Data Mining, Spring 2018

Problem Set #1: Supervised Learning – Regression and SVM

(Due on April 8, 2018 at 11:59pm)

Submission Instructions

These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a write-up in PDF format to DataMining_2018@126.com. The email title is formatted as "hwk1_学号_姓名".

Questions

1. 线性回归

某班主任为了了解本班同学的数学和其他科目考试成绩间关系，在某次阶段性测试中，他在全班学生中随机抽取 1 个容量为 5 的样本进行分析。该样本中 5 位同学的数学和其他科目成绩对应如下表：

学生编号	1	2	3	4	5
数学分数 m	89	91	93	95	97
物理分数 p	87	89	89	92	93
语文分数 c	72	76	74	71	76
英语分数 e	83	88	82	91	89
化学分数 ch	90	93	91	89	94

利用以上数据，建立 m 与其他变量的多元线性回归方程，并回答下列问题：

（1） 在线性回归中，利用梯度下降法，令参数向量$\boldsymbol{\theta}^0$初始值全为$\mathbf{0}$，学习率α为 1，算出经过第一次迭代后的参数向量$\boldsymbol{\theta}^1$；

答：编程实现梯度下降法的线性回归，令参数向量 θ ^0 初始值全为 0，设置学习率 α 为 1，经过第一次迭代后的参数向量$\boldsymbol{\theta}^1$如下所示：即为 93, 1, 0.24, 0.66666667, 0.32。

```
Iteration 0 | Theta: [93.        1.        0.24        0.66666667  0.32        ]
```

梯度下降法关键代码如下：

```python
# features scaling
def featuresNormalization(x):
    x_mean = np.mean(x, axis=0) # 列均值
    x_max = np.max(x, axis=0) # 列最大值
    x_min = np.min(x, axis=0) # 列最小值
    x_s = x_max - x_min
    x = (x - x_mean) / x_s
    return x, x_mean, x_s

# m denotes the number of examples here, not the number of features
def gradientDescent(x, y, theta, alpha, m, numIterations):
    x_T = np.transpose(x)
    for i in range(0, numIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y
        # avg cost function J
        cost = np.sum(loss ** 2) / (2 * m)
        print("Iteration %d | Cost: %f" % (i, cost))
        # avg gradient per example
        gradient = np.dot(x_T, loss) / m
        # update theta
        theta = theta - alpha * gradient
        print("Iteration %d | Theta: %s" % (i, theta))
    return theta
```

利用第（4）小题提供的分数，经历大约 6354 次迭代后，预测值趋于稳定，达到 88.9375：

```
Iteration 6353 | Theta: [93.        10.125      1.87499999 -2.8125     -2.18749998]
Theta:  [93.        10.125      1.87499999 -2.8125     -2.18749998]
Predit result:  [88.9375]
PS D:\linjiafengyang\Code\Python>
```

（2）讨论（1）中所算出的$\boldsymbol{\theta}^1$是否可以使线性回归中的代价函数$J(\boldsymbol{\theta})$下降，即$J(\boldsymbol{\theta}^1) < J(\boldsymbol{\theta}^0)$；

答：（1）中所算出的$\boldsymbol{\theta}^1$可以使线性回归中的代价函数$J(\boldsymbol{\theta})$下降，设置多次迭代，可以得到代价函数下降的结果：

```
Iteration 0 | Cost: 4328.500000
Iteration 0 | Theta: [93.         1.         0.24       0.66666667  0.32      ]
Iteration 1 | Cost: 2.617908
Iteration 1 | Theta: [93.         1.77339259  0.36155259  1.11294815  0.50641778]
Iteration 2 | Cost: 1.881742
Iteration 2 | Theta: [93.         2.38338123  0.41164078  1.40101774  0.60883555]
```

（3）讨论是否可以选取更佳的学习率α，经过第一次迭代后，使代价函数$J(\boldsymbol{\theta})$下降得更快；

答：经过不断尝试学习率α的取值，发现只有在 1.0001、1.00001 以及 1.000001 才可能使得代价函数$J(\boldsymbol{\theta})$下降得更快，其实当学习率为 0.9999999 或者 1.0000001 时，前几次迭代和学习率为 1 时是一样的，这涉及到精度问题，计算时应该会被当成 1，我们姑且不做讨论。所以总的来说，在此题中找到更佳的学习率α使代价函数$J(\boldsymbol{\theta})$下降得更快似乎不太可能，可以认为α为 1 就是最好的学习率。

```
| Cost: 4328.500000
| Theta: [93.
| Cost: 2.617908
| Theta: [93.
| Cost: 1.881742
| Theta: [93.
```

