哈尔滨工业大学计算机科学与技术学院

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：1190201303
姓名：　王艺丹

一、 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。

二、 实验要求及实验环境

2.1 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。

验证：

1. 可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一组实际数据加以测试。

2.2 实验环境

Windows 10；Anaconda 4.8.4；python 3.7.4；jupyter notebook 6.0.1

三、 概念设计思想（主要算法及数据结构）

主要推导：

分类器做分类问题的实质,就是是预测一个已知样本的位置标签，即$P(Y=1|X=<X_1,\ldots,X_n>)$，按照朴素贝叶斯方法，可以用贝叶斯概率公式将其转化为类条件概率(似然)和类概率的积，本次实验直接求该概率。

假设在二分类的情况下，推导逻辑回归表达式。令

$X=<X_1,X_2,\ldots,X_n>, Y\in\{0,1\}$，则有：

$$
\begin{aligned}
P(Y=1|X) &= \frac{P(X|Y=1)P(Y=1)}{P(X|Y=1)P(Y=1)+P(X|Y=0)P(Y=0)} \\
&= \frac{1}{1+\dfrac{P(X|Y=0)P(Y=0)}{P(X|Y=1)P(Y=1)}} \\
&= \frac{1}{1+\exp(\ln\dfrac{P(X|Y=0)P(Y=0)}{P(X|Y=1)P(Y=1)})} \\
&= \frac{1}{1+\exp(\ln\dfrac{P(X|Y=0)}{P(X|Y=1)}+\ln\dfrac{P(Y=0)}{P(Y=1)})}
\end{aligned}
$$

如果符合朴素贝叶斯假设，X 的每个维度 $X_1,X_2,\ldots,X_n$ 之间相互独立，则有

$$P(X|Y=0)=\prod_{i=1}^{n}P(X=X_i|Y=0)$$
$$P(X|Y=1)=\prod_{i=1}^{n}P(X=X_i|Y=1)$$

代入得

$$P(Y=1|X)=\frac{1}{1+\exp(\ln\frac{\prod_{i=1}^{n}P(X=X_i|Y=0)}{\prod_{i=1}^{n}P(X=X_i|Y=1)}+\ln\frac{P(Y=0)}{P(Y=1)})}$$

化简得

$$P(Y=1|X)=\frac{1}{1+\exp(\sum_{i=1}^{n}\ln\frac{P(X=X_i|Y=0)}{P(X=X_i|Y=1)}+\ln\frac{P(Y=0)}{P(Y=1)})}$$

若 $X_1,X_2,...,X_n$ 符合高斯分布 $N(\mu_{ik},\sigma_i)$，Y 符合伯努利分布，即：

$$P(X_i|Y=0)=\frac{e^{-\frac{(X_i-\mu_{i0})^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i}$$
$$P(X_i|Y=1)=\frac{e^{-\frac{(X_i-\mu_{i1})^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i}$$
$$P(Y=1)=\pi$$
$$P(Y=0)=1-\pi$$

代入原式得，

$$P(Y=1|X)=\frac{1}{1+\exp(\sum_{i=1}^{n}\ln(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^{\ 2}}X_i+\frac{\mu_{i1}^{\ 2}-\mu_{i0}^{\ 2}}{2\sigma_i^{\ 2}})+\ln\frac{1-\pi}{\pi})}$$

提出常数项后，上式可表示为这样的形式：

$$P(Y=1|X)=\frac{1}{1+\exp(\sum_{i=1}^{n}w_iX_i+w_0)}$$

因此有

$$P(Y=0|X)=\frac{\exp(\sum_{i=1}^{n}w_iX_i+w_0)}{1+\exp(\sum_{i=1}^{n}w_iX_i+w_0)}$$

逻辑回归的基本思想即为通过上面两个公式求算分类概率，基本方法即为通过训练数据 X 估计系数 $w_i(i=0,1,...,n)$，从而对测试数据进行分类。

如果采用最大似然（MLE）方法估计参数 $\mathbf{w}$，则

$$\begin{aligned}\mathbf{w}_{MLE}&=\arg\max P(Y,X|\mathbf{w})\\&=\arg\max\prod_{l}^{N}P(Y^l,X^l|\mathbf{w})\end{aligned}$$

但是由于每个样本 Y^l,X^l 的联合分布难以估计，实际中并未采用 MLE 方法进行估计，反而将 X^l 后置，用最大条件似然（MCLE）估计参数 $\mathbf{w}$：

$$\begin{aligned}\mathbf{w}_{MCLE}&=\arg\max P(Y|X,\mathbf{w})\\&=\arg\max\prod_{l}^{N}P(Y^l|X^l,\mathbf{w})\\&=\arg\max\sum_{l}^{N}\ln P(Y^l|X^l,\mathbf{w})\end{aligned}$$

令

$$P(Y=0|X)=\frac{1}{1+\exp(\sum_{i=1}^{n}w_iX_i+w_0)}$$

$$P(Y=1|X)=\frac{\exp(\sum_{i=1}^{n}w_iX_i+w_0)}{1+\exp(\sum_{i=1}^{n}w_iX_i+w_0)}$$

因此，损失函数设定为：

$$\begin{aligned}l(\mathbf{w})&=-\sum_{l}^{N}\ln P(Y^l|X^l,\mathbf{w})\\&=-\sum_{l}^{N}(Y^l\ln P(Y^l=1|X^l,\mathbf{w})+(1-Y^l)\ln P(Y^l=0|X^l,\mathbf{w}))\\&=-\sum_{l}^{N}(Y^l\ln\frac{P(Y^l=1|X^l,\mathbf{w})}{P(Y^l=0|X^l,\mathbf{w})}+\ln P(Y^l=0|X^l,\mathbf{w}))\\&=-\sum_{l}^{N}(Y^l(\sum_{i=1}^{n}w_iX_i+w_0)-\ln(1+\exp(\sum_{i=1}^{n}w_iX_i+w_0))\end{aligned}$$

对参数 $\mathbf{w}$ 求导得到，

$$\frac{\partial l(\mathbf{w})}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

若采用带有正则化项的损失函数，则

$$l(\mathbf{w}) = -\sum_l^N (Y^l (\sum_{i=1}^n w_i X_i + w_0) - \ln(1 + \exp(\sum_{i=1}^n w_i X_i + w_0)) + \frac{\lambda}{2} \| \mathbf{w}^T \mathbf{w} \|$$

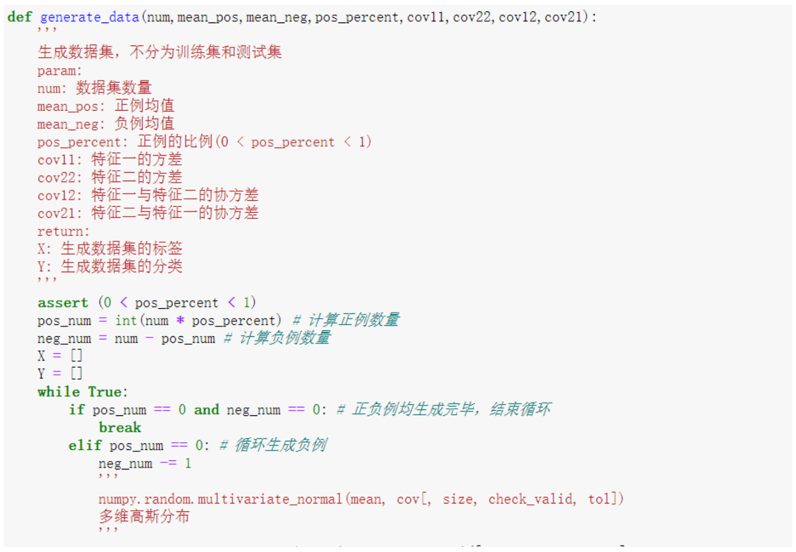
$$\frac{\partial l(\mathbf{w})}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W) + \lambda w_i$$

