凸优化大作业

杨跃浙组
组长：WA2214014 杨跃浙
成员：WA2214030 蔡文杰 WA2214022 杨昀川 WA2214026 王振宇 WA2214006 王家浩
代码编写与注释：杨跃浙
代码运行： 杨跃浙 蔡文杰 杨昀川
报告编辑： 杨跃浙 蔡文杰 王振宇
资料查找： 杨跃浙 蔡文杰 王家浩

一、数据收集与预处理

1、数据来源：https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

covtype.libsvm.binary.scale.bz2 (scaled to [0,1])

2、数据预处理：

采用切片方式获得标签和数据，将数据转为字典的形式获得特征向量。通过求字典的键的和来确定是否有特征缺失，对于有特征缺失的数据进行了舍去处理（因为所给数据集的量较大，故直接将特征缺失数据舍去，以免造成干扰）。最终将数据矩阵（张量）化。此数据已按比例缩放至(0,1)在 lasso 回归中表现较好，在逻辑回归中对数据进行了打乱，但是数据集没有表现离散分布，导致分类算法效果不佳。

二、问题

1）问题一：Lasso 回归问题

0、实验原理：

Lasso 回归是在普通最小二乘法的基础上，引入 L1 正则化项，通过最小化目标函数来实现模型的特征选择和系数稀疏化。

Lasso 回归的目标函数如下所示:minimize ‖Y - xβ‖^2 + λ‖β‖₁，其中，Y 是观测值向量，X 是特征知阵，β 是待估计的回归系数向量，λ 是控制正则化强度的超参数

L1 正则化项 λ‖β‖₁ 在目标函数中起到了关键作用。它引入了稀疏性，即使得一些特征的系数被压缩为零，从而实现了自动的特征选择。因此，Lasso 回归不仅可以进行预测，还可以识别出对目标变量有重要影响的特征。

Lasso 回归的损失函数正则化项为惩罚系数 λ 乘以 w 向量的 L1-范数的平方。

损失函数

$$\mathrm{Cost}(w) = \sum_{i=1}^{N} \left(y_i - w^T x_i\right)^2 + \lambda \|w\|_1$$

坐标下降法

坐标下降法（Coordinate Descent）是一种优化算法，用于求解无约束优化问题。它适用于目标函数可分解为各个变量的子问题的情况，即目标函数可以表示为各个变量的函数的和。

在第 k 次迭代时，更新权重系数的方法如下：

$$w_m^k \text{表示第} k \text{次迭代，第} m \text{个权重系数}$$

$$w_1^k = \underset{w1}{\operatorname{argmin}} \left(\mathrm{Cost}(w_1, w_2^{k-1}, \ldots, w_{m-1}^{k-1}, w_m^{k-1})\right)$$

$$w_2^k = \underset{w_2}{\operatorname{argmin}} \left(\mathrm{Cost}(w_1^k, w_2, \ldots, w_{m-1}^{k-1}, w_m^{k-1})\right)$$

$$\vdots$$

$$w_m^k = \underset{w_m}{\operatorname{argmin}} \left(\mathrm{Cost}(w_1^k, w_2^k, \ldots, w_{m-1}^k, w_m)\right)$$

1、数据导入

X 为特征向量集合，y 为对应的标签集合。

```
def read_file():
    """
    Read the dataset and process it as X(Training Dataset) and y (Target label value)

    """
    with open('./covtype.libsvm.binary.txt', 'r', encoding='utf-8') as f1:
        for num, line in enumerate(f1.readlines()):
            Y=int(line[0:1:1])
```

```
            line=line[2::]
            line='{'+line[:-25:]+'}'
            line=re.sub(' ',',',line)
            #print(line) (test)
            dic_data=eval(line)
            #Determine if all ten dimensions have values
            if sum(dic_data.keys())==55:
                X.append(list(dic_data.values()))
                y.append(Y)
```

2、求解 x*（采用 sklearn 库内置函数，因为 cvxpy 没有得出结果没弄明白）

```
def getx(X,y):
    """
    Find x*,through external libraries scikit-learn
    :param X: Training Dataset
    :param y: Target label value
    """
    from sklearn.linear_model import Lasso
    # Initialize Lasso regression, default to using coordinate descent method
    lasso = Lasso(alpha=0.1,fit_intercept=False)
    # Fitting linear models
    lasso.fit(X, y)
    global    theory_V
    # Weight coefficient
    theory_V = lasso.coef_
```