α = 1

```
Cost: 4328.500000
Theta: [93.0093
Cost: 2.617835
Theta: [92.99999907
Cost: 1.881617
Theta: [93.
```

α = 1.0001

```
Cost: 4328.500000
Theta: [93.00093
Cost: 2.617897
Theta: [92.99999999
Cost: 1.881730
Theta: [93.
```

α = 1.00001

（4）利用标准方程求出最优的多元线性回归方程（系数精确到 0.01），并预测该班物理分数 88、语文分数 73、英语分数 87、化学分数 92 同学的数学分数。

答：如下图，此时多元线性回归方程为：$h_{\theta}(x) = -19.5x_0 + 1.6875x_1 + 0.375x_2 - 0.3125x_3 - 0.4375x_4$。因此，预测该同学的数学分数如下：（注意上述公式 x_0 应该代入 1），与（1）作比较可以发现两者的预测值都为 88.9375。

```
Theta:  [-19.5       1.6875    0.375    -0.3125   -0.4375]
Predit result:  88.93749999937934
PS D:\linjiafengyang\Code\Python>
```

标准方程关键代码如下：

```
# theta = (X'X)^(-1)X'Y
# theta = np.dot(np.dot(np.linalg.inv(np.dot(X_T, X)), X_T), Y)
temp1 = np.dot(X_T, X)
temp2 = np.linalg.inv(temp1)
temp3 = np.dot(temp2, X_T)
theta = np.dot(temp3, Y)
print("Theta: ", theta)

x_predit = [1, 88, 73, 87, 92]
print("Predit result: ", np.dot(x_predit, theta))
```

（5）在 L2 正则化线性回归中，令正则化平衡系数λ为 1，利用标准方程求出最优的 L2 正则化多元线性回归方程（系数精确到 0.01），并比较其与（4）中得出的多元线性回归方程对数学分数的预测，哪个更好。

答：如下图，此时多元线性回归方程为：$h_\theta(x) = -19.99x_0 + 1.47x_1 + 0.07x_2 - 0.23x_3 - 0.06x_4$。因此，预测该同学的数学分数如下：（注意上述公式 x_0 应该代入 1），预测值为 89.8733。

```
Theta:  [-19.98847328   1.4734096    0.06935767  -0.22573622  -0.05676401]
Predit result:  89.87334176201837
PS D:\linjiafengyang\Code\Python>
```

L2 正则化关键代码如下：

```
# L2正则化
# theta = (X'X + lamda*matrix)^(-1)X'Y
temp1 = np.dot(X_T, X) + lamda * matrix

temp2 = np.linalg.inv(temp1)
temp3 = np.dot(temp2, X_T)
theta = np.dot(temp3, Y)
print("Theta: ", theta)

x_predit = [1, 88, 73, 87, 92]
print("Predit result: ", np.dot(x_predit, theta))
```

在学习线性回归过程中，我们知道正则化是用来避免过拟合问题的，然而在这道题中，我发现用 scikit-learn 实现线性回归算法，然后利用（4）中数据预测数学分数时，得到的结果恰恰是梯度下降法和无正则化的标准方程得到的结果，如下所示：也是 88.9375，因此我认为在这道题中，并不会出现严重的过拟合问题，因此不需要实现 L2 正则化算法，即可较为准确地预测，也就是说（4）中标准方程对数学分数的预测较好。

```
Predit result:  [88.9375]
PS D:\linjiafengyang\Code\Python>
```

Scikit-learn 关键代码如下：

```
Y = np.array([89, 91, 93, 95, 97])
X = np.array([[87, 72, 83, 90],
              [89, 76, 88, 93],
              [89, 74, 82, 91],
              [92, 71, 91, 89],
              [93, 76, 89, 94]])
m = np.alen(X)
ones = np.ones(m)
X = np.column_stack((ones, X))

model = linear_model.LinearRegression()
model.fit(X, Y)
x_predict = np.array([[1, 88, 73, 87, 92]])
result = model.predict(x_predict)
print("Predit result: ", result)
```

