计算机科学与技术学院2018机器学习—实验报告

- 课程名称： 机器学习
- 课程类型： 选修
- 实验题目： 逻辑回归
- 学号：1160100626
- 姓名：单心茹

实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

实验要求求及实验环境

实验要求

实现两种损失函数的参数估计：

1. 无惩罚项；
2. 加入对参数的惩罚

可以采用**梯度下降、共轭梯度或者牛顿法**等。

实验环境

Win10 + 8GB RAM + python 3.6

设计思想与实现

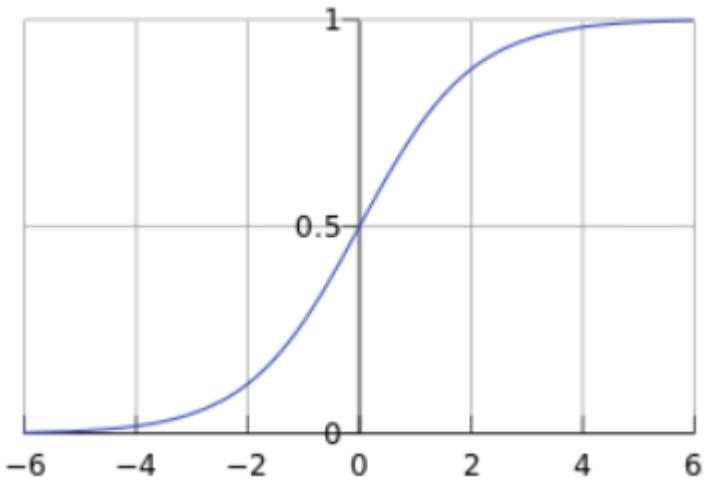
对于二分类问题，logistic regression模型是一个很好的判别模型

逻辑回归模型：

在分类时，需要一个函数将输出的值映射到[0,1]

Hypothesis：

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



对于Hypothesis输出的直观解释：

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

对于输入问题可以刻画为矩阵形式：

InputX：k个样本，m维特征

$$\begin{bmatrix} 1 & x_1^1 & x_2^1 \ldots\ldots x_m^1 \\ 1 & x_1^2 & x_2^2 \ldots\ldots x_m^2 \\ 1 & x_1^3 & x_2^3 \ldots\ldots x_m^3 \\ \ldots\ldots & & \\ 1 & x_1^k & x_2^k \ldots\ldots x_m^k \end{bmatrix}$$

InputY：标签，正例与反例

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \ldots\ldots \\ 1 \end{bmatrix}$$

Output：参数theta的学习值

则可以构造决策面分开两个类别，则**线性决策面**为：

$$\theta^T x = 0$$

也可以构造非线性的决策面将类别分开，**本实验中采用线性决策面**

Cost function：

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

要最小化损失函数，则可以采用梯度下降，共轭梯度，牛顿法进行优化，在后面部分详细叙述。

梯度下降

梯度下降法的目的是找到一组θ，让损失函数达到极小，算法主要思想为：**调整合适的步长，沿着梯度下降的方向搜索，直到满足收敛条件**

1. 初始化θ, 设置学习速率即步长α
2. 由公式

$$\theta j := \theta j + \frac{1}{m}\sum_{i=1}^{N}(h_\theta(x^{(i)}) - y^{(i)})^2 x_j^i$$

循环更新θ，直到满足收敛条件，**本实验中设置的收敛条件为前一次损失函数的值与后一次的差的绝对值小于一个人工设定的极小值。**

- **非正则化梯度下降实现的核心部分：**

```
def normal_gradientDescent(self):
    count = 0
    error = 0
    while count < self.loop_max:
        count += 1
        # 更新theta
        for j in range(0, self.dimension + 1):
            diff = 0
            for i in range(0, len(self.train_y)):
                diff += (self.sigmod(self.theta * self.train_x[i].T) -
self.train_y[i]) * self.train_x[i, j]
                self.theta[0, j] = self.theta[0, j] - self.alpha * diff

        # 计算损失函数
        loss_function = self.normal_lossFunction(self.theta, self.train_x,
self.train_y)
        if abs(loss_function - error) < self.epsilon:
            break
        else:
            error = loss_function
    return self.theta
```

