验过程中，外层循环首先遍历所有满足条件 $0 < \alpha_i < C$ 的样本点，即在间隔边界上的支持向盘点，检验它们是否满足 KKT 条件如果这些样本点都满足 KKT 条件，那么遍历整个训练集，检验它们是否满足 KKT 条件.

②第2个变量的选择

SMO 称选择第2个变量的过程为内层循环。假设在外层循环中已经找到第1个变量叫 $\alpha_1$ ，现在要在内层循环中找第2个变量 $\alpha_2$ 第2个变量选择的标准是希望能使 $\alpha_2$ 有足够大的变化。

由式(8)可知， $\alpha_2^{new}$ 是依赖于 $|E_1 - E_2|$ 的，为了加快计算速，一种简单的做法是选择 $\alpha_2$ ，使其对应的 $|E_1 - E_2|$ 最大。因为 $\alpha_1$ 已定， $E_1$ 也就确定了。如果 $E_1$ 是正的，那么选择最小的 $E_i$ 作为 $E_2$ ，如果 $E_i$ 是负的，那么选择最大的 $E_i$ 作为 $E_2$ ,为了节省计算时间，将所有 $E_i$ 保存在一个列表中。

在特殊情况下，如果内层循环通过以上方法选择的 $\alpha_2$ 不能使目标函数有足够的下降，那么采用以下启发式规则继续选择 $\alpha_2$ ·遍历在间隔边界上的支持向量点，依次将其对应的变量作为 $\alpha_2$ 试用，直到目标函数有足够的下降若找不到合适的 $\alpha_2$ 那么遍历训练数据集;若仍找不到合适的 $\alpha_2$ ，则放弃第1个 $\alpha_1$ ，再通过外层寻找 $\alpha_1$ 。

### 1.7.9 实战：简化版的SMO算法处理小规模数据集

支持向量机优缺点

优点：泛化错误率低，计算开销不大，结果易解释。

缺点：泛化错误率低，计算开销不大，结果易解释。

适用数据类型：数值型和标称型数据。

SVM的一般流程

(1) 收集数据：可以使用任意方法。

(2) 准备数据：需要数值型数据。

(3) 分析数据：有助于可视化分隔超平面。

(4) 训练算法：SVM的大部分时间都源自训练，该过程主要实现两个参数的调优。

(5) 测试算法：十分简单的计算过程就可以实现。

(6) 使用算法：几乎所有分类问题都可以使用SVM，值得一提的是，SVM本身是一个二类分类器，对多类问题应用SVM需要对代码做一些修改。

程序核心函数代码：

```
def selectJrand(i,m):
    j=i #we want to select any J not equal to i
    while (j==i):
        j = int(np.random.uniform(0,m))
    return j

def clipAlpha(aj,H,L): #s.t. 0<=\alpha<=C
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    dataMatrix = np.mat(dataMatIn); labelMat = np.mat(classLabels).transpose()
    b = 0; m,n = np.shape(dataMatrix)
    alphas = np.mat(np.zeros((m,1)))
    iter = 0
    while (iter < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            fXi = float(np.multiply(alphas,labelMat).T\
            *(dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])
            #if checks if an example violates KKT conditions
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C))
            or ((labelMat[i]*Ei > toler) and (alphas[i] > 0)):
                j = selectJrand(i,m)
                print("i",i,alphas[i])
                print("j",j,alphas[j])
                fXj = float(np.multiply(alphas,labelMat).T*\
                (dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
                if (labelMat[i] != labelMat[j]):
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
```

```
            else:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            if L==H: print ("L==H"); continue
            eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T -\
            dataMatrix[i,:]*dataMatrix[i,:].T -
                dataMatrix[j,:]*dataMatrix[j,:].T
            if eta >= 0: print ("eta>=0"); continue
            alphas[j] -= labelMat[j]*(Ei - Ej)/eta
            alphas[j] = clipAlpha(alphas[j],H,L)
            if (abs(alphas[j] - alphaJold) < 0.00001):
                print ("j not moving enough",alphas[j])
                continue
            alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])
            print("i",i,alphas[i])
            print("j",j,alphas[j]);
            #update i by the same amount as j
            #the update is in the oppostie direction
            b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)\
            *dataMatrix[i,:]*dataMatrix[i,:].T-labelMat[j]\
            *(alphas[j]-alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
            b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)\
            *dataMatrix[i,:]*dataMatrix[j,:].T-labelMat[j]\
            *(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
            if (0 < alphas[i]) and (C > alphas[i]): b = b1
            elif (0 < alphas[j]) and (C > alphas[j]): b = b2
            else: b = (b1 + b2)/2.0
            alphaPairsChanged += 1
            print ("iter: %d i:%d, pairs changed %d" %
                (iter,i,alphaPairsChanged))
        if (alphaPairsChanged == 0): iter += 1
        else: iter = 0
        print ("iteration number: %d" % iter)
    return b,alphas

def test():
    dataArr,labelArr = svmMLiA.loadDataSet('testSet.txt')
    b,alpha = svmMLiA.smoSimple(dataArr,labelArr,0.6,0.001,80)
    print (b)
    print (alpha[alpha>0])
```

```
    for i in range(100):
        if alpha[i]>0.0:
            print (i,dataArr[i],labelArr[i])
```