2. 逻辑回归

研究人员对使用雌激素与子宫内膜癌发病间的关系进行了 1:1 配对的病例对照研究。病例与对照按年龄相近、婚姻状况相同、生活的社区相同进行了配对。收集了年龄、雌激素药使用、胆囊病史、高血压和非雌激素药使用的数据。变量定义及具体数据如下：

match：配比组

case：case=1 病例；case=0 对照（未发病）

est：est=1 使用过雌激素；est=0 未使用雌激素；

gall：gall=1 有胆囊病史；gall=0 无胆囊病史；

hyper：hyper=1 有高血压；hyper=0 无高血压；

nonest：nonest=1 使用过非雌激素；nonest=0 未使用过非雌激素；

Match	Case	Est	Gall	Hyper	Nonest
1	1	1	1	0	1
1	0	0	1	0	0
2	1	1	0	1	1
2	0	0	0	0	1
3	1	1	1	0	1
3	0	1	0	1	1
4	1	1	0	0	0
4	0	1	0	1	1
5	1	1	0	1	1
5	0	0	0	0	0
6	1	1	1	0	1
6	0	0	0	0	0
7	1	1	0	0	1
7	0	0	0	0	0
8	1	1	1	1	1
8	0	0	0	1	1
9	1	1	0	0	1
9	0	1	0	0	1
10	1	0	0	0	1
10	0	0	0	0	1
11	1	1	0	1	1
11	0	1	0	1	1
12	1	0	0	0	1
12	0	0	0	1	1
13	1	1	0	1	1
13	0	0	0	0	0
14	1	1	0	0	1
14	0	0	0	0	0
15	1	1	0	1	1
15	0	1	0	0	1
16	1	1	0	0	1
16	0	1	0	1	1
17	1	1	0	0	1
17	0	0	0	0	0
18	1	0	1	0	1
18	0	0	0	1	0
19	1	1	1	0	1
19	0	1	1	0	0
20	1	1	0	0	0
20	0	1	0	1	1

（1） 调用逻辑回归函数或实现求解 L2 逻辑回归分析的梯度下降算法，求出最优的逻辑回归模型；

答：我采用的是无正则化的逻辑回归函数以及调用 scikit-learn.cross_validation 中的 train_test_split 模块把数据集随机分为训练集和测试集，其中测试集占 20%。

关键代码如下：

```python
def sigmoid(x):
    result = 1 / (1 + np.exp(-x))
    return result

# m denotes the number of examples here, not the number of features
def gradientDescent(x, y, theta, alpha, m, numIterations):
    x_T = np.transpose(x)
    y_T = np.transpose(y)
    for i in range(0, numIterations):
        hypothesis = sigmoid(np.dot(x, theta))
        loss = hypothesis - y
        # avg cost function J
        cost = 0 - (np.sum(np.dot(y_T, np.log(hypothesis)) +
                np.dot(1 - y_T, 1 - np.log(hypothesis)))) / m
        print("Iteration %d | Cost: %f" % (i, cost))
        # avg gradient per example
        gradient = np.dot(x_T, loss) / m
        # update theta
        theta = theta - alpha * gradient
        print("Iteration %d | Theta: %s" % (i, theta))
    return theta
```

```python
# 划分为训练集和测试集，测试集占1/5
X, X_test, Y, Y_test = train_test_split(X, Y, test_size=0.2)
```

运行该程序，不断调整迭代次数，得到最优的准确率，在我不断尝试的过程中，只有一次得到 100%的准确率，一般得到的较好的准确率为 87.5%和 75%，如下所示：**图中 Theta 表示该准确率对应的 θ 值，Predict and Y_test 表示预测的结果和测试集对应起来，比较容易观察，最后一个是测试集用于训练模型所得的准确率：**

75%的准确率，此时逻辑回归函数$h(z)=\frac{1}{1+e^{-z}}$，$z=-3x_0+3.89x_1+2.40x_2-2.54x_3+1.68x_4$，θ 并不恒定才能达到此准确率。

```
Theta:  [-3.00954243  3.88892561  2.39765188 -2.543838    1.68098673]
Predict and Y_test:
 [[0. 0.]
 [1. 1.]
 [1. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 1.]]
测试集准确率: 75.000000%
PS D:\linjiafengyang\Code\Python>
```