3、lasso 回归求解

1）用坐标下降法迭代实现 lasso 回归问题

matrix_minus 函数用于两矩阵相减即 x*-xk

```
def lassoUseCd(X, y, lambdas=0.1, max_iter=1000, tol=1e-4):
    """
    Lasso regression, using coordinate descent method
    :param X: Training Dataset (a ten dimensional vector)
    :param y: Target label value (1 or 2, binary classification)
    :param lambdas: Penalty term coefficient (Default to 0.1)
    :param max_iter: Maximum number of iterations (Default to 1000)
    :param tol: Tolerance value for variation (Default to 0.0001)

    :return: W: Weight coefficient
    """
    # Global variable declaration
    global theory_V
    global datas
    global itera

    # Initialize W as a zero vector
    W = np.zeros(X.shape[1])
    for it in range(max_iter):
        done = True
        # Traverse all independent variables
        for i in range(0, len(W)):
            # Record the previous round of coefficients
            w = W[i]
            # Find the optimal coefficient under current conditions
            W[i] = down(X, y, W, i, lambdas)
            # Continue cycling if one of the coefficient changes does not reach its tolerance value
            if (np.abs(w - W[i]) > tol):
                done = False
```

```
            # All coefficients do not change much, end the cycle
            if (done):
                break
            norm=np.linalg.norm(np.abs(matrix_minus(W,theory_V)))
            print(it+1,'\t',norm)
            itera.append(it+1)
            datas.append(norm)
        return W
```

2）求解代价函数梯度

```
def down(X, y, W, index, lambdas=0.1):
    """
    Find the optimal coefficient
    cost(W) = (x1 * w1 + x2 * w2 + ... - y)^2 / 2n + ... + λ(|w1| + |w2| + ...)
    Assuming w1 is a variable and all other values are constants, the cost function of w1 is a quadratic function of one variable, which can be written as follows:
    cost(w1) = (a * w1 + b)^2 / 2n + ... + λ|w1| + c (a, b, c, λ   All are constants)
    =>Unfolding
    cost(w1) = aa / 2n * w1^2 + (ab / n) * w1 + λ|w1| + c (aa, ab, c, λ All are constants)

    :param X: Training Dataset (a ten dimensional vector)
    :param y: Target label value (1 or 2, binary classification)
    :param W: Weight coefficient
    :param index: Index of w
    :param lambdas: Penalty term coefficient (Default to 0.1)

    :return: w: The optimal coefficient under current conditions
    """

    # The sum of coefficients of the expanded second-order term
    aa = 0
    # The sum of coefficients of the expanded first-order term
    ab = 0
    for i in range(X.shape[0]):
        # The coefficient of the first_order term in parentheses
        a = X[i][index]
        # Coefficients of constant terms in parentheses
        b = X[i][:].dot(W) - a * W[index] - y[i]
        # The coefficient of the expanded second-order term is the sum of the squares of the coefficients of the first-order term in parentheses
        aa = aa + a * a
        # The coefficient of the expanded first-order term is obtained by multiplying the coefficient of the first-order term in parentheses by the sum of the constant terms in parentheses
        ab = ab + a * b
    # As it is a univariate quadratic function, when the derivative is zero, the function value is the minimum value, and only the second-order coefficient, first-order coefficient, and constant λ
    return det(aa, ab, X.shape[0], lambdas)
```

3）通过代价函数的导数求各个维度的权重即 xk

```
def det(aa, ab, n, lambdas=0.1):
    """
    Calculate w through the derivative of the cost function, and when w=0, it is not differentiable
    det(w) = (aa / n) * w + ab / n + λ = 0 (w > 0)
    => w = - (ab / n + λ) / (aa / n)

    det(w) = (aa / n) * w + ab / n - λ = 0 (w < 0)
    => w = - (ab / n - λ) / (aa / n)

    det(w) = NaN (w = 0)
    => w = 0

    :param aa: Sum of coefficients of second-order terms
    :param ab: Sum of coefficients of the expanded first-order term
    :param n: Dimension of vectors
    :param lambdas: Penalty term coefficient (Default to 0.1)
```

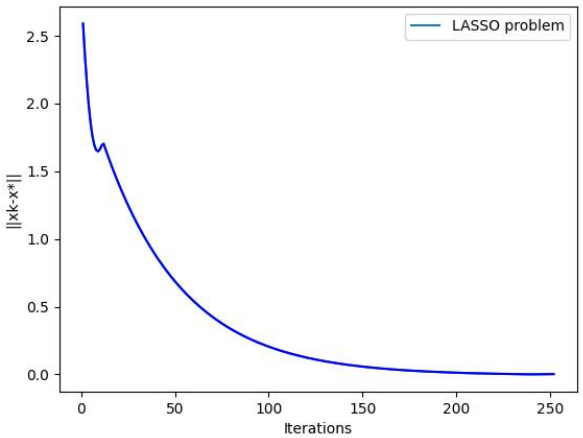
```
    :return: w:The optimal coefficient under current conditions
    """
    w = - (ab / n + lambdas) / (aa / n)
    if w < 0:
        w = - (ab / n - lambdas) / (aa / n)
        if w > 0:
            w = 0
    return w
```