实验结果运行时间：

```
Time For Run SMO:15.935288698252556s
```

运行结果可视化图：

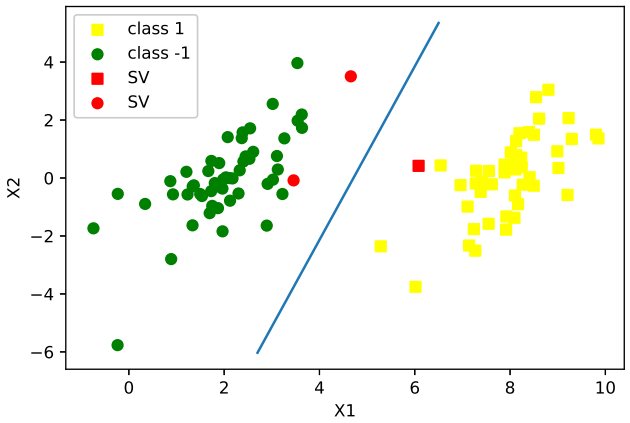



图 24: 简化版 SMO 算法运行结果图

简化版的 SMO 算法：

① 应用简化版的 SMO 算法来处理小规模数据集，不注重它的运行效率。

② $\alpha$ 对选取比较粗糙，这是效率低下的主要原因。

③ 不考虑 $\eta = (K_{11} + K_{22} - 2K_{12}) \leq 0$ 情况，事实上，这对算法的收敛没有太大影响。

④ 没有向线性不可分的情况拓展，即就是没有使用核技巧，难以应用在实际中。

### 1.7.10 实战：完整版的SMO算法加速优化

程序核心函数代码：

```
class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler): # Initialize the
        structure with the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
```

```
        self.tol = toler
        self.m = np.shape(dataMatIn)[0]
        self.alphas = np.mat(np.zeros((self.m,1)))
        self.b = 0
        self.eCache = np.mat(np.zeros((self.m,2))) #first column is valid flag

def calcEk(oS, k):
    fXk = float(np.multiply(oS.alphas,oS.labelMat).T*(oS.X*oS.X[k,:].T)) + oS.b
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJrand(i,m):
    j=i #we want to select any J not equal to i
    while (j==i):
        j = int(np.random.uniform(0,m))
    return j

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def selectJ(i, oS, Ei):      #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei] #set valid #choose the alpha that gives the maximum
        delta E
    validEcacheList = np.nonzero(oS.eCache[:,0].A)[0];print("the length of
        validEcacheList",len(validEcacheList))
    if (len(validEcacheList)) > 1:
        for k in validEcacheList: #loop through valid Ecache values and find
            the one that maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:   #in this case (first time around) we don't have any valid eCache
```

```
        values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEk(oS, k):#after any alpha has changed update the new value in the
    cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def innerL(i, oS):
    Ei = calcEk(oS, i)
    if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or
        ((oS.labelMat[i]*Ei > oS.tol) and (oS.alphas[i] > 0)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L==H: print ("L==H"); return 0
        eta = 2.0 * oS.X[i,:]*oS.X[j,:].T - oS.X[i,:]*oS.X[i,:].T -
            oS.X[j,:]*oS.X[j,:].T
        if eta >= 0: print ("eta>=0"); return 0
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print ("j not moving
            enough"); return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold -
            oS.alphas[j])#update i by the same amount as j
        updateEk(oS, i) #added this for the Ecache              #the update is
            in the oppostie direction
        b1 = oS.b - Ei-
            oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.X[i,:]*oS.X[i,:].T -
            oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.X[i,:]*oS.X[j,:].T
        b2 = oS.b - Ej-
            oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.X[i,:]*oS.X[j,:].T -
```

```
            oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.X[j,:]*oS.X[j,:].T
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else: return 0

def smoP(dataMatIn, classLabels, C, toler, maxIter): #full Platt SMO
    oS = optStruct(np.mat(dataMatIn),np.mat(classLabels).transpose(),C,toler)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet: #go over all
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
                print ("fullSet, iter: %d i:%d, pairs changed %d" %
                    (iter,i,alphaPairsChanged))
            iter += 1
        else:#go over non-bound (railed) alphas
            nonBoundIs = np.nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print ("non-bound, iter: %d i:%d, pairs changed %d" %
                    (iter,i,alphaPairsChanged))
            iter += 1
        if entireSet: entireSet = False #toggle entire set loop
        elif (alphaPairsChanged == 0): entireSet = True
        print ("iteration number: %d" % iter)
    return oS.b,oS.alphas

def test():
    dataArr,labelArr = svmMLiA.loadDataSet('testSet.txt')
    b,alpha = svmMLiA.smoSimple(dataArr,labelArr,0.6,0.001,80)
    print (b)
    print (alpha[alpha>0])
    for i in range(100):
        if alpha[i]>0.0:
            print (i,dataArr[i],labelArr[i])
```