87.5%的准确率，此时逻辑回归函数$h(z)=\frac{1}{1+e^{-z}}$，$z=-7.76x_0+8.35x_1+0.34x_2-8.53x_3+8.10x_4$，θ 并不恒定才能达到此准确率。

```
Theta:  [-7.76369554  8.34692615  0.33984214 -8.53324279  8.10224873]
Predict and Y_test:
 [[1. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
测试集准确率: 87.500000%
PS D:\linjiafengyang\Code\Python>
```

因此，最优的逻辑回归模型可认为是$h(z)=\frac{1}{1+e^{-z}}$，$z=-7.76x_0+8.35x_1+0.34x_2-8.53x_3+8.10x_4$。

（2）尝试找出对影响子宫内膜癌发病的最直接的因素；

答：在我不断尝试的过程中，发现准确率高时，有这么几个特点：

① **θ_2值即 EST 雌激素和 θ_4值即 Nonest 非雌激素这两个数都比较大，且 EST 比 Nonest 的系数要大，说明这两个因素对子宫内膜癌发病有比较严重的影响。**

② **θ_3值即 Gall 胆囊病史在这种情况下一般为非负数，说明也有一定的影响。**

总的来说，影响子宫内膜癌发病的最直接因素，我认为是 EST 雌激素的使用。

下面第（3）小题也支持上述结论。

（3）编程实现求解 L2 正则化逻辑回归分析的梯度下降算法，并求出正则化平衡系数λ为 1 时的最优正则化

逻辑回归模型（加分题）。

答：与（1）同理，此时正则化代价函数，采用逻辑回归函数以及调用 scikit-learn.cross_validation 中的 train_test_split 模块把数据集随机分为训练集和测试集，其中测试集占 20%。

关键代码如下：

```python
def sigmoid(x):
    result = 1 / (1 + np.exp(-x))
    return result

# m denotes the number of examples here, not the number of features
def gradientDescent(x, y, theta, alpha, lamda, m, numIterations):
    x_T = np.transpose(x)
    y_T = np.transpose(y)
    for i in range(0, numIterations):
        hypothesis = sigmoid(np.dot(x, theta))
        loss = hypothesis - y
        # avg cost function J
        cost = 0 - (np.sum(np.dot(y_T, np.log(hypothesis)) + np.dot(1 - y_T, 1 - np.log(hypothesis)))
        + lamda * np.sum(np.dot(theta.T, theta)) / (2 * m)
        print("Iteration %d | Cost: %f" % (i, cost))
        # avg gradient per example
        gradient = np.dot(x_T, loss) / m
        # update theta
        theta = theta - alpha * (gradient + lamda * theta / m)
        print("Iteration %d | Theta: %s" % (i, theta))
    return theta
```

```python
# 划分为训练集和测试集，测试集占1/5
X, X_test, Y, Y_test = train_test_split(X, Y, test_size=0.2)

theta = np.zeros(n)
alpha = 1
lamda = 1
```

运行该程序，不断调整迭代次数，得到最优的准确率，一般得到的较好的准确率为 87.5%和 75%，如下所示：

75%的准确率，此时逻辑回归函数$h(z)=\frac{1}{1+e^{-z}}$，$z=-1.03x_0+1.14x_1+0.46x_2-0.73x_3+0.74x_4$，θ 并不恒定才能达到此准确率。

```
Theta:  [-1.03281935  1.1351348   0.46226623 -0.7339268   0.73808971]
Predict and Y_test:
 [[1. 1.]
 [0. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 0.]
 [1. 1.]
 [1. 1.]]
测试集准确率: 75.000000%
PS D:\linjiafengyang\Code\Python>
```

87.5%的准确率，此时逻辑回归函数$h(z)=\frac{1}{1+e^{-z}}$，$z=-0.82x_0+1.03x_1+0.39x_2-0.75x_3+0.72x_4$，θ 并不恒定才能达到此准确率。

```
Theta:  [-0.82050222  1.03129187  0.38778328 -0.75356117  0.71757723]
Predict and Y_test:
 [[0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]]
测试集准确率: 87.500000%
PS D:\linjiafengyang\Code\Python>
```

因此，最优的逻辑回归模型可认为是$\mathbf{h(z)} = \frac{1}{1+e^{-z}}$， $\mathbf{z} = -0.82x_0 + 1.03x_1 + 0.39x_2 - 0.75x_3 + 0.72x_4$，同样可以发现，该模型依然可验证（2）中的结论。