使用梯度下降进行更新：

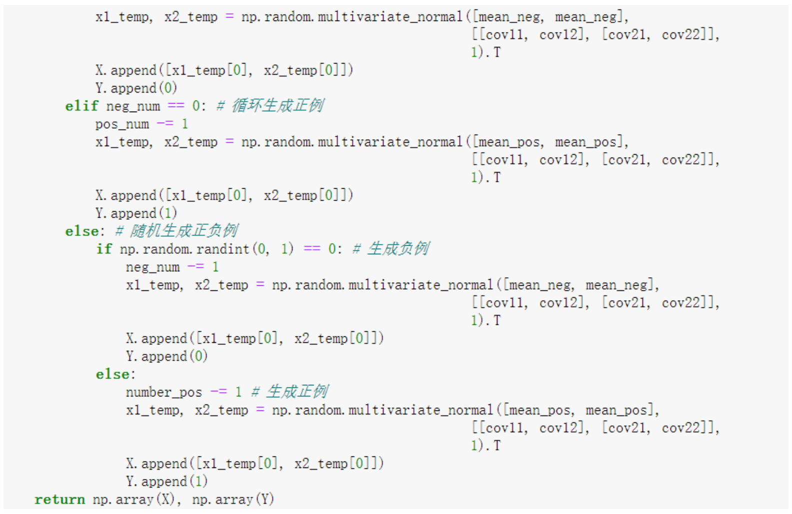
$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}$$

3.1 生成数据

主要思想即为根据给定的数据个数与正负例比例，随机循环生成正负例，改变其特征间的协方差(协方差矩阵的反对角线上是否相等且为 0)可以分别生成满足朴素贝叶斯与不满足的数据集。



```python
def generate_data(num, mean_pos, mean_neg, pos_percent, cov11, cov22, cov12, cov21):
    '''
    生成数据集，不分为训练集和测试集
    param:
    num: 数据集数量
    mean_pos: 正例均值
    mean_neg: 负例均值
    pos_percent: 正例的比例(0 < pos_percent < 1)
    cov11: 特征一的方差
    cov22: 特征二的方差
    cov12: 特征一与特征二的协方差
    cov21: 特征二与特征一的协方差
    return:
    X: 生成数据集的标签
    Y: 生成数据集的分类
    '''
    assert (0 < pos_percent < 1)
    pos_num = int(num * pos_percent) # 计算正例数量
    neg_num = num - pos_num # 计算负例数量
    X = []
    Y = []
    while True:
        if pos_num == 0 and neg_num == 0: # 正负例均生成完毕，结束循环
            break
        elif pos_num == 0: # 循环生成负例
            neg_num -= 1
            '''
            numpy.random.multivariate_normal(mean, cov[, size, check_valid, tol])
            多维高斯分布
            '''
```



```python
            x1_temp, x2_temp = np.random.multivariate_normal([mean_neg, mean_neg],
                                                             [[cov11, cov12], [cov21, cov22]],
                                                             1).T
            X.append([x1_temp[0], x2_temp[0]])
            Y.append(0)
        elif neg_num == 0: # 循环生成正例
            pos_num -= 1
            x1_temp, x2_temp = np.random.multivariate_normal([mean_pos, mean_pos],
                                                             [[cov11, cov12], [cov21, cov22]],
                                                             1).T
            X.append([x1_temp[0], x2_temp[0]])
            Y.append(1)
        else: # 随机生成正负例
            if np.random.randint(0, 1) == 0: # 生成负例
                neg_num -= 1
                x1_temp, x2_temp = np.random.multivariate_normal([mean_neg, mean_neg],
                                                             [[cov11, cov12], [cov21, cov22]],
                                                             1).T
                X.append([x1_temp[0], x2_temp[0]])
                Y.append(0)
            else:
                number_pos -= 1 # 生成正例
                x1_temp, x2_temp = np.random.multivariate_normal([mean_pos, mean_pos],
                                                             [[cov11, cov12], [cov21, cov22]],
                                                             1).T
                X.append([x1_temp[0], x2_temp[0]])
                Y.append(1)
    return np.array(X), np.array(Y)
```

3.2 分割测试集与数据集

思想较为简单，不作赘述，看代码及注释即可：

```python
def split_data(X,Y,test_percent):
    '''
    将数据集分为训练集和测试集
    param:
    X: 数据集标签
    Y: 数据集分类
    test_percent: 测试集占比
    return:
    np.array(x_train): 训练集标签
    np.array(y_train): 训练集分类
    np.array(x_test): 测试集标签
    np.array(y_test): 测试集分类
    '''
    num = len(X) # 获取全部数据集数量
    num_test = int(num * test_percent) # 计算测试集数量
    num_train = num - num_test #计算训练集数量
    x_train = [] # 初始化
    x_test = []
    y_train = []
    y_test = []
    for i in range(num):
        if num_test > 0: # 如果测试集还未生成完毕
            if num_train == 0 or np.random.randint(2) == 0: #训练集生成完毕/随机数=0, 生成测试集
                num_test -= 1
                x_test.append(X[i])
                y_test.append(Y[i])
            else: # 生成训练集
                num_train -= 1
                x_train.append(X[i])
                y_train.append(Y[i])
        else: # 测试集生成完毕, 其余全部归为训练集
            num_train -= 1
            x_train.append(X[i])
            y_train.append(Y[i])
    return np.array(x_train), np.array(y_train), np.array(x_test), np.array(y_test)
```

3.3 计算准确率

思想较为简单，不作赘述，看代码及注释即可：

```python
def cal_accuracy(x, y, w):
    num_data = x.shape[0]
    num_feature = x.shape[1]
    correct_count = 0
    X = np.ones((num_data, num_feature + 1))  # 构造X矩阵, 第一维都设置成1, 方便与w相乘
    X[:, 1:num_feature + 1] = x # 提取x
    for i in range(num_data):
        label = 0
        if w @ X[i].T >= 0:
            label = 1
        if label == y[i]:
            correct_count += 1
    accuracy = correct_count / num_data
    return accuracy
```

3.4 梯度下降

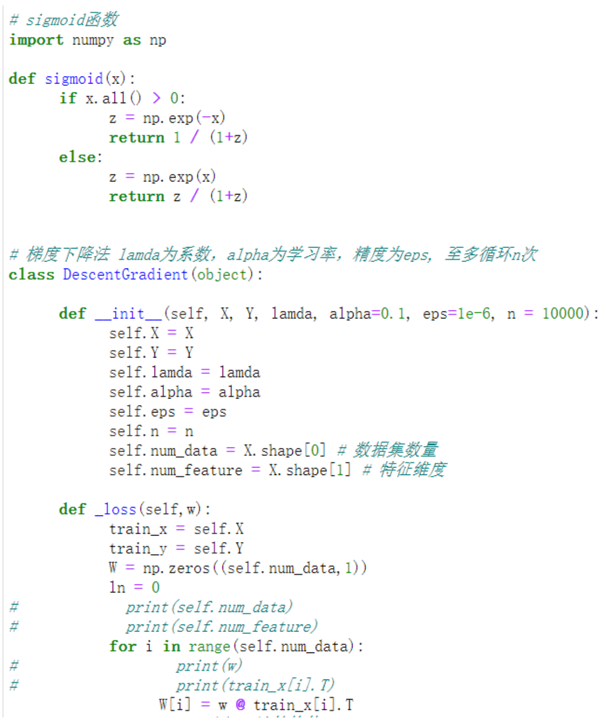
带有惩罚项的梯度下降法参数估计，与上一节中不带正则项的参数估计推导过程几乎相同，这里仅仅给出带有正则项的参数更新公式如下