实验结果运行时间：

```
Time For Run CompleteSMO:0.06899004697106648s
```

运行结果可视化图：

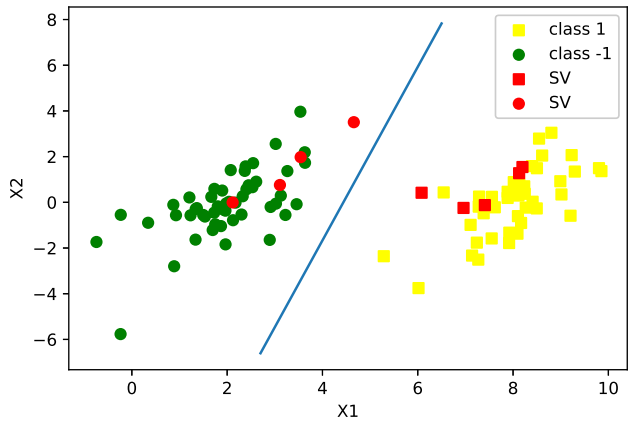



图 25: 完整版 SMO 算法运行结果图

完整版的 SMO 算法：

① 算法运行效率相较于简化版明显提高，可利用其在大数据量下进行分类。

② 对 $\alpha$ 对变量的高效选取是算法高效的保证。

③ 还是没有考虑 $\eta = (K_{11} + K_{22} - 2K_{12}) \leq 0$ 情况，因对算法的收敛没有太大影响。

④ 还是没有向线性不可分的情况拓展，即就是没有使用核技巧，难以应用在实际中。

### 1.7.11 实战：引入核函数处理线性不可分情况

程序核心函数代码：

```
class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler, kTup): # Initialize
        the structure with the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = np.shape(dataMatIn)[0]
        self.alphas = np.mat(np.zeros((self.m,1)))
        self.b = 0
```

```
        self.eCache = np.mat(np.zeros((self.m,2))) #first column is valid flag
        self.K = np.mat(np.zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def selectJrand(i,m):
    j=i #we want to select any J not equal to i
    while (j==i):
        j = int(np.random.uniform(0,m))
    return j

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def kernelTrans(X, A, kTup): #calc the kernel or transform data to a higher
    dimensional space
    m,n = np.shape(X)
    K = np.mat(np.zeros((m,1)))
    if kTup[0]=='lin': K = X * A.T #linear kernel
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = np.exp(K/(-1*kTup[1]**2)) #divide in NumPy is element-wise not
            matrix like Matlab
    else: raise NameError('Houston We Have a Problem -- \
    That Kernel is not recognized')
    return K

def calcEk(oS, k):
    fXk = float(np.multiply(oS.alphas,oS.labelMat).T*oS.K[:,k]) + oS.b
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJ(i, oS, Ei):      #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
```

```
    oS.eCache[i] = [1,Ei] #set valid #choose the alpha that gives the maximum
        delta E
    validEcacheList = np.nonzero(oS.eCache[:,0].A)[0];print("the length of
        validEcacheList",len(validEcacheList))
    if (len(validEcacheList)) > 1:
        for k in validEcacheList: #loop through valid Ecache values and find
            the one that maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:  #in this case (first time around) we don't have any valid eCache
        values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEk(oS, k):#after any alpha has changed update the new value in the
    cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]


def innerL(i, oS):
    Ei = calcEk(oS, i)
    if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or
        ((oS.labelMat[i]*Ei > oS.tol) and (oS.alphas[i] > 0)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L==H: print ("L==H"); return 0
        eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j] #changed for kernel
        if eta >= 0: print ("eta>=0"); return 0
```

```
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print ("j not moving
            enough"); return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold -
            oS.alphas[j])#update i by the same amount as j
        updateEk(oS, i) #added this for the Ecache              #the update is
            in the oppostie direction
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] -
            oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[i,j]
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]-
            oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[j,j]
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else: return 0

def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)): #full
    Platt SMO
    oS =
        optStruct(np.mat(dataMatIn),np.mat(classLabels).transpose(),C,toler,kTup)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet:  #go over all
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
                print ("fullSet, iter: %d i:%d, pairs changed %d" %
                    (iter,i,alphaPairsChanged))
            iter += 1
        else:#go over non-bound (railed) alphas
            nonBoundIs = np.nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print ("non-bound, iter: %d i:%d, pairs changed %d" %
                    (iter,i,alphaPairsChanged))
            iter += 1
```

```
        if entireSet: entireSet = False #toggle entire set loop
        elif (alphaPairsChanged == 0): entireSet = True
        print ("iteration number: %d" % iter)
    return oS.b,oS.alphas

def testRbf(k1=1.5):
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', k1))
    #C=200 important
    datMat=np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    svInd=np.nonzero(alphas.A>0)[0]
    sVs=datMat[svInd];print (sVs) #get matrix of only support vectors
    labelSV = labelMat[svInd];
    print ("there are %d Support Vectors" % np.shape(sVs)[0])
    m,n = np.shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if np.sign(predict)!=np.sign(labelArr[i]): errorCount += 1
    print ("the training error rate is: %f" % (float(errorCount)/m))
    dataArr,labelArr = loadDataSet('testSetRBF2.txt')
    errorCount = 0
    datMat=np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    m,n = np.shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if np.sign(predict)!=np.sign(labelArr[i]): errorCount += 1
    print ("the test error rate is: %f" % (float(errorCount)/m))
```

运行结果:

```
there are 21 Support Vectors
the training error rate is: 0.020000
the test error rate is: 0.030000
Time For Run CompleteSMOWithKernel:0.5218268112548685
```

引入核函数完整版的 SMO 算法:

① 在完整 SMO 算法的基础之上，引入了核函数（径向基函数）。

② 向非线性情况进行了扩展，可以利用其进行项目实战。

### 1.7.12 实战：利用SVM进行识别字体

数据形式:

```
#two training example.
00000000000000001111000000000000
00000000000000011111111000000000
00000000000000011111111100000000
00000000000000011111111110000000
00000000000000011111111110000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000001111111111100000000
00000000000001111111111100000000
00000000000001111111111100000000
00000000000001111111111000000000
00000000000011111111111000000000
00000000000111111111111000000000
00000001111111111111111000000000
00000011111111111111110000000000
00000111111111111111110000000000
00000111111111111111110000000000
00000011111111111111110000000000
00000011111110111111110000000000
00000001111100011111110000000000
00000000000000011111110000000000
00000000000000011111100000000000
00000000000000111111110000000000
00000000000000011111110000000000
00000000000000011111110000000000
00000000000000011111111000000000
00000000000000011111111000000000
00000000000000011111111000000000
00000000000000000111111110000000
00000000000000000111111100000000
```

```
00000000001111100000000000000000
00000000011111100000000000000000
00000000111111111000000000000000
00000000111111111110000000000000
00000001111111111110000000000000
00000001111111111110000000000000
00000001111110111111000000000000
00000001111110011111110000000000
00000001111100011111111000000000
00000001111100011111111000000000
00000001111100011111111110000000
00000001111100011111111110000000
00000001111100111111111000000000
00000001111111111111111000000000
00000000011111111111111100000000
00000000001111111111111100000000
00000000000111111111111100000000
00000000000111111111111100000000
00000000000000000011111000000000
00000000000000000011111100000000
00000000000000000011111110000000
00000000000000000011111110000000
00000000000000000011111100000000
00000000000000000011111100000000
00000000000000000011111100000000
00000000000000000011111110000000
00000001111100011111111110000000
00000001111111111111111100000000
00000001111111111111111000000000
00000001111111111111111000000000
00000000111111111111111000000000
00000000001111111111100000000000
```

数据处理函数:

```
def loadImages(dirName):
    from os import listdir
    hwLabels = []
    trainingFileList = listdir(dirName)      #load the training set
    m = len(trainingFileList);
```

```
    trainingMat = np.zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]   #1_xx.txt or 9_xx.txt
        fileStr = fileNameStr.split('.')[0] #take off .txt   1_xx or 9_xx
        classNumStr = int(fileStr.split('_')[0]) #1 or 9
        if classNumStr == 9: hwLabels.append(-1)
        else: hwLabels.append(1)
        trainingMat[i,:] = img2vector('%s/%s' % (dirName, fileNameStr))
    return trainingMat, hwLabels

def img2vector(filename): #32x32 per feature
    returnVect = np.zeros((1,1024)) # numpy array
    fr = open(filename) #_io.TextIOWrapper
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect

def testDigits(kTup=('rbf', 10)):
    dataArr,labelArr = loadImages('trainingDigits')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, kTup)
    datMat=np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    svInd=np.nonzero(alphas.A>0)[0]
    sVs=datMat[svInd] #support vector
    labelSV = labelMat[svInd];
    print ("there are %d Support Vectors" % np.shape(sVs)[0])
    m,n = np.shape(datMat)
    errorCount = 0
    for i in range(m):#support vector only
        kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if np.sign(predict)!=np.sign(labelArr[i]): errorCount += 1
    print ("the training error rate is: %f" % (float(errorCount)/m))
    dataArr,labelArr = loadImages('testDigits')
    errorCount = 0
    datMat=np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    m,n = np.shape(datMat)
    for i in range(m):#support vector only
        kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
```

```
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if np.sign(predict)!=np.sign(labelArr[i]): errorCount += 1
    print ("the test error rate is: %f" % (float(errorCount)/m) )
```

模型准确率及结果:

```
there are 112 Support Vectors
the training error rate is: 0.000000
the test error rate is: 0.010753
Time For Run CompleteSMOWithKernel:9.26518176738s
```

利用引入核函数的完整 SMO 算法进行一个手写数字识别的实战二分类，得到了不错的分类效果。

### 1.7.13 小结

支持向量机是一种分类器。之所以称为“机”是因为它会产生一个二值决策结果，即它是一种决策“机”。支持向量机的泛化错误率较低，也就是说它具有良好的学习能力，且学到的结果具有很好的推广性。这些优点使得支持向量机十分流行，有些人认为它是监督学习中最好的定式算法。

支持向量机试图通过求解一个二次优化问题来最大化分类间隔。在过去，训练支持向量机常采用非常复杂并且低效的二次规划求解方法。John.C.Platt引入了 SMO 算法，此算法可以通过每次只优化2个 $\alpha$ 值来加快SVM的训练速度。

核方法或者说核技巧会将数据（有时是非线性数据）从一个低维空间映射到一个高维空间，可以将一个在低维空间中的非线性问题转换成高维空间下的线性问题来求解。核方法不止在 SVM 中适用，还可以用于其他算法中。而其中的径向基函数是一个常用的度量两个向量距离的核函数。

支持向量机是一个二类分类器。当用其解决多类问题时，则需要额外的方法对其进行扩展。 SVM 的效果也对优化参数和所用核函数中的参数敏感。

## 1.8 AdaBoost算法

当做重要决定时，大家可能都会考虑吸取多个专家而不只是一个人的意见。那么机器学习处理的问题也是这样。这就是元算法（meta-algorithm）背后的思路。元算法是对其他算法进行组合的一种方式。接下来我们将集中关注一个称作AdaBoost的最流行的元算法。由于某些人认为AdaBoost是最好的监督学习的方法，所以该方法是机器学习工具箱中最强有力的工具之一。

首先讨论不同分类器的集成方法（bagging），然后主要关注boosting方法及其代表分类器Adaboost。 AdaBoost算法将应用在上述单层决策树分类器之上。将在一个难数据集上应用AdaBoost分类器，以了解该算法是如何迅速超越其他分类器的。

### 1.8.1 集成方法

我们自然可以将不同的分类器组合，而这种组合结果则被称为集成方法（ensemble method）或者元算法（meta-algorithm）。使用集成方法时会有多种形式：可以是不同算法的集成，也可以是同一算法在不同设置下的集成，还可以是数据集不同部分分配给不同分类器之后的集成。

**bagging：基于数据重抽样的分类器方法**

自举汇聚法（bootstrap aggregating），也称为bagging方法，是在从原始数据集选择S次后得到S个新数据集的一种技术。新数据集和原数据集的大小相等。每个数据集都是通过在原始数据集中随机选择一个样本来进行替换而得到的①。这里的替换就意味着可以多次地选择同一样本。这一性质就允许新数据集中可以有重复的值，而原始数据集的某些值在新集合中则不再出现。