- **正则化的梯度下降：**

Cost function：

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

其中需要注意的是θ_0不参与正则化，所以在循环更新θ的算法变为：

$$
\begin{aligned}
&Repeat\ \ until\ \ convergence\{\\
&\theta_0:=\theta_0-\alpha\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)})-y^{(i)})\cdot x_0^{(i)})\\
&\theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)})-y^{(i)})\cdot x_j^{(i)}+\frac{\lambda}{m}\theta_j)\\
&\qquad for\ \ j=1,2,...n\\
&\}
\end{aligned}
$$

代码实现与非正则化大致相同，不重复贴上。

牛顿法

基本思想：在现有极小点估计值的附近对$f(x)$做二阶泰勒展开，进而找到极小点的下一个估计值，则

$$\varphi(x)=f(x_k)+f^{'}(x_k)(x-x_k)+\frac{1}{2}f^{''}(x_k)(x-x_k)^2$$

令$\varphi(x)=0$可得：

$$x_{k+1}=x_k-\frac{f^{'}(x_k)}{f^{''}(x_k)}$$

由于是求极小点，极值条件要求它为$\varphi(x)$的驻点，故两边作用一个梯度算子：

$$g_k+H_k*(x-x_k)=0$$

若Hessian矩阵非奇异，则可得：

$$x_{k+1}=x_k-H_k^{-1}g_k$$

其中搜索方向$d_k=-H_k^{-1}g_k$称为**牛顿方向。**

其中梯度g：

$$g=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)}$$

Hessian矩阵：

$$H=\frac{1}{m}[\sum_{i=1}^{m}(h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T]$$

Algorithm：

1. 给定初值θ为零向量和精度阈值ϵ，并令$k=0$

2. 计算g_k和H_k
3. 若g_k的范数 < ϵ，则停止迭代；否则确定搜索方向$d_k = -H_k^{-1}g_k$
4. 更新参数$\theta = \theta - H_k^{-1}g_k$

重复步骤2~4，直到精度达到要求或者达到一定的迭代次数。

当目标函数是二次函数时，由于泰勒展开函数与原目标函数不是近似而是完全相同的二次式，海森矩阵退化成一个常数矩阵，从任一点出发，只需一步迭代就能达到$f(x)$的极小点，因此**牛顿法是一种具有二次收敛性的算法**，对于非二次收敛性的函数，若函数的二次性态较强，或迭代点已进入极小点的邻域，则其收敛速度也是很快的，这是牛顿法的主要优点。

牛顿法实现核心部分如下：

```
def normal_newtonMethod(self):
    """ train a logistic regression model without regression
        :return: learned theta
    """
    k = 0
    while k < self.loop_max:
        g = self.normal_gradient(self.train_x, self.train_y, self.theta)
        H = self.hessianMatrix(self.train_x, self.train_y, self.theta)

        if np.linalg.norm(g) < self.epsilon:
            break
        else:
            self.theta = self.theta - (H.I * g.T).T
            k = k + 1
    print(self.theta)
    return self.theta
```

梯度计算部分实现：

```
    def normal_gradient(self, X, Y, theta):
        """ calculate gradient
        :param X: matrix x
        :param Y: matrix y
        :param theta: matrix theta
        :return: gradient
        """
        return (self.sigmod(theta*X.T).T - Y).T * X/len(Y)
```

Hessian矩阵计算部分实现：

```
    def hessianMatrix(self, X, Y, theta):
        """calculate Hessian Matrix
        :param X: matrix x
        :param Y: matrix y
        :param theta: matrix theta
        :return: Hessian Matrix
        """
        h = self.sigmod(theta*X.T).T
        m = np.multiply(h, (1-h))
        M = X/len(Y)
        for i in range(0, self.dimension + 1):
            M[:, i] = np.multiply(M[:, i], m)
        return X.T * M
```