$$w_{i+1} = w_i + \eta\lambda w_i + \eta X^T[\boldsymbol{y} - h(X^T w)]$$

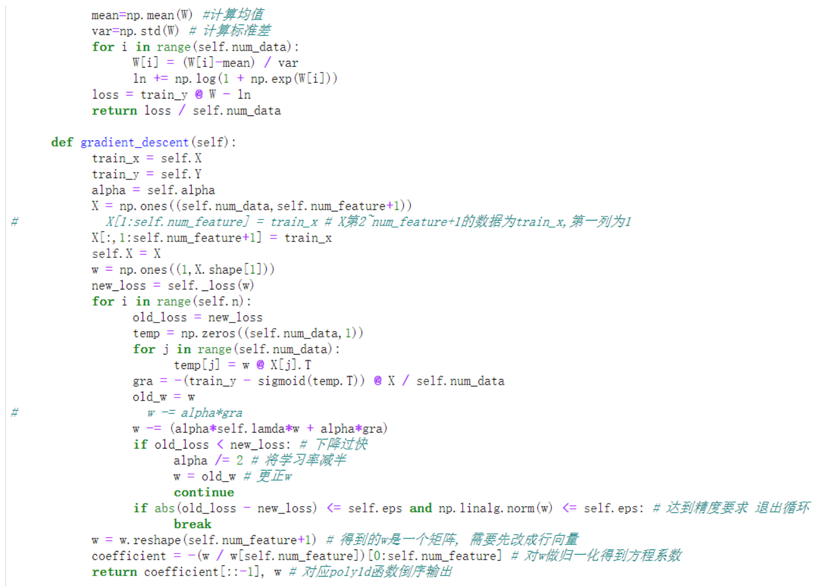
伪代码如下：

```
梯度下降法
Require：学习率$lr$和初始参数$\boldsymbol{w}$
Repeat
    梯度计算 $\boldsymbol{g} = \lambda\boldsymbol{w} + X^T[\boldsymbol{y} - h(X^T w)]$
    参数更新 $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{\eta} * \boldsymbol{g}$
Until 达到收敛条件 $\|\boldsymbol{g}\| < eps$
```

具体实现：



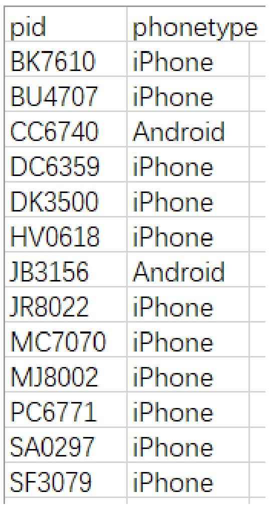
```python
# sigmoid函数
import numpy as np

def sigmoid(x):
    if x.all() > 0:
        z = np.exp(-x)
        return 1 / (1+z)
    else:
        z = np.exp(x)
        return z / (1+z)


# 梯度下降法 lamda为系数，alpha为学习率，精度为eps, 至多循环n次
class DescentGradient(object):

    def __init__(self, X, Y, lamda, alpha=0.1, eps=1e-6, n = 10000):
        self.X = X
        self.Y = Y
        self.lamda = lamda
        self.alpha = alpha
        self.eps = eps
        self.n = n
        self.num_data = X.shape[0] # 数据集数量
        self.num_feature = X.shape[1] # 特征维度

    def _loss(self,w):
        train_x = self.X
        train_y = self.Y
        W = np.zeros((self.num_data,1))
        ln = 0
#         print(self.num_data)
#         print(self.num_feature)
        for i in range(self.num_data):
#             print(w)
#             print(train_x[i].T)
            W[i] = w @ train_x[i].T
```



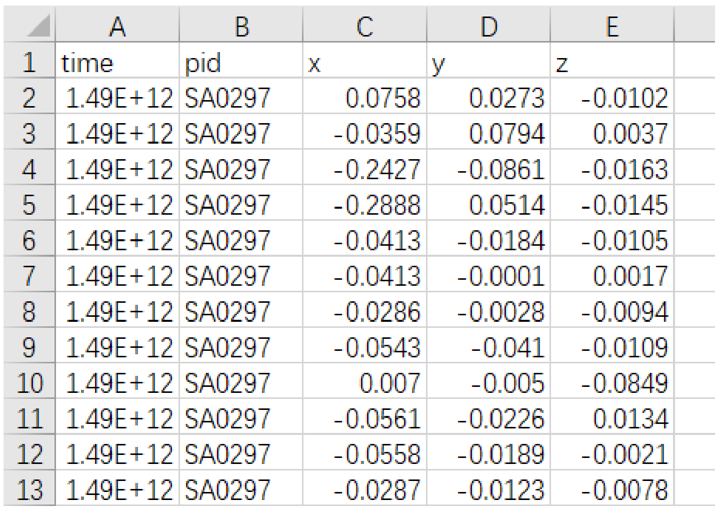
```python
        mean=np.mean(W) #计算均值
        var=np.std(W) # 计算标准差
        for i in range(self.num_data):
            W[i] = (W[i]-mean) / var
            ln += np.log(1 + np.exp(W[i]))
        loss = train_y @ W - ln
        return loss / self.num_data

    def gradient_descent(self):
        train_x = self.X
        train_y = self.Y
        alpha = self.alpha
        X = np.ones((self.num_data,self.num_feature+1))
#         X[1:self.num_feature] = train_x # X第2~num_feature+1的数据为train_x,第一列为1
        X[:,1:self.num_feature+1] = train_x
        self.X = X
        w = np.ones((1,X.shape[1]))
        new_loss = self._loss(w)
        for i in range(self.n):
            old_loss = new_loss
            temp = np.zeros((self.num_data,1))
            for j in range(self.num_data):
                temp[j] = w @ X[j].T
            gra = -(train_y - sigmoid(temp.T)) @ X / self.num_data
            old_w = w
#             w -= alpha*gra
            w -= (alpha*self.lamda*w + alpha*gra)
            if old_loss < new_loss: # 下降过快
                alpha /= 2 # 将学习率减半
                w = old_w # 更正w
                continue
            if abs(old_loss - new_loss) <= self.eps and np.linalg.norm(w) <= self.eps: # 达到精度要求 退出循环
                break
        w = w.reshape(self.num_feature+1) # 得到的w是一个矩阵, 需要先改成行向量
        coefficient = -(w / w[self.num_feature])[0:self.num_feature] # 对w做归一化得到方程系数
        return coefficient[::-1], w # 对应poly1d函数倒序输出
```