在S个数据集建好之后，将某个学习算法分别作用于每个数据集就得到了S个分类器。当我们要对新数据进行分类时，就可以应用这S个分类器进行分类。与此同时，选择分类器投票结果中最多的类别作为最后的分类结果。

当然，还有一些更先进的bagging方法，比如随机森林（random forest）。

**boosting：基于错误提升分类器性能**

boosting是一种与bagging很类似的技术。不论是在boosting还是bagging当中，所使用的多个分类器的类型都是一致的。但是在前者当中，不同的分类器是通过串行训练而获得的，每个新分类器都根据已训练出的分类器的性能来进行训练。boosting是通过集中关注被已有分类器错分的那些数据来获得新的分类器。

因boosting分类的结果是基于所有分类器的加权求和结果的，boosting与bagging不太一样。bagging中的分类器权重是相等的，而boosting中的分类器权重并不相等，每个权重代表的是其对应分类器在上一轮迭代中的成功度。

### 1.8.2 AdaBoost

AdaBoost是boosting方法中一个最流行的版本。AdaBoost是adaptive boosting（自适应boosting）的缩写。

能否使用弱分类器和多个实例来构建一个强分类器？这是一个非常有趣的理论问题。这里的“弱”意味着分类器的性能比随机猜测要略好，但是也不会好太多。这就是说，在二分类情况下弱分类器的错误率会高于50%，而“强”分类器的错误率将会低很多。AdaBoost算法即脱胎于上述理论问题。

**运行过程**

训练数据中的每个样本，并赋予其一个权重，这些权重构成了向量D。一开始，这些权重都初始化成相等值。首先在训练数据上训练出一个弱分类器并计算该分类器的错误率，然后在同一数据集上再次训练弱分类器。在分类器的第二次训练当中，将会重新调整每个样本的权重，其中第一次分对的样本的权重将会降低，而第一次分错的样本的权重将会提高。为了从所有弱分类器中得到最终的分类结果，AdaBoost为每个分类器都分配了一个权重值 $\alpha$，这些 $\alpha$ 值是基于每个弱分类器的错误率进行计算的。其中，错误率 $\epsilon$ 的定义为：

$$\epsilon = \frac{\text{未正确分类的样本数目}}{\text{所有样本数目}}$$

而 $\alpha$ 的计算公式如下：

$$\alpha = \frac{1}{2}\ln(\frac{1-\epsilon}{\epsilon})$$

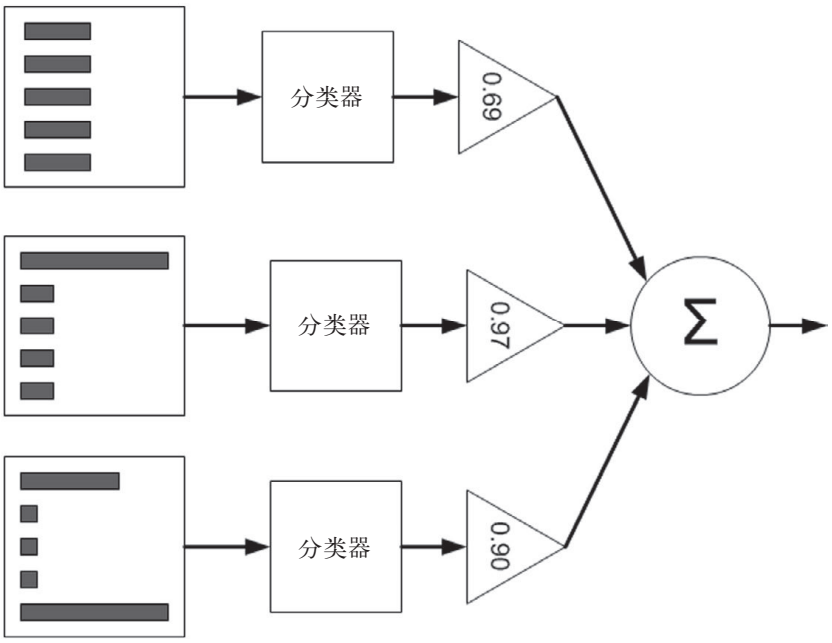



图 26: 算法流程

计算出 $\alpha$ 值之后，可以对权重向量D进行更新，以使得那些正确分类的样本的权重降低而错分样本的权重升高。D的计算方法如下。

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{\sum D} \quad \text{该样本被正确分类}$$

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{\sum D} \quad \text{该样本被错误分类}$$

在计算出D之后，AdaBoost又开始进入下一轮迭代。AdaBoost算法会不断地重复训练和调整权重的过程，直到训练错误率为0或者弱分类器的数目达到用户的指定值为止。

**AdaBoost训练误差界**

AdaBoost算法最终分类器的训练误差界为

$$\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i) \leq \frac{1}{N}\sum_{i} \exp(-y_i f(x_i)) = \prod_{m} Z_m$$