PS：这里我还使用 **scikit-learn** 实现逻辑回归，同样把数据集分成训练集和测试集，测试集占 **20%**。
关键代码如下：

```python
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
import numpy as np
```

```python
# 划分为训练集和测试集
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

# 逻辑回归
model = LogisticRegression()
model.fit(X_train, Y_train)
print("Theta: ", model.coef_)

# 预测
predict = model.predict(X_test)
right = sum(predict == Y_test)
# 将预测值和真实值放在一块，便于观察
predict = np.hstack((predict.reshape(-1, 1), Y_test.reshape(-1, 1)))
print("Predict and Y_test: \n", predict)
# 计算在测试集上的准确度
print('测试集准确率：%f%%' % (right*100.0 / predict.shape[0]))
```

运行该程序，不断调整迭代次数，得到最优的准确率，一般得到的较好的准确率为 **87.5%**和 **75%**，只有一次拿到了 **100%**的准确率，如下所示：同样可以发现，用 **sklearn** 得到的结果：θ_2值即 **EST** 雌激素和θ_4值即 **Nonest** 非雌激素这两个数都比较大，且 **EST** 比 **Nonest** 的系数要大，因此可验证（2）中的结论：雌激素的使用应该是导致发病的最直接因素。

```
Theta:  [[-0.60686501  1.14268859  0.37219659 -0.62965556  0.97336385]]
Predict and Y_test:
 [[0 0]
 [1 1]
 [0 0]
 [1 1]
 [1 0]
 [0 1]
 [1 1]
 [1 1]]
测试集准确率：75.000000%
PS D:\linjiafengyang\Code\Python>
```

```
Theta:  [[-0.44848425  1.06318014  0.20006868 -0.72020305  0.88167477]]
Predict and Y_test:
 [[1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 0]
 [0 0]
 [1 1]]
测试集准确率：87.500000%
PS D:\linjiafengyang\Code\Python>
```

```
Theta:  [[-0.47765436  0.94531117  0.53529884 -0.69611927  0.81254548]]
Predict and Y_test:
 [[1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]]
测试集准确率: 100.000000%
PS D:\linjiafengyang\Code\Python>
```

3. 支持向量机

考虑以下的两类训练样本集

特征 1	特征 2	类标
1	1	+
2	2	+
2	0	+
0	0	−
1	0	−
0	1	−

（1） 在图中画出这 6 个训练样本点和支持向量机对应的最优超平面(决策边界),并写出对应的超平面方程;

答：编程实现核函数为线性函数的支持向量机算法，得到如下图的结果，以及相对应的 θ 参数值如下：

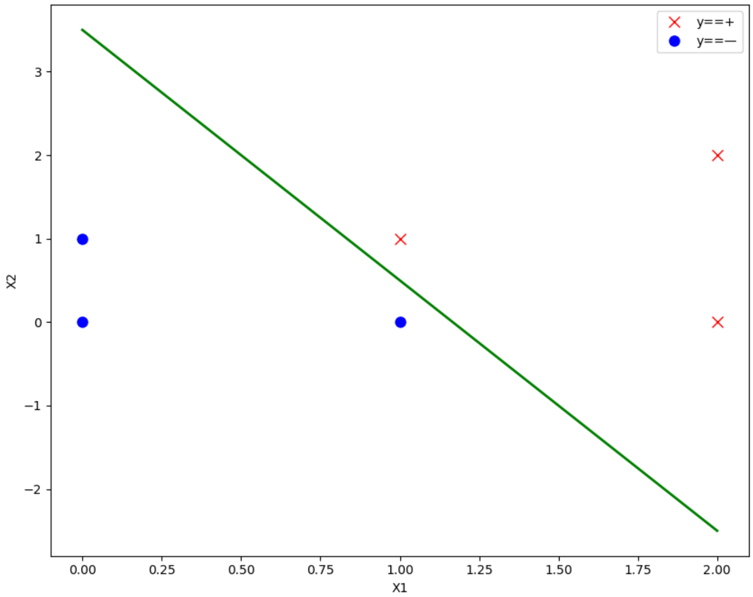
```
PS D:\linjiafengyang\Code\Python> & C:/Anaconda3/python.exe d:/linjiafengyang/Code/
Theta 1 and theta 2:  [[1.2 0.4]]
Theta 0:  [-1.4]
k:  -3.0
b:  3.5
最优超平面（决策边界）的方程: y = -3x + 3.5
PS D:\linjiafengyang\Code\Python>
```