3、 实验结果

输出收敛曲线

```
def output(row,column):
    """
    Output Convergence Graph
    :param row: Data of x
    :param column: Data of y
    """
    import matplotlib.pyplot as plt
    plt.figure()
    plt.plot(row,column,label='LASSO problem')
    plt.plot(row,column,'b-')
    plt.xlabel('Iterations')
    plt.ylabel('||xk-x*||')
    plt.legend()
    plt.savefig('LASSO.png')
```

得到当 λ 取 0.1 时输出的 lasso 回归收敛曲线（随迭代次数增加）



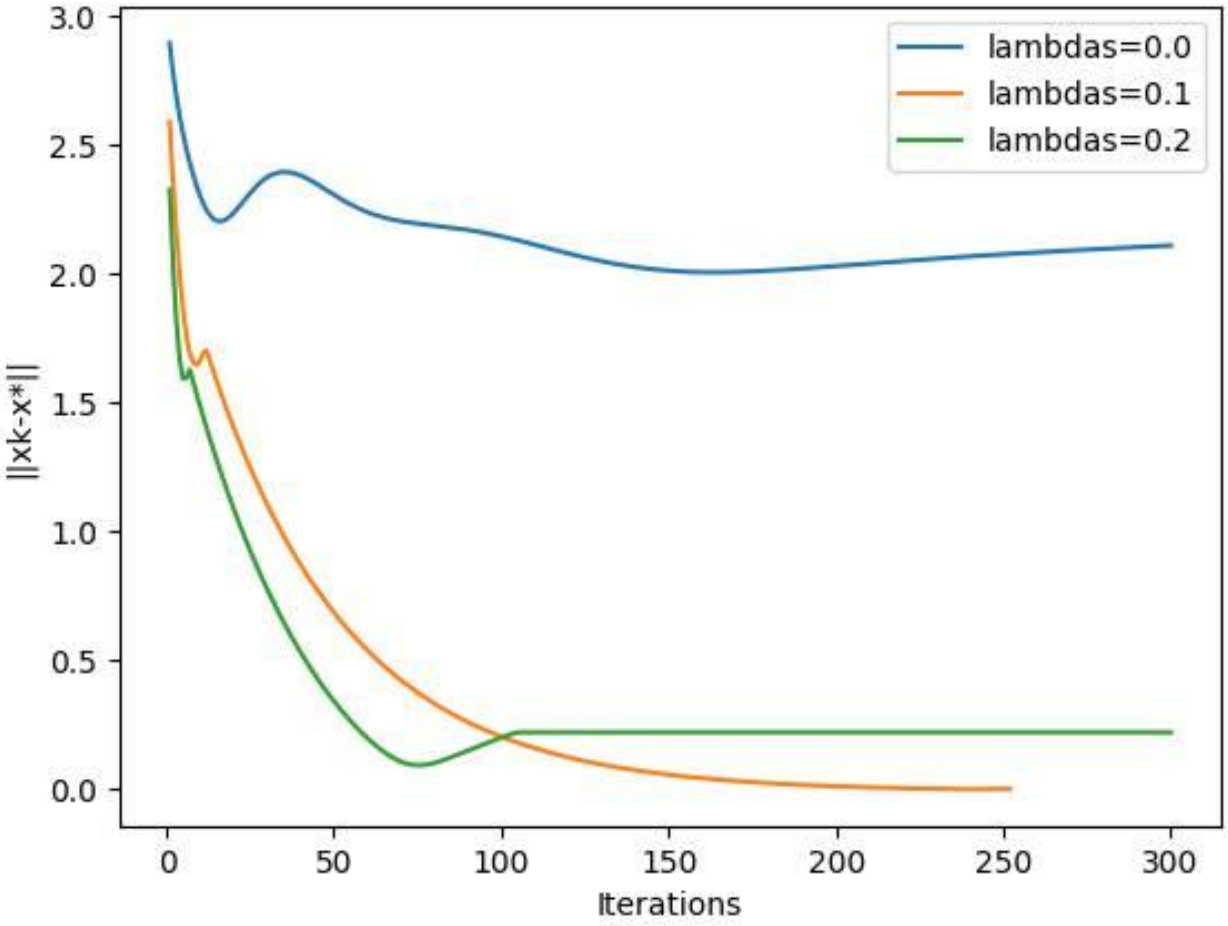
CPU 运行时间：1413.4s（迭代次数 252 次）

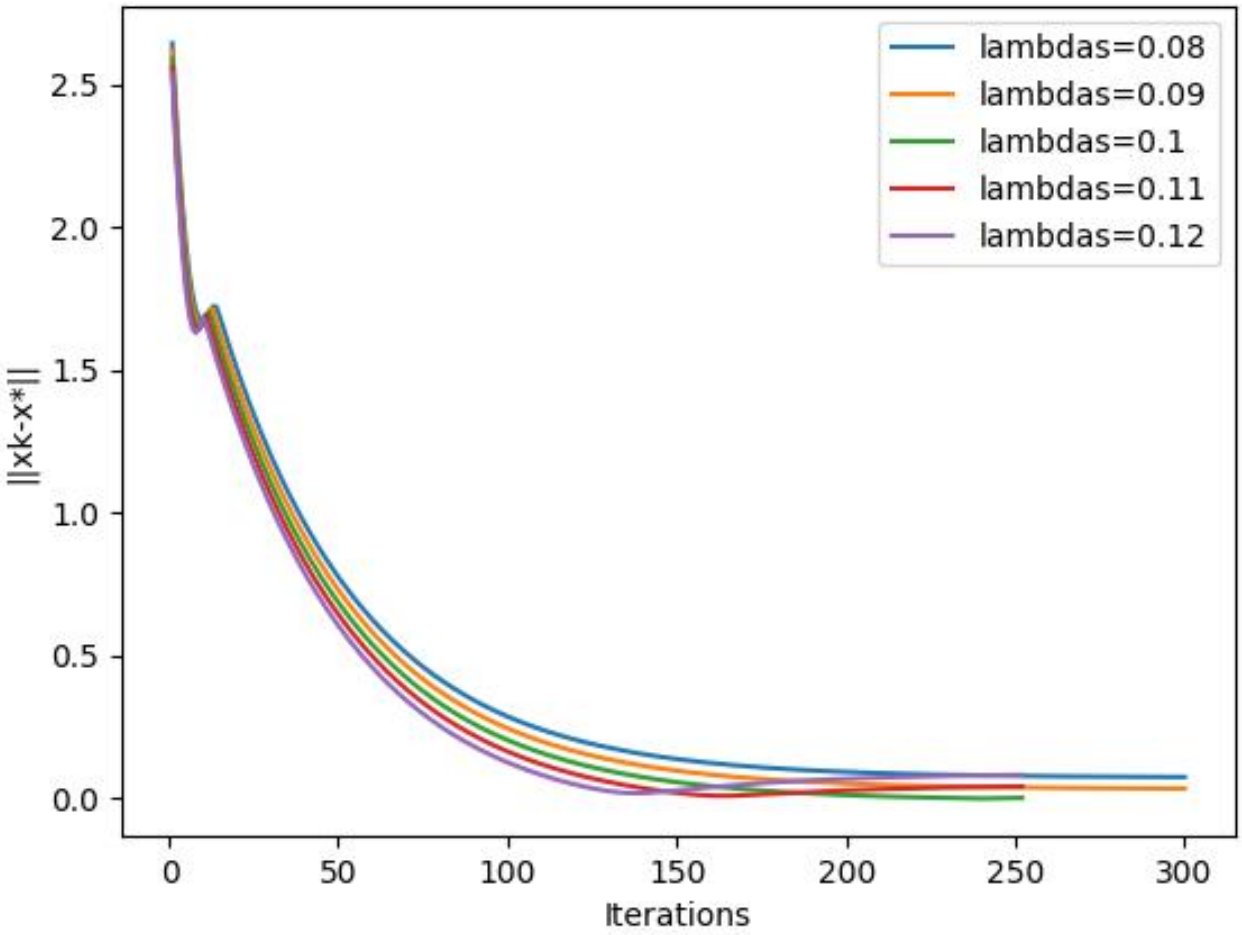
```
228      0.002530558863604031
229      0.0022911148328024
230      0.002057280044220861
231      0.0018289230953621933
232      0.0016059156616336954
233      0.0013881324249783247
234      0.0011754510025428072
235      0.0009677518790019027
236      0.0007649183379136414
237      0.0005668363976444995
238      0.0003733947469444527
239      0.0001844846814873212
240      4.435719276004758e-11
241      0.0001801628350346949
242      0.00035610519914618224
243      0.0005279259177939782
244      0.0006957215451148961
245      0.000859586373590251
246      0.0010196124864762567
247      0.001175898099760954
248      0.001328506163432380 8
249      0.0014775473094374909
250      0.001623097000968715
251      0.001765237029251969
252      0.0019040472696276118
Running time: 1413.4150869846344 Seconds

进程已结束,退出代码0
```