**AdaBoost 算法还有另一个解释，即可以认为 AdaBoost 算法是模型为加法模型、损失函数为指数函数、学习算法为前向分步算法时的二分类学习方法。**

AdaBoost 算法是前向分步加法算法的特例．这时，模型是由基本分类器组成的加法模型，损失函数是指数函数。

### 1.8.3 实战：实现Adaboost基于决策树桩

**AdaBoost的一般流程**

**(1) 收集数据**：可以使用任意方法。

**(2) 准备数据**：依赖于所使用的弱分类器类型，本章使用的是单层决策树，这种分类器可以处理任何数据类型。当然也可以使用任意分类器作为弱分类器。作为弱分类器，简单分类器的效果更好。

**(3) 分析数据**：可以使用任意方法。

**(4) 训练算法**：AdaBoost的大部分时间都用在训练上，分类器将多次在同一数据集上训练弱分类器。

**(5) 测试算法**：计算分类的错误率。

**(6) 使用算法**：同SVM一样，AdaBoost预测两个类别中的一个。如果想把它应用到多个类别的场合，那么就要像多类SVM中的做法一样对AdaBoost进行修改。

核心函数:

```
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):#just classify the
    data
    retArray = np.ones((np.shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray

def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = np.inf #init error sum, to +infinity
    for i in range(n):#loop over all dimensions
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps
        for j in range(-1,int(numSteps)+1):#loop over all range in current
            dimension
            for inequal in ['lt', 'gt']: #go over less than and greater than
                threshVal = (rangeMin + float(j) * stepSize)
                predictedVals =
                    stumpClassify(dataMatrix,i,threshVal,inequal)#call stump
                    classify with i, j, lessThan
                errArr = np.mat(np.ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                weightedError = D.T*errArr #calc total error multiplied by D
#               print ('split: dim %d, thresh %.2f, thresh ineqal: %s, the
    weighted error is %.3f' %(i, threshVal, inequal, weightedError))
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst


def adaBoostTrainDS(dataArr,classLabels,numIt=40):
```

```
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m,1))/m) #init D to all equal
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)#build Stump
#        print ("D:",D.T)
        alpha = float(0.5*np.log((1.0-error)/max(error,1e-16)))#calc alpha,
            throw in max(error,eps) to account for error=0
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump)              #store Stump Params in Array
#        print ("classEst: ",classEst.T)
        expon = np.multiply(-1*alpha*np.mat(classLabels).T,classEst) #exponent
            for D calc, getting messy
        D = np.multiply(D,np.exp(expon))                            #Calc New D for
            next iteration
        D = D/D.sum()
        #calc training error of all classifiers, if this is 0 quit for loop
            early (use break)
        aggClassEst += alpha*classEst
#        print ("aggClassEst: ",aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) !=
            np.mat(classLabels).T,np.ones((m,1)))
        errorRate = aggErrors.sum()/m
        print ("total error: ",errorRate)
        if errorRate == 0.0: break
    return weakClassArr,aggClassEst
```

以决策树桩为基本的弱分类器，实现Adaboost算法，具有很理想的训练误差，当弱分类器的数目不断增加训练错误率会下降，但是模型的泛化能力则会降低。

### 1.8.4 实战：从疝气病症预测病马的死亡率

运行结果:

```
total error: 0.284280936455
total error: 0.284280936455
total error: 0.247491638796
total error: 0.247491638796
total error: 0.254180602007
total error: 0.240802675585
```

```
total error: 0.240802675585
total error: 0.220735785953
total error: 0.247491638796
total error: 0.230769230769
error rate: 0.238805970149
the Area Under the Curve is: 0.8582969635063604
```

从结果来看，相比逻辑回归的错误率34%，AdaBoostd的错误率已经有了较大的提升。

### 1.8.5 非均衡分类问题

在之前所有的分类问题，我们的假设所有类别的分类代价是一样的。这样显然与现实情况的结果代价是不符合的，例如垃圾邮件宁愿不分也不愿错分。事实上在大多数情况下不同类别的分类代价并不相等。对于非均衡的分类问题，常用的措施是其他分类指标衡量；代价函数的分类器决策控制；处理非均衡问题的数据抽样方法。

**其他分类性能度量指标：正确率、召回率及 ROC 曲线**

到现在为止，本书都是基于错误率来衡量分类器任务的成功程度的。错误率指的是在所有测试样例中错分的样例比例。实际上，这样的度量错误掩盖了样例如何被分错的事实。在机器学习中，有一个普遍适用的称为混淆矩阵（confusion matrix）的工具，它可以帮助人们更好地了解分类中的错误。

针对一个简单的二类问题。在这个二类问题中，如果将一个正例判为正例，那么就可以认为产生了一个真正例（True Positive，TP，也称真阳）；如果对一个反例正确地判为反例，则认为产生了一个真反例（True Negative，TN，也称真阴）。相应地，另外两种情况则分别称为伪反例（False Negative， FN，也称假阴）和伪正例（False Positive，FP，也称假阳）。

| | | 预测结果 | |
|---|---|---|---|
| | | +1 | −1 |
| 真实结果 | +1 | 真正例（TP） | 伪反例（FN） |
| | −1 | 伪正例（FP） | 真反例（TN） |

在分类中，当某个类别的重要性高于其他类别时，就可以利用上述定义来定义出多个比错误率更好的新指标。第一个指标是正确率（Precision），它等于TP/(TP+FP)，给出的是预测为正例的样本中的真正正例的比例。第二个指标是召回率（Recall），它等于TP/(TP+FN)，给出的是预测为正例的真实正例占所有真实正例的比例。在召回率很大的分类器中，真正判错的正例的数目并不多。

可以很容易构造一个高正确率或高召回率的分类器，但是很难同时保证两者成立。如果将任何样本都判为正例，那么召回率达到百分之百而此时正确率很低。构建一个同时使正确率和召回率最大的分类器是有难度的。

另一个用于度量分类中的非均衡性的工具是ROC曲线（ROC curve），ROC代表接收者操作特征（receiver operating characteristic），它最早在二战期间由电气工程师构建雷达系统时使用过。