加入正则项的情况：

- $g = g + \frac{\lambda}{m}\theta$
- 计算 Hessian 矩阵 H，再让 H 加上下面这一项：

$$\frac{\lambda}{m}\begin{bmatrix} 0 & 0 & 0.....0 \\ 0 & 1 & 0.....0 \\ 0 & 0 & 1.....0 \\ ..... & & \\ 0 & 0 & 0.....1 \end{bmatrix}$$

考察朴素贝叶斯假设

朴素贝叶斯中的朴素一词的来源就是**假设各特征之间相互独立**。这一假设使得朴素贝叶斯算法变得简单，但有时会牺牲一定的分类准确率。

在逻辑回归模型中，我们也**假设各特征之间相互独立**，故可以**生成相互不独立的数据**，测试模型不满足这个强假设条件时的表现。

1. **利用高斯分布生成相互不独立的样本数据：**

二维高斯分布：

- μ =(μ1 , μ2)：各位变量的均值；
- Σ：协方差矩阵，描述各维变量之间的相关度。对于二维高斯分布，有：

$$\begin{bmatrix} \sigma_{11} & \sigma 12 \\ \sigma_{21} & \sigma 22 \end{bmatrix}$$

均值表征的是各维变量的中心，它使得整个二维高斯曲面在xoy平面上移动。

对于协方差矩阵，对角线上的两个元素，即σ_{11}和σ_{22}表征的是x维和y维变量的方差，决定了整个高斯曲面在某一维度上的“跨度”，方差越大，“跨度”越大。

协方差矩阵的斜对角线上面的两个元素表征的是各维变量的相关性，越大则相关程度越大。相关则不独立。

故可以由高斯分布分别生成两类二维数据，特征是互相依赖的，则可以验证不满足当贝叶斯假设时，模型的表现情况。

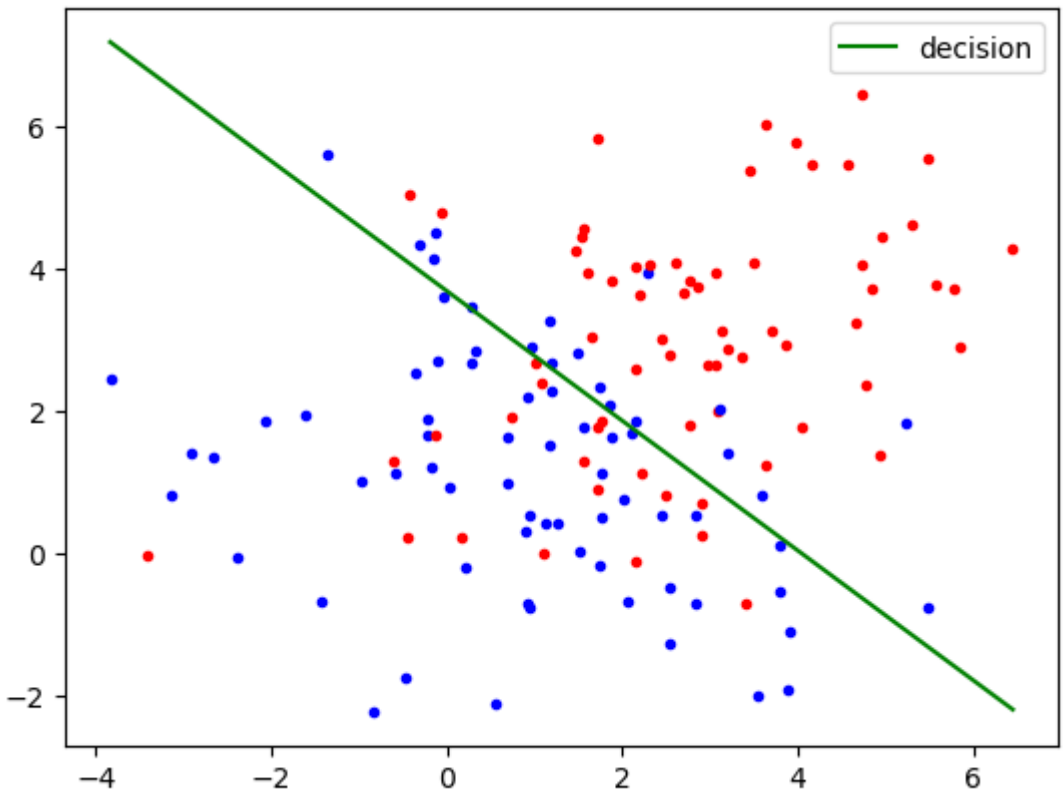
在`newton_logisticRegression.py`中具体实现：

```python
def testIndependence():
    mu1 = np.array([[1, 1]])
    sigma1 = np.array([[3, 0.5], [0, 3]])
    R1 = cholesky(sigma1)
    s1 = np.dot(np.random.randn(100, 2), R1) + mu1
    # 注意绘制的是散点图，而不是直方图
    x1 = s1[:, 0]
    x2 = s1[:, 1]
    mu2 = np.array([[5, 5]])
    sigma2 = np.array([[3, 0.5], [0.8, 3]])
    R2 = cholesky(sigma2)
    s2 = np.dot(np.random.randn(100, 2), R2) + mu2
    x3 = s2[:, 0]
    x4 = s2[:, 1]
    with open('testIndependence.txt', 'w') as f:
        for i in range(0, len(x1)):
            f.write(str(x1[i]) + ',' + str(x2[i]) + ',' + '1\n')
            f.write(str(x3[i]) + ',' + str(x4[i]) + ',' + '0\n')
        f.close()