之后采用了没有钳制在 0-1 范围内的数据，收敛曲线与此图相似

之后我们考虑了不同的λ对收敛速度以及收敛效果的影响，发现λ越大收敛速度越快，但同时可能导致收敛效果不佳。
输出了 λ 在 0。0-0.2 范围内的收敛曲线以及 0.1 附近的收敛曲线（最大迭代次数为 300 次）


lambdas=0.0
lambdas=0.1
lambdas=0.2
||xk-x*||
Iterations
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250
300



考虑 λ 的最优值，通过查询资料发现，scikit-learn 通过交叉验证来公开设置 Lasso 中 alpha 参数的对象: LassoCV 和 LassoLarsCV。LassoLarsCV 是基于最小角回归的算法。

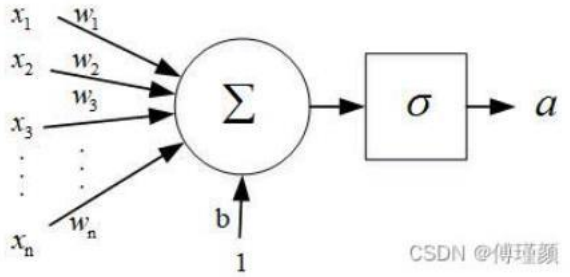
对于带有很多共线回归器 (collinearity) 的高维数据集，LassoCV 是经常被选择的模型。然而，LassoLarsCV 在寻找更有相关性的 alpha 参数值上更有优势，而且如果样本数量与特征数量相比非常小时，通常 LassoLarsCV 比 LassoCV 要快。

2）问题二：逻辑（对数几率）回归问题

0、实验原理:

（1）谈到回归问题，第一反应是：y=kx+b 在二维平面上是一条直线。当 k 和 b 确定时，对于回归问题，假设为 x 面积，经过线性映射，可以得到其体积 y，则完成回归任务；对于分类问题，假设 x 为某个特征，经过线性映射，得到 y>0，或<0，或=0，若规定大于 0 的为正标签，小于等于 0 的为负标签，则完成了分类任务。

同理可得，当方程为多元方程时 $f(x)=w_1x_1+w_2x_2+w_3x_3+...+w_nx_{1n}+b=\vec{w}\cdot\vec{x}+b=w^Tx$，如下图所示：



sigmoid 函数

输入数据，经过函数映射为，该函数为 **sigmoid** 函数，形式为 $sigmoid=\frac{1}{1+e^{-z}}$

（2）输入和输出形式

输入：$x=(x_1\ x_2\ x_3\ ...x_n\ 1)$

输出：

$a=\frac{1}{1+e^{-\left(w_1x_1+w_2x_2+w_3x_3+...+w_nx_n+b\right)}}=\frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$，其中 $\sigma(z)=\frac{1}{1+e^{-z}}$

这里和如图所示，分别为输入数据和待求参数，**b** 为偏置项，为了后续推导方便，设定：$\vec{w}\cdot\vec{x}+b=w^Tx$，即

$w=(w_1\ w_2\ w_3\ ...\ \ w_n\ b)$，$x=(x_1\ x_2\ x_3\ ...x_n\ 1)$。

输出值就是概率值，对中参数的求导过程如下所示，后面会用到，先求出来放在这里哈：$a'=(\frac{1}{1+e^{-(w^Tx)}})'=\frac{e^{-w^Tx}\cdot x^T}{(1+e^{-w^Tx})^2}=a(1-a)x^T$

求导过程为除法求导运算法则，需要注意一个推导公式：$(w^Tx)'=\frac{\partial w^Tx}{\partial w}=x^T$。

（3）基于目标函数求解参数 w

极大似然估计提供了一种基于给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”。简单说来，就是知道了模型和结果，求解使得

事件结果以最大概率发生时出现的参数。

基于逻辑回归的计算式，对应标签 1 和 0 的概率分别为：

$$P(Y=1|X)=\frac{1}{1+e^{-(w^T x)}}=p(x)$$

$$P(Y=0|X)=\frac{1}{1+e^{-(w^T x)}}=1-p(x)$$

第一步，构造极大似然函数，计算这些样本的似然函数，其实就是把每个样本的概率乘起来：

$$L(w,x)=\prod_{i=1}^{n}P(y_i|x_i)=\prod_{i=1}^{n}p^{y_i}(1-p)^{(1-y_i)}$$

第二步，两边取对数得：

$$ln\prod_{i=1}^{n}p^{y_i}(1-p)^{(1-y_i)}=\sum_{i=1}^{n}y_i lnp+\sum_{i=1}^{n}(1-y_i)ln(1-p)$$