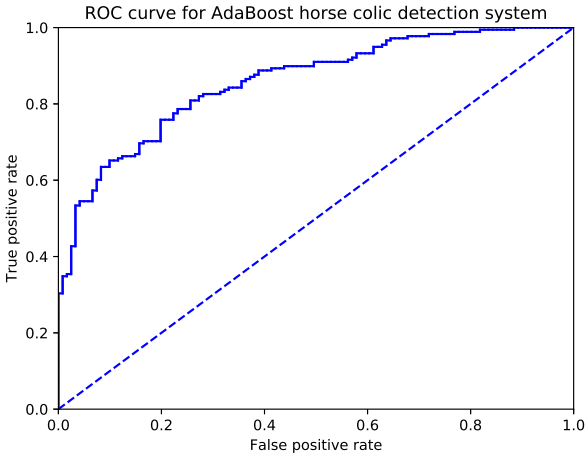



图 27: 实战ROC曲线

给出了两条线，一条虚线一条实线。图中的横轴假阳率=FP/(FP+TN)，而纵轴真阳率=TP/(TP+FN)。ROC曲线给出的是当阈值变化时假阳率和真阳率的变化情况。左下角的点所对应的是将所有样例判为反例的情况，而右上角的点对应的则是将所有样例判为正例的情况。虚线给出的是随机猜测的结果曲线。

ROC曲线不但可以用于比较分类器，还可以基于成本效益（cost-versus-benefit）分析来做出决策。由于在不同的阈值下，不同的分类器的表现情况可能各不相同，因此以某种方式将它们组合起来或许会更有意义。如果只是简单地观察分类器的错误率，那么我们就难以得到这种更深入的洞察效果了。在理想的情况下，最佳的分类器应该尽可能地处于左上角，这就意味着分类器在假阳率很低的同时获得了很高的真阳率。例如在垃圾邮件的过滤中，这就相当于过滤了所有的垃圾邮件，但没有将任何合法邮件误识为垃圾邮件而放入垃圾邮件的文件夹中。

对不同的ROC曲线进行比较的一个指标是曲线下的面积（Area Unser the Curve，AUC）。AUC给出的是分类器的平均性能值，当然它并不能完全代替对整条曲线的观察。一个完美分类器的AUC为1.0，而随机猜测的AUC则为0.5。

为了画出ROC曲线，分类器必须提供每个样例被判为阳性或者阴性的可信程度值。尽管大多数分类器都能做到这一点，但是通常情况下，这些值会在最后输出离散分类标

签之前被清除。朴素贝叶斯能够提供一个可能性，而在Logistic回归中输入到Sigmoid函数中的是一个数值。在AdaBoost和SVM中，都会计算出一个数值然后输入到sign()函数中。所有的这些值都可以用于衡量给定分类器的预测强度。为了创建ROC曲线，首先要将分类样例按照其预测强度排序。先从排名最低的样例开始，所有排名更低的样例都被判为反例，而所有排名更高的样例都被判为正例。该情况的对应点为¡1.0,1.0¿。然后，将其移到排名次低的样例中去，如果该样例属于正例，那么对真阳率进行修改；如果该样例属于反例，那么对假阴率进行修改。

**基于代价函数的分类器决策控制**

除了调节分类器的阈值之外，我们还有一些其他可以用于处理非均衡分类代价的方法，其中的一种称为代价敏感的学习（cost-sensitive learning）。考虑表7-4中的代价矩阵，第一张表给出的是到目前为止分类器的代价矩阵（代价不是0就是1）。我们可以基于该代价矩阵计算其总代价：TP*0+FN*1+FP*1+TN*0。接下来我们考虑下面的第二张表，基于该代价矩阵的分类代价的计算公式为：TP*(-5)+FN*1+FP*50+TN*0。采用第二张表作为代价矩阵时，两种分类错误的代价是不一样的。类似地，这两种正确分类所得到的收益也不一样。如果在构建分类器时，知道了这些代价值，那么就可以选择付出最小代价的分类器。

在分类算法中，我们有很多方法可以用来引入代价信息。在AdaBoost中，可以基于代价函数来调整错误权重向量D。在朴素贝叶斯中，可以选择具有最小期望代价而不是最大概率的类别作为最后的结果。在SVM中，可以在代价函数中对于不同的类别选择不同的参数C。上述做法就会给较小类更多的权重，即在训练时，小类当中只允许更少的错误。

| | | 预测结果 | |
|---|---|---|---|
| | | +1 | −1 |
| 真实结果 | +1 | 0 | 1 |
| | −1 | 1 | 0 |

| | | 预测结果 | |
|---|---|---|---|
| | | +1 | −1 |
| 真实结果 | +1 | −5 | 1 |
| | −1 | 50 | 0 |

图 28: 代价矩阵

**处理非均衡问题的数据抽样方法**

另外一种针对非均衡问题调节分类器的方法，就是对分类器的训练数据进行改造。这可以通过欠抽样（undersampling）或者过抽样（oversampling）来实现。过抽样意味着复制样例，而欠抽样意味着删除样例。不管采用哪种方式，数据都会从原始形式改造

为新形式。抽样过程则可以通过随机方式或者某个预定方式来实现。

通常也会存在某个罕见的类别需要我们来识别，比如在信用卡欺诈当中。如前所述，正例类别属于罕见类别。我们希望对于这种罕见类别能尽可能保留更多的信息，因此，我们应该保留正例类别中的所有样例，而对反例类别进行欠抽样或者样例删除处理。这种方法的一个缺点就在于要确定哪些样例需要进行剔除。但是，在选择剔除的样例中可能携带了剩余样例中并不包含的有价值信息。