由$\boldsymbol{h_\theta}(\boldsymbol{x}) = \boldsymbol{\theta^T x} = \boldsymbol{\theta_0 + \theta_1 x_1 + \theta_2 x_2}$，此时$\boldsymbol{\theta_0 = -1.4}$，$\boldsymbol{\theta_1 = 1.2}$，$\boldsymbol{\theta_2 = 0.4}$，由 y==1 即$\boldsymbol{h_\theta}(\boldsymbol{x}) = \boldsymbol{\theta^T x \geq 0}$，因此最优超平面（决策边界）方程为：

$$\mathbf{x2} = -\frac{\boldsymbol{\theta_0 + \theta_1 x1}}{\boldsymbol{\theta_2}} = -\frac{\mathbf{-1.4 + 1.2}\boldsymbol{x}\mathbf{1}}{\mathbf{0.4}} = \mathbf{-3}\boldsymbol{x}\mathbf{1 + 3.5}$$

即 k=-3.0，b=3.5.

图表如下所示：



（2） 假设增加一些训练样本点，这些点能被正确分类且远离最优超平面（决策边界），说明最优超平面（决策边界）不受新增训练样本点影响，而线性回归会受影响的原因；

答：支持向量机考虑的是局部的点，也就是和分类最相关的少数点，从而得到决策边界，因而对新增训练样本不敏感，仍能准确地分类，不会轻易改变决策边界；而线性回归考虑全局的点，当新增样本出现异常点时，此时很难去拟合该点，从而对线性回归本身所得假设函数造成影响。

（3） 指出哪些是支持向量，并求出两个异类支持向量到最优超平面（决策边界）的距离之和；

答：支持向量是满足$3x1 + x2 - 3.5 = 1$和$3x1 + x2 - 3.5 = -1$的所有训练样本点，比如(1，1.5)和(1，-0.5)都是支持向量，前者对应 y==1，后者对应 y==0；

两个异类支持向量到最优超平面（决策边界）的距离之和为$\frac{2}{\sqrt{9+1}} = 0.6325$。

（4） 通过寻找拉格朗日待定乘数α_i来构造对偶空间的解，并将其与（1）中结果作比较。

答：对偶问题为：

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$s.t. \sum_{i=1}^{m} \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1, \cdots, m$$

解出 α 后，可得到模型：

$$\begin{aligned} f(x) &= \omega^T x + b \\ &= \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b \end{aligned}$$

上述过程满足 KKT 条件，即要求：

$$
\begin{cases}
\alpha_i \geq 0 \\
y_i f(x_i) - 1 \geq 0 \\
\alpha_i (y_i f(x_i) - 1) = 0
\end{cases}
$$

然后利用 SMO 算法实现求解过程，对上述原理进行编程实现，不断调整松弛变量 C，从而得到下面的结果：Python 实现结果：

```
[[-1.4]]
[[1.2 0.4]]
k: -3.0
b: 3.5
最优超平面（决策边界）的方程: y = -3.0x + 3.5
PS D:\linjiafengyang\Code\Python>
```

代码参考了博客 https://blog.csdn.net/willbkimps/article/details/54697698，该博客给出了 SMO 算法实现关键代码，在上面所求最优超平面方程的情况下，松弛变量 C 为 1，我修改了一下成功解得拉格朗日待定乘子 α ᵢ 如下：分别为 1/0/0.6/0/1/0.6：

```
[[1. ]
 [0. ]
 [0.6]
 [0. ]
 [1. ]
 [0.6]]
```