tips：由于极大似然函数中有连乘符号，取对数，将连乘变为加和。

目标函数为：$L(x,w)=\sum_{i=1}^{n}y_i lnp+\sum_{i=1}^{n}(1-y_i)ln(1-p)$

其中，y 为真值，p 为预测值。

原函数求最大值，等价于乘以负 1 后求最小值。对于 n 个数据累加后值较大，用梯度下降容易导致梯度爆炸，可处于样本总数 n，即

$$L(x,w)=-\frac{1}{n}(\sum_{i=1}^{n}y_i lnp+\sum_{i=1}^{n}(1-y_i)ln(1-p))$$

第三步，对目标函数中参数 w 求导：

$$\frac{\partial L}{\partial w}=-\frac{1}{n}(\frac{y_i}{p}p'-\frac{1-y_i}{1-p}p')=-\frac{1}{n}(\frac{y_i}{p}-\frac{1-y_i}{1-p})p(1-p)x^T$$

$$=-\frac{1}{n}(y_i(1-p)x^T-(1-y_i)px^T)=-\frac{1}{n}(y_i-p)x^T$$

（为了求导过程更清晰，先去掉求和符号）。

tips：$p(x)'=p(x)(1-p(x))x^T$，该求导过程，涉及到对概率值 p 的求导，这个求导过程在前面已经推导完成。

添加求和符号后为：$\frac{\partial L}{\partial w}=-\frac{1}{n}\sum_{i=n}^{n}(y_i-p)x^T$

基于梯度下降法求得最优 w:$w_{t+1}=w_t+\frac{1}{n}\sum_{i=n}^{n}(y_i-p)x^T$

1、数据导入

在 lasso 问题处理数据的基础上增加了对数据的打乱（在测试过程中发现数据的标签呈集中分布，对处理回归和分类产生了影响）

将数据集分为训练集与测试集用于实现分类问题，训练集与测试集的划分方式为：训练集为打乱后数据集的前 5000 项，测试集为打乱后数据集的后 100 项（数据集超 20000 项）

X_train:训练集特征向量集合，y_train 训练集标签集合

X_test 测试集特征向量集合，y_test 测试集标签集合

```
def load_dataset():
    X=[]
    y=[]
    with open('./covtype.libsvm.binary.scale.txt', 'r', encoding='utf-8') as f1:
        for num, line in enumerate(f1.readlines()):
            Y=int(line[0:1:1])-1
            line=line[2::]
            line='{'+line[:-12:]+'}'
            line=re.sub(' ',',',line)
            #print(line) (test)
            dic_data=eval(line)
            #Determine if all ten dimensions have values
            if sum(dic_data.keys())==55:
                X.append(list(dic_data.values()))
                y.append(Y)
    import random
    random.seed()
    index = [i for i in range(len(X))]
    random.shuffle(index)
    X1=[]
    y1=[]
    for i in index:
        X1.append(X[i])
        y1.append (y[i])
    X=X1
    y=y1
    X=np.array(X)
    #X=X[:,8:10]
    #print(X)
    y=np.array(y)
    train_x, train_y = X[0:5000], y[0:5000]
    test_x, test_y = X[len(X)-100:], y[len(y)-100:]
    return train_x, train_y, test_x, test_y
```

2、用 scikit-learn 的内置求解器解决逻辑回归问题（“l2”正则化）

```
def getx(X,y):
    from sklearn.linear_model import LogisticRegression
    reg = LogisticRegression(penalty="l2")
    reg.fit(X, y)
    global    theory_Vw,theory_Vb
    theory_Vw = reg.coef_
    theory_Vb=float(reg.intercept_)
```

3、解决逻辑回归问题

其中 loss 函数（损失函数）计算出现问题，可能是由于数据被钳制在[0,1]范围内较小造成的,在使用另一个未被按比例缩放至[0,1]的数据集时，损失函数部分正确，呈收敛趋势，剩余问题仍没有解决

```
class LogisticRegression:
    def __init__(self, learning_rate=0.003, iterations=100):
        self.learning_rate = learning_rate
        self.iterations = iterations

    def fit(self, X, y):

        self.weights = np.random.randn(X.shape[1])
        self.bias = 0

        for i in range(self.iterations):
            y_hat = sigmoid(np.dot(X, self.weights) + self.bias)

            #loss = (-1 / len(X)) * np.sum(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))

            dw = (1 / len(X)) * np.dot(X.T, (y_hat - y))
            db = (1 / len(X)) * np.sum(y_hat - y)


            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        y_hat = sigmoid(np.dot(X, self.weights) + self.bias)
        y_hat[y_hat >= 0.5] = 1
        y_hat[y_hat < 0.5] = 0
        return y_hat

    def score(self, y_pred, y):
        accuracy = (y_pred == y).sum() / len(y)
        return accuracy
```