3.5 UCI 数据集处理

选取 UCI 手机数据集与银行，其中手机数据集中 pid 与手机型号关系，部分数据集展示如下：

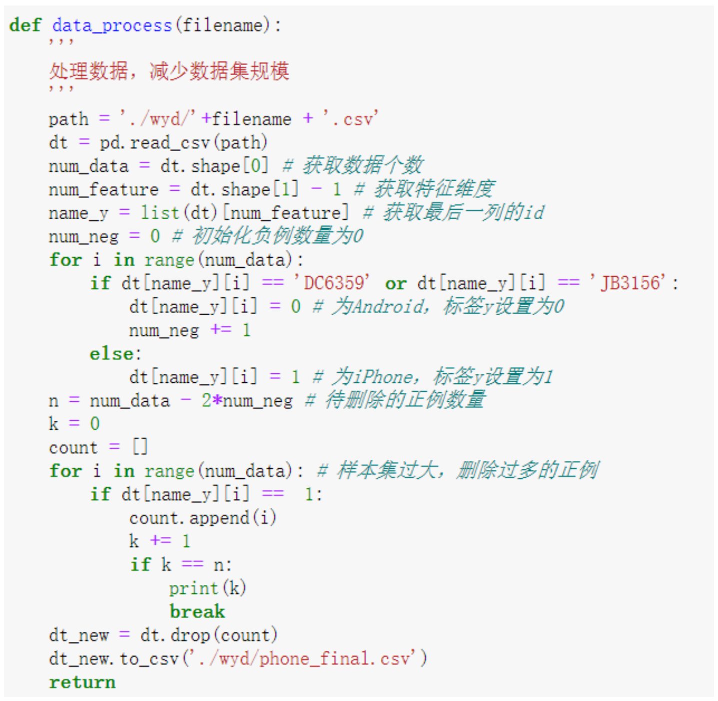


pid	phonetype
BK7610	iPhone
BU4707	iPhone
CC6740	Android
DC6359	iPhone
DK3500	iPhone
HV0618	iPhone
JB3156	Android
JR8022	iPhone
MC7070	iPhone
MJ8002	iPhone
PC6771	iPhone
SA0297	iPhone
SF3079	iPhone



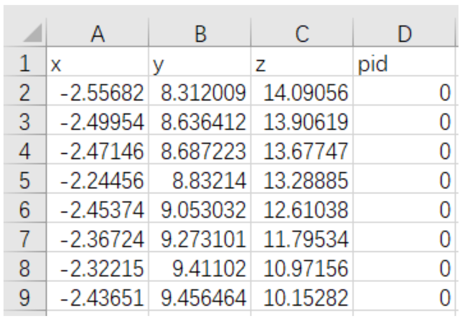
	A	B	C	D	E
1	time	pid	x	y	z
2	1.49E+12	SA0297	0.0758	0.0273	-0.0102
3	1.49E+12	SA0297	-0.0359	0.0794	0.0037
4	1.49E+12	SA0297	-0.2427	-0.0861	-0.0163
5	1.49E+12	SA0297	-0.2888	0.0514	-0.0145
6	1.49E+12	SA0297	-0.0413	-0.0184	-0.0105
7	1.49E+12	SA0297	-0.0413	-0.0001	0.0017
8	1.49E+12	SA0297	-0.0286	-0.0028	-0.0094
9	1.49E+12	SA0297	-0.0543	-0.041	-0.0109
10	1.49E+12	SA0297	0.007	-0.005	-0.0849
11	1.49E+12	SA0297	-0.0561	-0.0226	0.0134
12	1.49E+12	SA0297	-0.0558	-0.0189	-0.0021
13	1.49E+12	SA0297	-0.0287	-0.0123	-0.0078

则利用 pid 对数据集进行标签划分，iPhone 对应标签为 1，Android 标签为 0 由于样本集过大，iPhone 数据较多，最终删除部分 iPhone 数据保证正负例比例为 1：1，具体实现如下：最终数据写入 phone_final.csv 文件

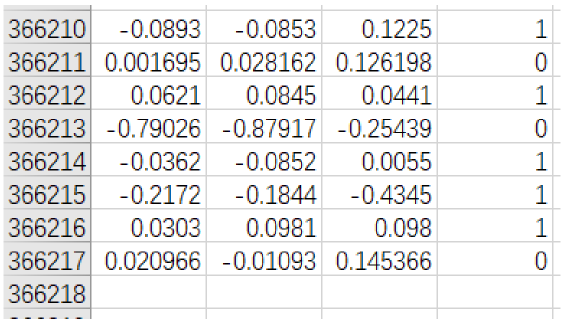


```python
def data_process(filename):
    '''
    处理数据，减少数据集规模
    '''
    path = './wyd/'+filename + '.csv'
    dt = pd.read_csv(path)
    num_data = dt.shape[0] # 获取数据个数
    num_feature = dt.shape[1] - 1 # 获取特征维度
    name_y = list(dt)[num_feature] # 获取最后一列的id
    num_neg = 0 # 初始化负例数量为0
    for i in range(num_data):
        if dt[name_y][i] == 'DC6359' or dt[name_y][i] == 'JB3156':
            dt[name_y][i] = 0 # 为Android，标签y设置为0
            num_neg += 1
        else:
            dt[name_y][i] = 1 # 为iPhone，标签y设置为1
    n = num_data - 2*num_neg # 待删除的正例数量
    k = 0
    count = []
    for i in range(num_data): # 样本集过大，删除过多的正例
        if dt[name_y][i] ==  1:
            count.append(i)
            k += 1
            if k == n:
                print(k)
                break
    dt_new = dt.drop(count)
    dt_new.to_csv('./wyd/phone_final.csv')
    return
```

处理后最终 phone_final 文件部分数据展示如下：
最终得到 366216 组数据：

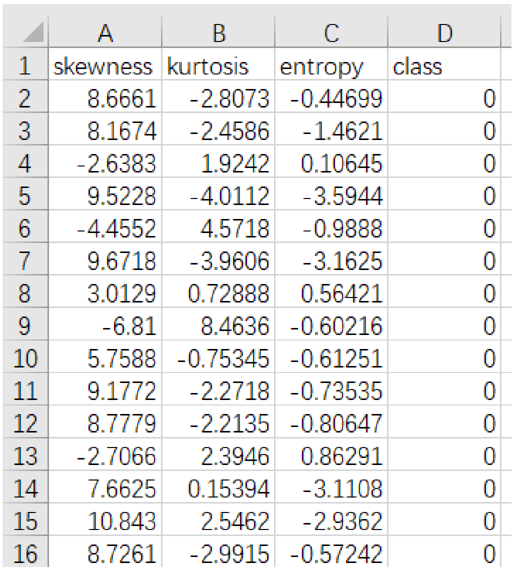


	A	B	C	D
1	x	y	z	pid
2	-2.55682	8.312009	14.09056	0
3	-2.49954	8.636412	13.90619	0
4	-2.47146	8.687223	13.67747	0
5	-2.24456	8.83214	13.28885	0
6	-2.45374	9.053032	12.61038	0
7	-2.36724	9.273101	11.79534	0
8	-2.32215	9.41102	10.97156	0
9	-2.43651	9.456464	10.15282	0

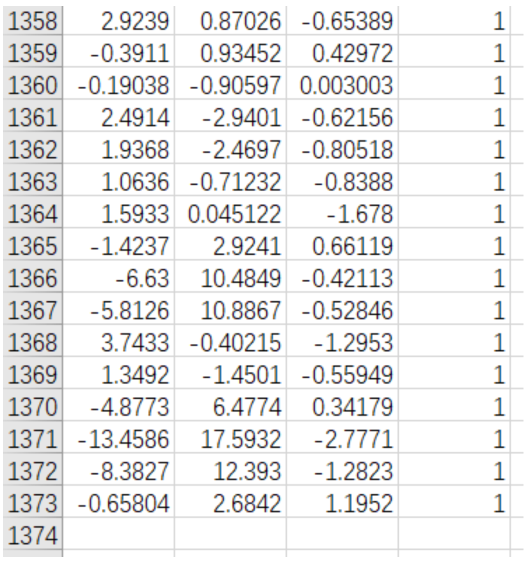


366210	-0.0893	-0.0853	0.1225	1
366211	0.001695	0.028162	0.126198	0
366212	0.0621	0.0845	0.0441	1
366213	-0.79026	-0.87917	-0.25439	0
366214	-0.0362	-0.0852	0.0055	1
366215	-0.2172	-0.1844	-0.4345	1
366216	0.0303	0.0981	0.098	1
366217	0.020966	-0.01093	0.145366	0
366218				