```

2. **测试模型表现：**

下图为某一次测试的模型表现情况：



accuracy = 0.75 precision = 0.741 recall = 0.793

可以看到特征依赖性较强，分类效果不好，不满足朴素贝叶斯假设，模型表现不佳。

模型评估

假设一个二分类问题，样本有正负两个类别。那么模型预测的结果和真实标签的组合就有4种：**TP，FP，FN，TN。**

- TP：实际为正样本模型预测为正样本
- FP：实际为负样本模型预测为正样本
- FN：实际为正样本模型预测为负样本
- TN：实际为负样本模型预测为负样本

-	**Relevant**	**NonRelevant**
Retrieved	true positives （TP）	false positives（FP）
Not Retrieved	false negatives（FN）	true negatives （TN）

- Precision：

$$P = \frac{TP}{TP + FP}$$

- Recall:

$$R = \frac{TP}{TP + FN}$$

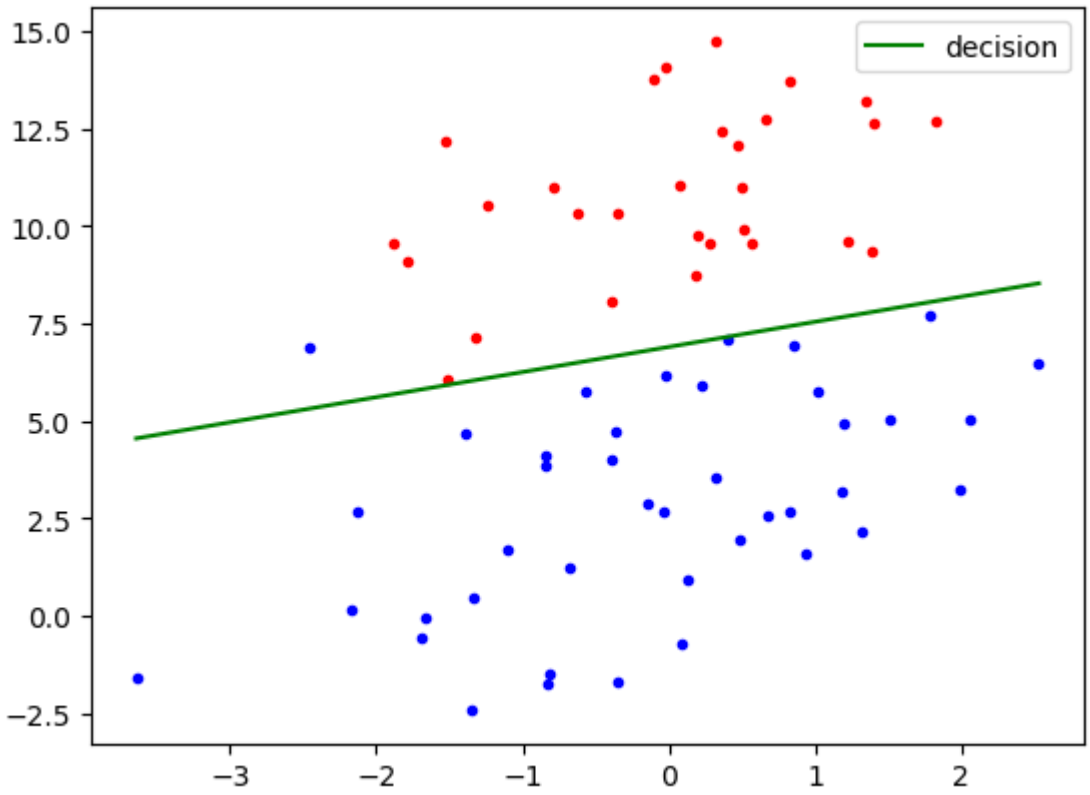
实验结果与分析

二维特征分类

数据：见压缩包内`data.txt`，二维特征，0,1分类，100个样本数据

梯度下降法优化结果

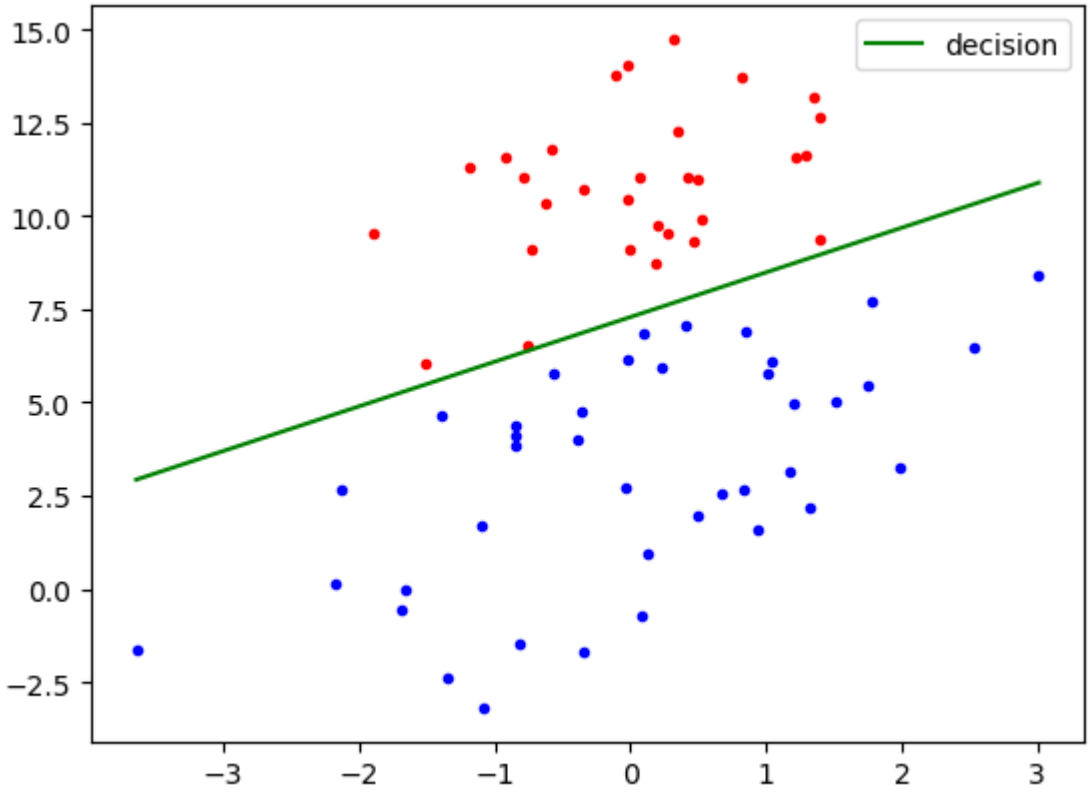
由于每次运行划分数据集为7:3的训练集与测试集，随机抽取样本，结果具有随机性，下图为某次训练的分类结果：