然后利用该公式：

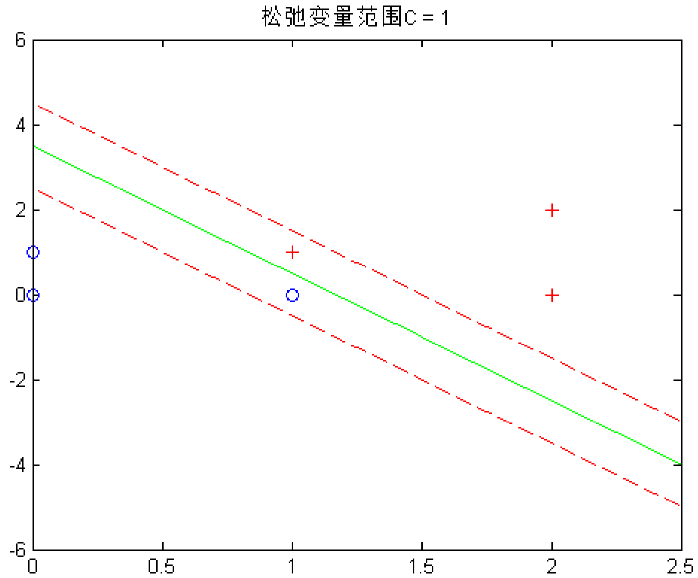
$$
\begin{aligned}
f(x) &= \omega^T x + b \\
&= \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b
\end{aligned}
$$

求解，求解代码实现如下：即可求得最优超平面方程如上所述：y = -3.0x + 3.5。

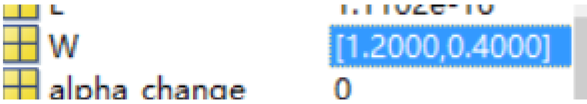
```
alphas = np.array(alphas)
labelMat = np.array(labelMat)

theta12 = (alphas.T*labelMat).dot(dataMat)
print(theta12)
theta0 = theta12[0,0]
theta1 = theta12[0,1]
k = -theta0 / theta1
b = -b / theta1
```

Matlab 实现结果：松弛变量 C 为 1，绿色线为最优超平面，两条红色线均为支持向量：

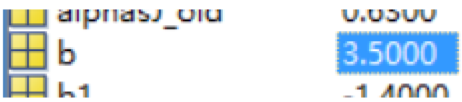


对应 **W** 值：**[1.2, 0.4]**

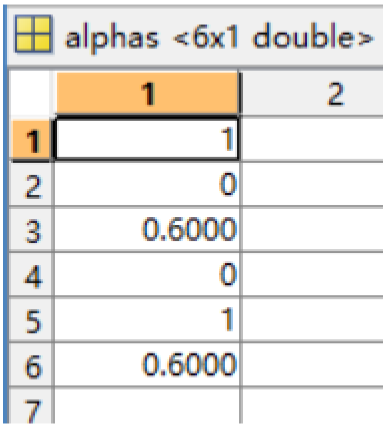


对应 **k** 值：**-3**

对应 **b** 值：**3.5**



对应拉格朗日待定乘子 α_i 如下：分别为 **1/0/0.6/0/1/0.6**：


alphas <6x1 double>

	1	2
1	1	
2	0	
3	0.6000	
4	0	
5	1	
6	0.6000	
7		

Matlab 实现 **SMO** 算法参考博客 https://blog.csdn.net/on2way/article/details/47730367，博客中代码有错误，算法得到的 **b** 值忘记除以 **W** 的第二个值，即原 **b** 值为**-1.4**，需进行如下处理：**-(-1.4) / 0.4=3.5.**

综上所述，所得到的最优超平面（决策边界）方程：

$$x2 = -3x1 + 3.5$$

是能够较好地实现分类效果的。

作业思考

1. **线性回归与逻辑回归的区别：**
   线性回归主要用来**解决连续值预测**的问题，逻辑回归用来**解决分类**的问题，输出的属于某个类别的概率。
2. **逻辑回归与支持向量机的区别：**
   两种方法都是常见的**分类算法**，两者的根本目的都是一样的。
   目标函数：逻辑回归采用的是 logistical loss，svm 采用的是 hinge loss。这两个损失函数的目的都是增加对分类影响较大的数据点的权重，减少与分类关系较小的数据点的权重。
   训练样本点：SVM 的处理方法是只考虑 support vectors，也就是和分类最相关的少数点，去学习分类器。而逻辑回归通过非线性映射，大大减小了离分类平面较远的点的权重，相对提升了与分类最相关的数据点的权重。
   简单性：逻辑回归相对来说模型更简单，容易实现，特别是大规模线性分类时比较方便。而 SVM 的理解和优化相对来说复杂一些。但是 SVM 的理论基础更加牢固，有一套结构化风险最小化的理论基础，虽然一般使用的人不太会去关注。还有很重要的一点，SVM 转化为对偶问题后，分类只需要计算与少数几个支持向量的距离，这个在进行复杂核函数计算时优势很明显，能够大大简化模型和计算量。