银行数据集处理结果部分展示如下，共 1372 组数据：



	A	B	C	D
1	skewness	kurtosis	entropy	class
2	8.6661	-2.8073	-0.44699	0
3	8.1674	-2.4586	-1.4621	0
4	-2.6383	1.9242	0.10645	0
5	9.5228	-4.0112	-3.5944	0
6	-4.4552	4.5718	-0.9888	0
7	9.6718	-3.9606	-3.1625	0
8	3.0129	0.72888	0.56421	0
9	-6.81	8.4636	-0.60216	0
10	5.7588	-0.75345	-0.61251	0
11	9.1772	-2.2718	-0.73535	0
12	8.7779	-2.2135	-0.80647	0
13	-2.7066	2.3946	0.86291	0
14	7.6625	0.15394	-3.1108	0
15	10.843	2.5462	-2.9362	0
16	8.7261	-2.9915	-0.57242	0



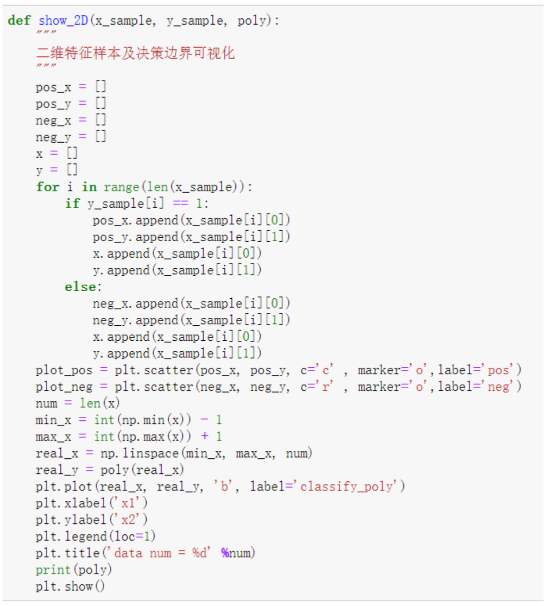
1358	2.9239	0.87026	-0.65389	1
1359	-0.3911	0.93452	0.42972	1
1360	-0.19038	-0.90597	0.003003	1
1361	2.4914	-2.9401	-0.62156	1
1362	1.9368	-2.4697	-0.80518	1
1363	1.0636	-0.71232	-0.8388	1
1364	1.5933	0.045122	-1.678	1
1365	-1.4237	2.9241	0.66119	1
1366	-6.63	10.4849	-0.42113	1
1367	-5.8126	10.8867	-0.52846	1
1368	3.7433	-0.40215	-1.2953	1
1369	1.3492	-1.4501	-0.55949	1
1370	-4.8773	6.4774	0.34179	1
1371	-13.4586	17.5932	-2.7771	1
1372	-8.3827	12.393	-1.2823	1
1373	-0.65804	2.6842	1.1952	1
1374				

3.6 UCI 数据读入

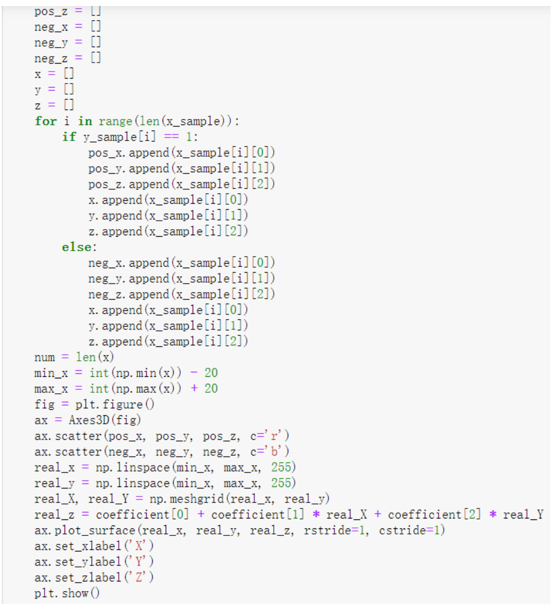
```
def data_read(filename):
    path = './wyd/'+filename + '.csv'
    dt = pd.read_csv(path)
    num_data = dt.shape[0] # 获取数据个数
    num_feature = dt.shape[1] - 1 # 获取特征维度
    name_y = list(dt)[num_feature] # 获取最后一列的id
    X = np.array(dt.drop([name_y],axis = 1),dtype=float)
    Y = np.array(dt[name_y])
    return X,Y
```

3.7 决策结果可视化

二维/三维数据集可视化：



```
def show_2D(x_sample, y_sample, poly):
    """
    二维特征样本及决策边界可视化
    """
    pos_x = []
    pos_y = []
    neg_x = []
    neg_y = []
    x = []
    y = []
    for i in range(len(x_sample)):
        if y_sample[i] == 1:
            pos_x.append(x_sample[i][0])
            pos_y.append(x_sample[i][1])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
        else:
            neg_x.append(x_sample[i][0])
            neg_y.append(x_sample[i][1])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
    plot_pos = plt.scatter(pos_x, pos_y, c='c' , marker='o',label='pos')
    plot_neg = plt.scatter(neg_x, neg_y, c='r' , marker='o',label='neg')
    num = len(x)
    min_x = int(np.min(x)) - 1
    max_x = int(np.max(x)) + 1
    real_x = np.linspace(min_x, max_x, num)
    real_y = poly(real_x)
    plt.plot(real_x, real_y, 'b', label='classify_poly')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(loc=1)
    plt.title('data num = %d' %num)
    print(poly)
    plt.show()
```



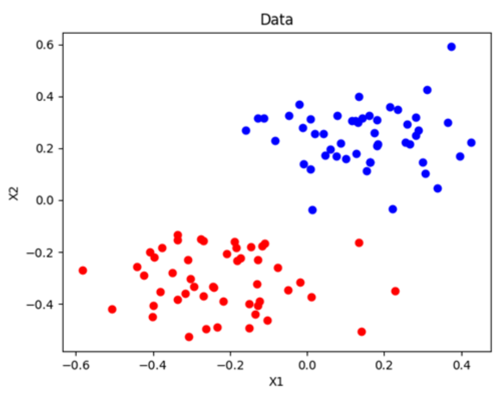
```
    pos_z = []
    neg_x = []
    neg_y = []
    neg_z = []
    x = []
    y = []
    z = []
    for i in range(len(x_sample)):
        if y_sample[i] == 1:
            pos_x.append(x_sample[i][0])
            pos_y.append(x_sample[i][1])
            pos_z.append(x_sample[i][2])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
            z.append(x_sample[i][2])
        else:
            neg_x.append(x_sample[i][0])
            neg_y.append(x_sample[i][1])
            neg_z.append(x_sample[i][2])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
            z.append(x_sample[i][2])
    num = len(x)
    min_x = int(np.min(x)) - 20
    max_x = int(np.max(x)) + 20
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(pos_x, pos_y, pos_z, c='r')
    ax.scatter(neg_x, neg_y, neg_z, c='b')
    real_x = np.linspace(min_x, max_x, 255)
    real_y = np.linspace(min_x, max_x, 255)
    real_X, real_Y = np.meshgrid(real_x, real_y)
    real_z = coefficient[0] + coefficient[1] * real_X + coefficient[2] * real_Y
    ax.plot_surface(real_x, real_y, real_z, rstride=1, cstride=1)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()
```