上述问题的一种解决办法，就是选择那些离决策边界较远的样例进行删除。假定我们有一个数据集，其中有50例信用卡欺诈交易和5000例合法交易。如果我们想要对合法交易样例进行欠抽样处理，使得这两类数据比较均衡的话，那么我们就需要去掉4950个样例，而这些样例中可能包含很多有价值的信息。这看上去有些极端，因此有一种替代的策略就是使用反例类别的欠抽样和正例类别的过抽样相混合的方法。

要对正例类别进行过抽样，我们可以复制已有样例或者加入与已有样例相似的点。一种方法是加入已有数据点的插值点，但是这种做法可能会导致过拟合的问题。

# 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 $\eta$ 与输入 X 是线性相关的，即： $\eta = \theta x$ ,

若 $\eta$ 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 $\epsilon$ 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:

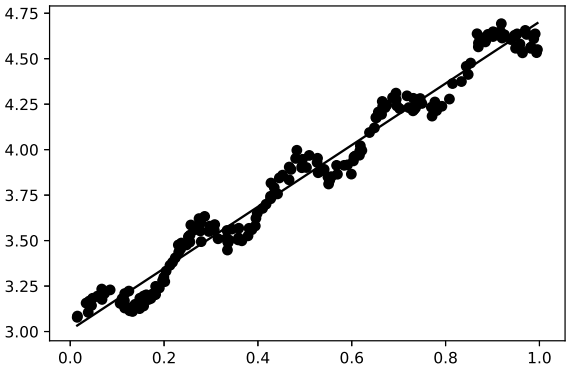



图 29: 线性回归得到拟合直线

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:

$$
\begin{aligned}
&LR:\\
&\quad (1).Fit\ \theta\ to\ minimize\ \sum_{i=1}^{m}(y^{(i)}-\theta^T x^{(i)})^2\\
&\quad (2).Output:\ \theta^T x\\
&LWLR:\\
&\quad (1).Fit\ \theta\ to\ minimize\ \sum_{i=1}^{m}w_{(i)}(y^{(i)}-\theta^T x^{(i)})^2\\
&\quad (2).Output:\ \theta^T x\\
&\quad weighted:\\
&\quad w_{(i)}=exp(-\frac{(x_{(i)}-x)^2}{2\tau^2})
\end{aligned}
$$

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 $\tau$ 为衰减因子，即权重衰减的速率。

$\tau$ 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 $\tau$ .

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 $\tau$ 的作用范围：

实验结果可视化：

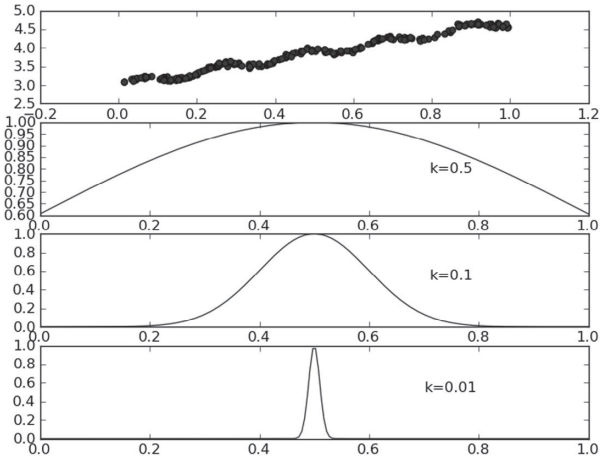



图 30: $\tau$ 的作用范围

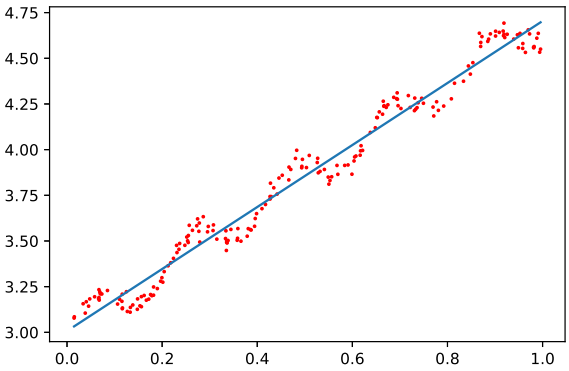


图 31: $\tau = 1$

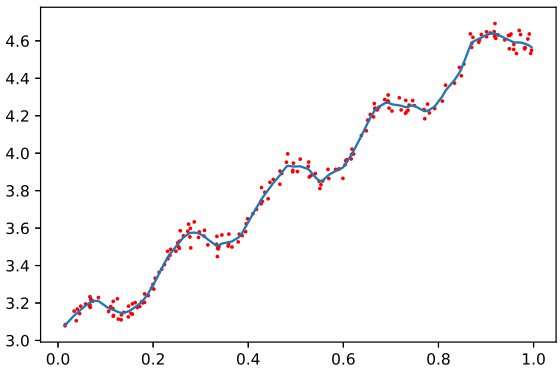


图 32: $\tau = 0.01$

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。

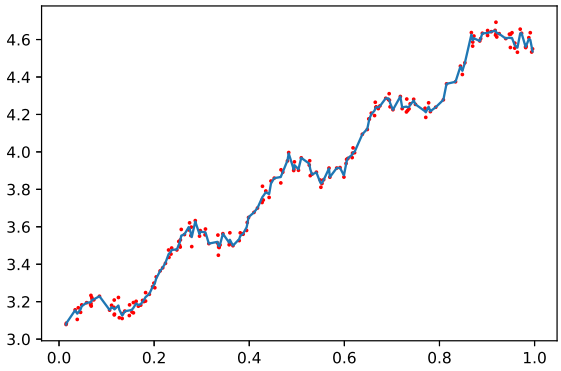



图 33: $\tau = 0.003$

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
```