accuracy = 0.933 precision = 1.0 recall = 0.917 loop_count = 723

牛顿法优化结果

由于每次运行划分数据集为7:3的训练集与测试集，随机抽取样本，结果具有随机性，下图为某次训练的分类结果：



accuracy =0.867 precision = 0.917 recall = 0.846 loop_count = 4

UCI数据测试

利用UCI数据集中的[Haberman's Survival Data Set]

Abstract: Dataset contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer

Data Set Characteristics:	Multivariate	**Number of Instances:**	306	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	3	**Date Donated**	1999-03-04
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	156150

数据描述：该数据集包含了一项研究的病例，该研究是在1958年至1970年期间在芝加哥大学比林斯医院(University of Chicago's Billings Hospital)进行的，研究对象是接受过乳腺癌手术的患者的存活率。

特征描述（3维特征）：

1. 手术病人年龄(数字)
2. 患者手术年(1900年，数字)
3. 腋窝淋巴结数目(数字)
4. 生存状态(类属性)

   ——1 = 患者存活5年或更长时间 **(在本实验中为1)**

   ——2 = 病人在5年内死亡 **（在本实验中为0）**

由于数据分类标签为1和2，模型中对0,1分类，则数据初始处理时将1->1, 2->0.

如果画图表示3维特征结果，在空间中很难看出分类是否正确合理，同理**多维特征时，画图较为复杂，故用accuracy，precision，recall评估模型的好坏。**

- UCI数据牛顿法测试（对于正例1）

	accuracy	precision	recall
1	0.652	0.690	0.920
2	0.717	0.752	0.941
3	0.771	0.790	0.971

- UCI梯度下降法测试(学习速率 = 0.00001 , 对于正例1)：

	accuracy	precision	recall
1	0.684	0.705	0.952
2	0.717	0.735	0.969
3	0.739	0.752	0.985

结论

- **加入正则项后**，对于收敛的判断更加合理，尤其在梯度下降法中，**避免了循环次数过多，产生过拟合的现象。**
- 最初创建模型时，使用100个样本点的 `data.txt` 训练，发现 `precision` 和 `recall` 同时都较高，不符合 `precision-recall曲线` **。采用306个样本点的** `haberman.txt` **训练后，** `precision``和recall` **符合** `precision-recall` **曲线，且可以看出模型表现良好，没有发生其中一个值非常高的情况，而是两者比较折中。**
- 逻辑回归的强前提条件是各特征之间相互独立，当特征不满足时，分类效果不佳。

参考文献

代码信条-Code Belief

Coursera公开课笔记: 斯坦福大学机器学习第六课"逻辑回归(Logistic Regression)"

Exercise: Logistic Regression and Newton's Method

牛顿法与拟牛顿法学习笔记（一）牛顿法

《统计学习方法》——李航

Andrew Ng Coursera机器学习课程

附录：源代码

详见压缩包内的 `GD_logisticRegression.py` (梯度下降法)， `newton_logisticRegression.py` (牛顿法)

每个文件内包含正则化与非正则化的模型代码，**运行后默认展示对二维特征`data.txt`的分类结果（非正则与正则）**