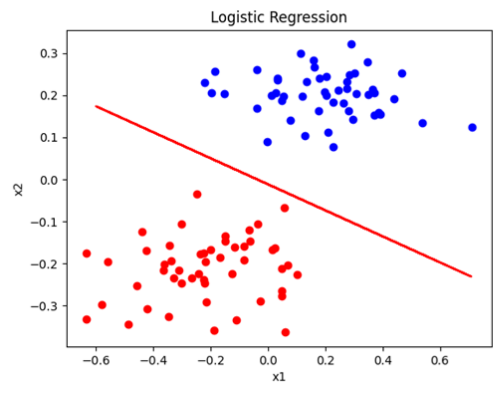
四、 实验结果与分析

```
def __init__(self, X, Y, lamda=np.exp(-7), alpha=0.01, eps=1e-6, n = 10000):
```

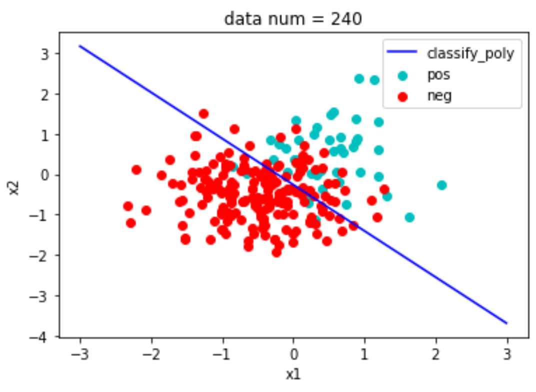
均选取学习率$\alpha = 0.01$，精度$eps = 1 \times 10^{-6}$，正则系数$\lambda = e^{-7}$, 迭代次数上限 $n = 10000$，带正则项的梯度下降法

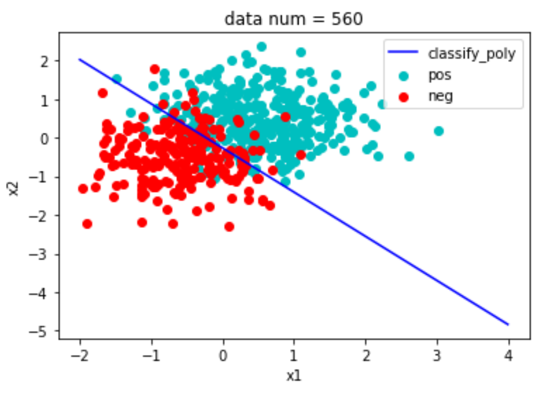
4.1 满足朴素贝叶斯，人工生成数据集结果：





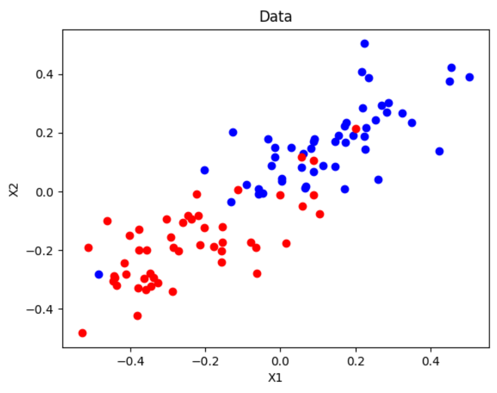
在测试集应用决策边界，准确率为 97.16%

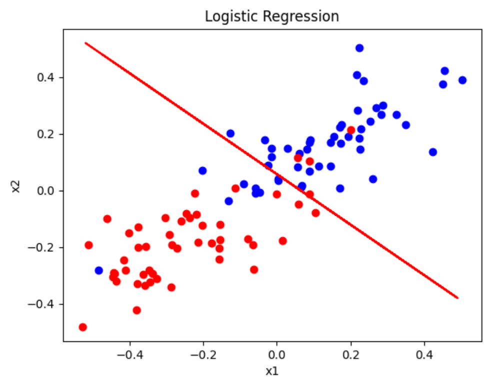




测试集准确率为 86.54%　　　　　　　　测试集准确率为 83.17%

4.2 不满足朴素贝叶斯，人工生成数据集结果：





在测试集应用决策边界，准确率为 87.72%

4.2.1 删除正则项，决策结果对比

利用其他数据集，删除增设正则项，准确率如下：

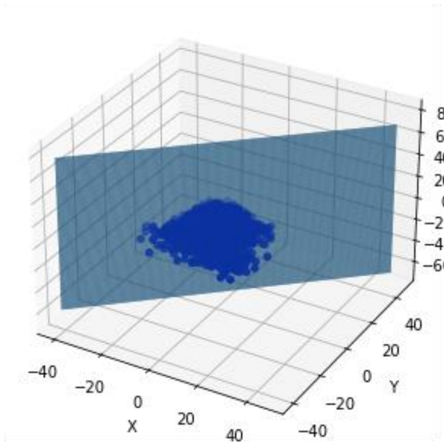
删除正则项，准确率：　0.7712895377128953

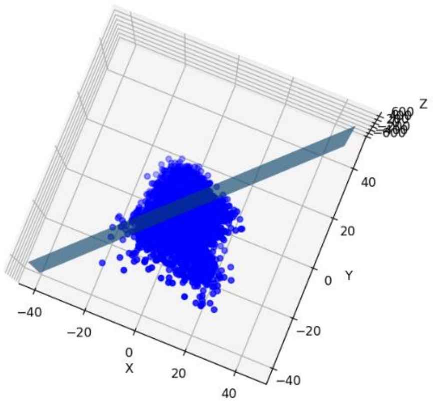
添加正则项，准确率：　0.7980535279805353

可以看出，添加惩罚项使得准确率仅仅提高了 0.03%，并且观察参数可以发现，正则化项使参数值减少，模型复杂度降低，对于减缓模型的过拟合有一定的作用。然而，经过多次实验发现，在数据集较少的情况下，逻辑回归模型的过拟合问题并不严重，正则化项的优化作用微乎其微，同时由于参数原本量级就较小，正则化项对参数的影响也很小，多数情况下有无正则化项都不会影响决策面的选择和分类的准确率。随着训练数据的增加，更难产生过拟合现象，正则化项的影响更弱

4.3 UCI 数据集

4.3.1 手机数据集

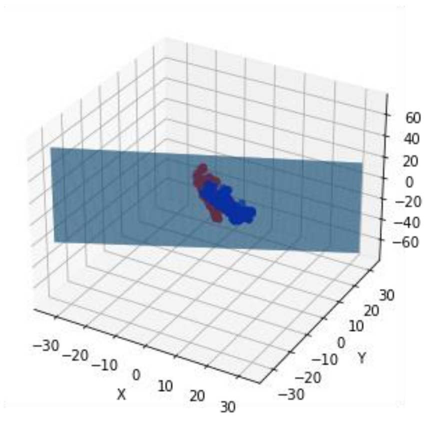


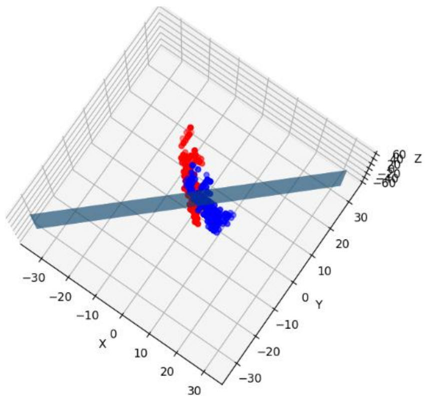


仔细看可以看出数据集颜色不同，但由于数据过多，则各特征之间精度插值过小，颜色深浅重叠导致难以区分，测试集准确率达到： `0.6802228209422558`

由于该数据集跑了一整个多下午，故不调整颜色重新跑代码了…，该数据集数据间相差过小且精度(小数点位数)过大，**且各维度之间不满足朴素贝叶斯**，导致模型决策结果并不是很理想，进一步选取数据规模较小且各维度间相关性较弱的银行数据集。