4、实验结果

实验结果发现 sklearn 内部的求解器求得的权重矩阵和截距与我们求得的权重矩阵与截距相差较大，虽呈收敛趋势，但相差极大，故没有输出收敛曲线图像，尝试修改代码中逻辑回归中的 λ 的值和正则项 1 范数和 2 范数的选择，仍没有达到可观效果，然后尝试修改求解器的选择与求解器中正则项的选择，仍不能达到可观收敛效果。

考虑将逻辑回归应用到分类问题，由于考虑到逻辑回归是线性回归+sigmoid 激活函数，采用如下激活函数对测试集进行分类预测

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

采用如下函数进行预测分类与计算得分

```
def predict1( X):
    global theory_Vw,theory_Vb
    y_hat = sigmoid(np.dot( theory_Vw,X.T) + theory_Vb)
    y_hat[y_hat >= 0.5] = 1
    y_hat[y_hat < 0.5] = 0
    return y_hat
def score1(Y,y):
    accuracy = (Y == y).sum() / len(y)
    print("Test set Accuracy*:",accuracy)
```

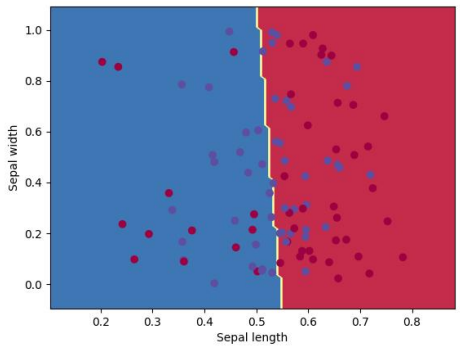
由于数据集维度较高，采取分割和降维的方式来使数据可视化

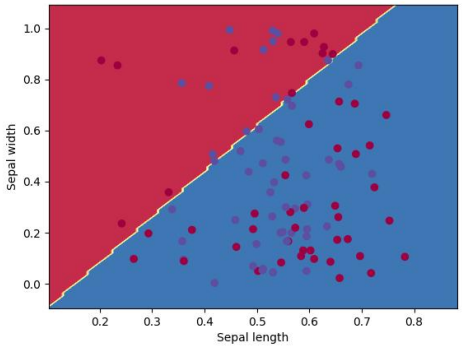
其中第一行为 sklearn 内置求解器求得的测试集准确度，第二行为逻辑回归算法求得的测试集准确度，第三行为 CPU 运算时间，左图为 sklearn 求解器求得的对测试集的分类边界（y=w.T*x+b）.右图为逻辑回归算法求得的对测试集的分类边界。

选取 1，2 维度

```
Test set Accuracy*: 0.57
Test set Accuracy:  0.51
Running time: 15.602251529693604 Seconds

进程已结束,退出代码0
```

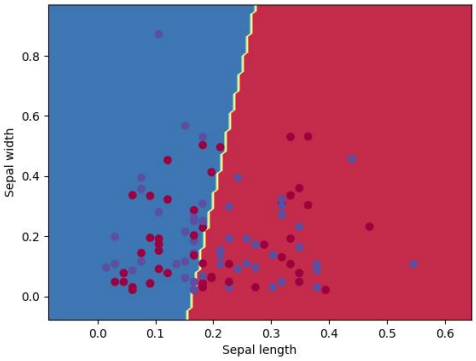


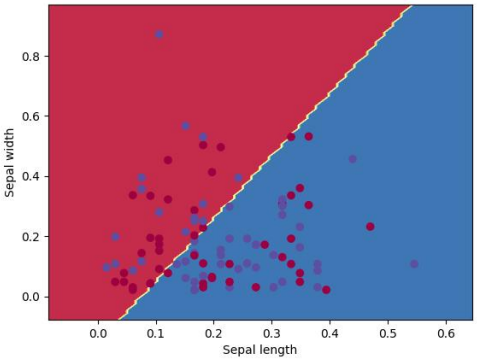


选取 3,4 维度

```
Test set Accuracy*: 0.45
Test set Accuracy:  0.58
Running time: 16.21587109565735 Seconds

进程已结束,退出代码0
```