4.3.2 银行数据集





准确率达到：

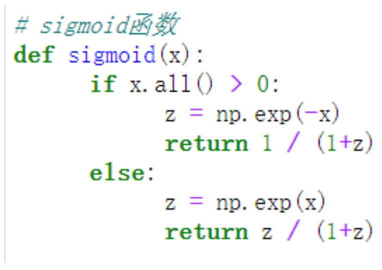
```
0.8166666666666667
```

4.4 对于数据溢出的处理技巧

当数据量级过大，在 sigmoid 函数中取 e 的指数操作很容易导致数据溢出。经查阅相关资料和进行数学推导，发现可以通过改变 sigmoid 函数的计算方式来防止数据溢出。

$$sigmoid(x)=\begin{cases}\dfrac{1}{1+e^{-x}},x\geq 0\\ \dfrac{e^{x}}{1+e^{x}},x<0\end{cases}$$

代码实现如下：



```
# sigmoid函数
def sigmoid(x):
    if x.all() > 0:
        z = np.exp(-x)
        return 1 / (1+z)
    else:
        z = np.exp(x)
        return z / (1+z)
```

五、 结论

✧ 在数据量很小时，逻辑回归也会遭受轻微的过拟合现象，可以通过添加正则化项加以缓解。但是逻辑回归的过拟合现象并不严重，只要数据集稍微增加，就基本不存在过拟合现象， 正则化项的影响也十分微小。

✧ 对于线性可分的数据集，逻辑回归可以找到一个完美的决策面进行分类；对于非线性可分的数据集，逻辑回归只能找出一个使损失尽可能小的一个线性决策面，并不能很好地处理这种数据集。事实上，是否符合朴素贝叶斯假设对分类结果的影响也是基于其是否线性可分而言的。如果数据并非线性可分，即使符合朴素贝叶斯假设，也不能被逻辑回归完美分类。但逻辑回归总会尽可能找到与训练数据更接近的决策面，使更多的点被正确分类。

六、 参考文献

[1] 李航, 《统计学习方法》(第三版)

[2] 周志华, 《机器学习》

[3] Pattern Recognition and Machine Learning.

[4] Strang, G. 2006. Linear algebra and its applications. Belmont, CA: Thomson, Brooks/Cole.

[5] Gradient descent wiki

[6] https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/

七、 附录：源代码（带注释）

gra_descent_wyd.py

```
import numpy as np


# sigmoid 函数
def sigmoid(x):
    if x.all() > 0:
        z = np.exp(-x)
        return 1 / (1+z)
    else:
        z = np.exp(x)
        return z / (1+z)



# 梯度下降法 lamda 为系数，alpha 为学习率，精度为 eps，至多循环 n 次
class DescentGradient(object):

    def __init__(self, X, Y, lamda=np.exp(-7), alpha=0.01, eps=1e-6, n
= 10000):
        self.X = X
        self.Y = Y
        self.lamda = lamda
        self.alpha = alpha
        self.eps = eps
        self.n = n
        self.num_data = X.shape[0] # 数据集数量
        self.num_feature = X.shape[1] # 特征维度

    def _loss(self,w):
        train_x = self.X
        train_y = self.Y
        W = np.zeros((self.num_data,1))
        ln = 0
#         print(self.num_data)
#         print(self.num_feature)
        for i in range(self.num_data):
#             print(w)
#             print(train_x[i].T)
            W[i] = w @ train_x[i].T
        mean=np.mean(W) #计算均值
        var=np.std(W) # 计算标准差
        for i in range(self.num_data):
            W[i] = (W[i]-mean) / var
            ln += np.log(1 + np.exp(W[i]))
        loss = train_y @ W - ln
        return loss / self.num_data

    def gradient_descent(self):
        train_x = self.X
        train_y = self.Y
        alpha = self.alpha
        X = np.ones((self.num_data,self.num_feature+1))
#         X[1:self.num_feature] = train_x # X第2~num_feature+1的数据为t
rain_x,第一列为1
        X[:,1:self.num_feature+1] = train_x
        self.X = X
        w = np.ones((1,X.shape[1]))
        new_loss = self._loss(w)
        for i in range(self.n):
            old_loss = new_loss

            temp = np.zeros((self.num_data,1))

            for j in range(self.num_data):

                temp[j] = w @ X[j].T

            gra = -(train_y - sigmoid(temp.T)) @ X / self.num_data

            old_w = w

#             w -= alpha*gra

            w -= (alpha*self.lamda*w + alpha*gra)

            if old_loss < new_loss: # 下降过快

                alpha /= 2 # 将学习率减半

                w = old_w # 更正 w

                continue

            if abs(old_loss - new_loss) <= self.eps and np.linalg.norm
(w) <= self.eps: # 达到精度要求 退出循环

                break

        w = w.reshape(self.num_feature+1) # 得到的 w 是一个矩阵，需要先改成行
向量

        coefficient = -(w / w[self.num_feature])[0:self.num_feature] #
对 w 做归一化得到方程系数

        return coefficient[::-1], w # 对应 poly1d 函数倒序输出
```

logistics regression.ipynb

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from wyd import gra_descent_wyd

from mpl_toolkits.mplot3d import Axes3D

# 极大似然估计

def likelihood(train_x,train_y,w):

    num = train_x.shape[0]
    W = np.zeros((num,1))

    ln = 0

    for i in range(num):

        W[i] = np.dot(w,train_x[i].T)

        ln += np.log(1+np.exp(W[i]))

    return np.dot(train_y,W)-ln

def generate_data(num,mean_pos,mean_neg,pos_percent,cov11,cov22,cov1
2,cov21):

    '''

    生成数据集，不分为训练集和测试集

    param:

    num: 数据集数量

    mean_pos: 正例均值

    mean_neg: 负例均值

    pos_percent: 正例的比例(0 < pos_percent < 1)

    cov11: 特征一的方差

    cov22: 特征二的方差

    cov12: 特征一与特征二的协方差

    cov21: 特征二与特征一的协方差

    return:

    X: 生成数据集的标签

    Y: 生成数据集的分类

    '''

    assert (0 < pos_percent < 1)

    pos_num = int(num * pos_percent) # 计算正例数量

    neg_num = num - pos_num # 计算负例数量

    X = []

    Y = []

    while True:

        if pos_num == 0 and neg_num == 0: # 正负例均生成完毕，结束循环
```

```
            break
        elif pos_num == 0: # 循环生成负例
            neg_num -= 1
            '''
            numpy.random.multivariate_normal(mean, cov[, size, check_va
lid, tol])
            多维高斯分布
            '''
            x1_temp, x2_temp = np.random.multivariate_normal([mean_neg,
 mean_neg],
                                                              [[cov11, cov12],
[cov21, cov22]],
                                                              1).T
            X.append([x1_temp[0], x2_temp[0]])
            Y.append(0)
        elif neg_num == 0: # 循环生成正例
            pos_num -= 1
            x1_temp, x2_temp = np.random.multivariate_normal([mean_pos,
 mean_pos],
                                                              [[cov11, cov12],
[cov21, cov22]],
                                                              1).T
            X.append([x1_temp[0], x2_temp[0]])
            Y.append(1)
        else: # 随机生成正负例
            if np.random.randint(0, 1) == 0: # 生成负例
                neg_num -= 1
                x1_temp, x2_temp = np.random.multivariate_normal([mean_n
eg, mean_neg],
                                                              [[cov11, cov12],
[cov21, cov22]],
                                                              1).T
                X.append([x1_temp[0], x2_temp[0]])
                Y.append(0)
            else:
                number_pos -= 1 # 生成正例
                x1_temp, x2_temp = np.random.multivariate_normal([mean_p
os, mean_pos],
                                                    [[cov11, cov12],
[cov21, cov22]],
                                                    1).T
                X.append([x1_temp[0], x2_temp[0]])
                Y.append(1)
    return np.array(X), np.array(Y)
def split_data(X,Y,test_percent):
    '''
    将数据集分为训练集和测试集
    param:
    X: 数据集标签
    Y: 数据集分类
    test_percent: 测试集占比
    return:
    np.array(x_train): 训练集标签
    np.array(y_train): 训练集分类
    np.array(x_test): 测试集标签
    np.array(y_test): 测试集分类
    '''
    num = len(X) # 获取全部数据集数量
    num_test = int(num * test_percent) # 计算测试集数量
    num_train = num - num_test #计算训练集数量
    x_train = [] # 初始化
    x_test = []
    y_train = []
    y_test = []
    for i in range(num):
        if num_test > 0: # 如果测试集还未生成完毕
            if num_train == 0 or np.random.randint(2) == 0: #训练集生成完毕/随机数=0，生成测试集
                num_test -= 1
                x_test.append(X[i])
                y_test.append(Y[i])
            else: # 生成训练集
                num_train -= 1
                x_train.append(X[i])
                y_train.append(Y[i])
        else: # 测试集生成完毕，其余全部归为训练集
            num_train -= 1
            x_train.append(X[i])
            y_train.append(Y[i])
    return np.array(x_train), np.array(y_train), np.array(x_test), np.array(y_test)
def cal_accuracy(x, y, w):
    num_data = x.shape[0]
    num_feature = x.shape[1]
    correct_count = 0
    X = np.ones((num_data, num_feature + 1))  # 构造 X 矩阵，第一维都设置成 1，方便与 w 相乘
    X[:, 1:num_feature + 1] = x # 提取 x
    for i in range(num_data):
        label = 0
        if w @ X[i].T >= 0:
            label = 1
        if label == y[i]:
            correct_count += 1
    accuracy = correct_count / num_data
    return accuracy
def show_2D(x_sample, y_sample, poly):
    """
    二维特征样本及决策边界可视化
    """
    pos_x = []
    pos_y = []
    neg_x = []
    neg_y = []
    x = []
    y = []
    for i in range(len(x_sample)):
        if y_sample[i] == 1:
            pos_x.append(x_sample[i][0])
            pos_y.append(x_sample[i][1])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
        else:
            neg_x.append(x_sample[i][0])
            neg_y.append(x_sample[i][1])
            x.append(x_sample[i][0])
            y.append(x_sample[i][1])
    plot_pos = plt.scatter(pos_x, pos_y, c='c' , marker='o',label='pos
')
    plot_neg = plt.scatter(neg_x, neg_y, c='r' , marker='o',label='neg
')
    num = len(x)
    min_x = int(np.min(x)) - 1
    max_x = int(np.max(x)) + 1
    real_x = np.linspace(min_x, max_x, num)
    real_y = poly(real_x)

    plt.plot(real_x, real_y, 'b', label='classify_poly')

    plt.xlabel('x1')

    plt.ylabel('x2')

    plt.legend(loc=1)

    plt.title('data num = %d' %num)

    print(poly)

    plt.show()

def show_3D(x_sample,y_sample,coefficient):

    """

    三维特征样本及决策面可视化

    """

    pos_x = []

    pos_y = []

    pos_z = []

    neg_x = []

    neg_y = []

    neg_z = []

    x = []

    y = []

    z = []

    for i in range(len(x_sample)):

        if y_sample[i] == 1:

            pos_x.append(x_sample[i][0])

            pos_y.append(x_sample[i][1])

            pos_z.append(x_sample[i][2])

            x.append(x_sample[i][0])

            y.append(x_sample[i][1])

            z.append(x_sample[i][2])

        else:
            neg_x.append(x_sample[i][0])

            neg_y.append(x_sample[i][1])

            neg_z.append(x_sample[i][2])

            x.append(x_sample[i][0])

            y.append(x_sample[i][1])

            z.append(x_sample[i][2])

    num = len(x)

    min_x = int(np.min(x)) - 20

    max_x = int(np.max(x)) + 20

    fig = plt.figure()

    ax = Axes3D(fig)

    ax.scatter(pos_x, pos_y, pos_z, c='r')

    ax.scatter(neg_x, neg_y, neg_z, c='b')

    real_x = np.linspace(min_x, max_x, 255)

    real_y = np.linspace(min_x, max_x, 255)

    real_X, real_Y = np.meshgrid(real_x, real_y)

    real_z = coefficient[0] + coefficient[1] * real_X + coefficient[2]
* real_Y

    ax.plot_surface(real_x, real_y, real_z, rstride=1, cstride=1)

    ax.set_xlabel('X')

    ax.set_ylabel('Y')

    ax.set_zlabel('Z')

    plt.show()

def data_read(filename):

    path = './wyd/'+filename + '.csv'

    dt = pd.read_csv(path)

    num_data = dt.shape[0] # 获取数据个数

    num_feature = dt.shape[1] - 1 # 获取特征维度

    name_y = list(dt)[num_feature] # 获取最后一列的 id

    X = np.array(dt.drop([name_y],axis = 1),dtype=float)
    Y = np.array(dt[name_y])

    return X,Y

'''

利用高斯分布自己生成的数据集

'''

num = 800

mean_pos = 0.5

mean_neg = -0.5

pos_percent = 0.5

cov11 = 0.5

cov22 = 0.5

cov12 = 0

cov21 = 0

X,Y= generate_data(num,mean_pos,mean_neg,pos_percent,cov11,cov22,cov1
2,cov21)

train_x, train_y, test_x, test_y = split_data(X, Y, 0.3)

coefficient, w = gra_descent_wyd.DescentGradient(train_x, train_y,lam
da=0).gradient_descent()

poly = np.poly1d(coefficient)

show_2D(train_x,train_y,poly)

show_2D(test_x,test_y,poly)

accuracy = cal_accuracy(test_x, test_y, w)

print(accuracy)


'''

uci 银行三维数据集

'''

X_uci,Y_uci = data_read('banknote')

train_x_uci, train_y_uci, test_x_uci, test_y_uci = split_data(X_uci, Y
_uci, 0.3)
```

```
co_uci, w_uci = gra_descent_wyd.DescentGradient(train_x_uci, train_y_uci,lamda=0).gradient_descent()

print(co_uci)


poly = np.poly1d(co_uci)

%matplotlib notebook

show_3D(train_x_uci,train_y_uci, co_uci)

show_3D(test_x_uci, test_y_uci, co_uci)

ac_uci = cal_accuracy(test_x_uci, test_y_uci, w_uci)

print(ac_uci)


'''

uci 手机三维数据集

'''

X_uci,Y_uci = data_read('phone_final')

train_x_uci, train_y_uci, test_x_uci, test_y_uci = split_data(X_uci, Y_uci, 0.3)

co_uci, w_uci = gra_descent_wyd.DescentGradient(train_x_uci, train_y_uci,lamda=0).gradient_descent()

# print(co_uci)

poly = np.poly1d(co_uci)

%matplotlib notebook

show_3D(train_x_uci,train_y_uci, co_uci)

show_3D(test_x_uci, test_y_uci, co_uci)

ac_uci = cal_accuracy(test_x_uci, test_y_uci, w_uci)

print(ac_uci)
```