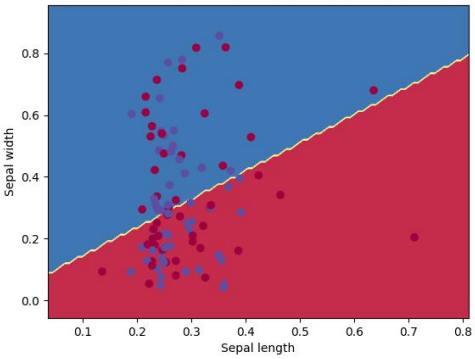


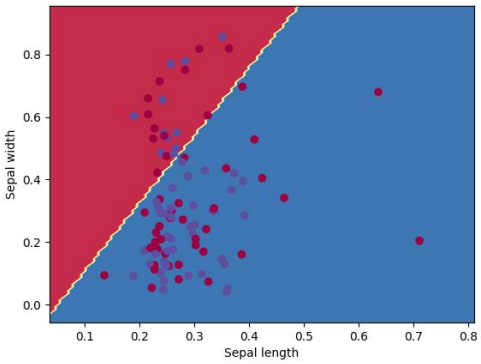


选取 5,6 维度

```
Test set Accuracy*: 0.49
Test set Accuracy:  0.53
Running time: 16.14130997657776 Seconds

进程已结束,退出代码0
```

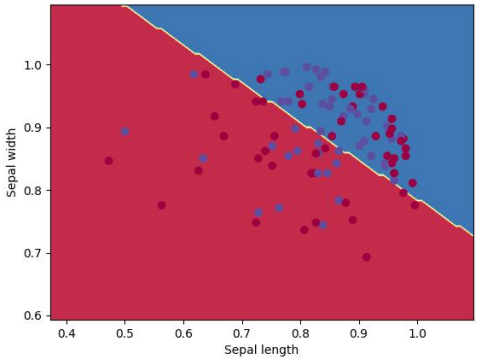


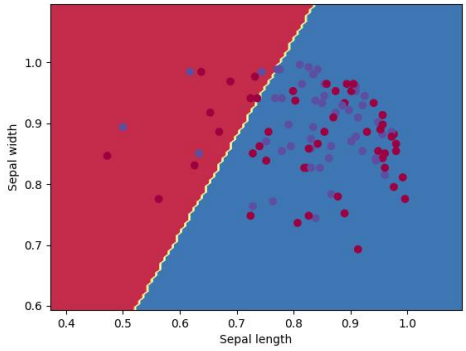


选取 7,8 维度

```
Test set Accuracy*: 0.58
Test set Accuracy:  0.56
Running time: 15.448865413665771 Seconds

进程已结束,退出代码0
```

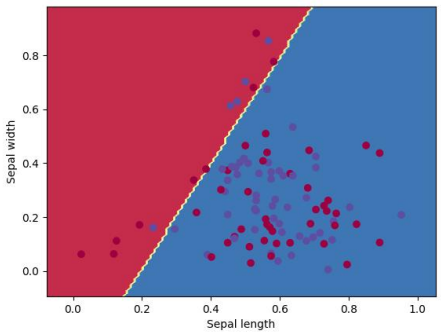


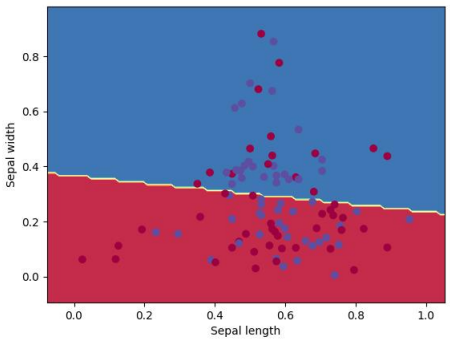


选取 9，10 维度

```
Test set Accuracy*: 0.55
Test set Accuracy:  0.57
Running time: 17.121548175811768 Seconds

进程已结束,退出代码0
```





综合考虑 10 个维度的准确度

第一次结果

```
Test set Accuracy*: 0.6
Test set Accuracy:  0.53
Running time: 29.384506464004517 Seconds

进程已结束,退出代码0
```

第二次结果

```
Test set Accuracy*: 0.56
Test set Accuracy:  0.51
Running time: 27.319608211517334 Seconds

进程已结束,退出代码0
```

基于此输出图像考虑未能实现逻辑回归收敛的可能原因是数据特征不明显，数据离散不明显，在换成未钳制在**[0,1]**的数据后，输出的图像仍有类似的问题，虽然数据呈离散分布，但对于特征相似的数据仍有可能处于不同类别，导致分类和收敛的效果均不佳。后考虑可能是由于训练集数据量仍不够大导致的，但在增大训练集数据量之后仍无法明显提高准确度。在尝试很多方式后仍无法解决此问题，后续考虑可以使用数据预处理和优化特征选择的方式来解决此问题，但还未能实现。

三、参考资料

【Python 实现逻辑回归(Logistic Regression) - CSDN App】http://t.csdnimg.cn/eWB83
【机器学习算法系列（五）- Lasso 回归算法（Lasso Regression Algorithm） - CSDN App】http://t.csdnimg.cn/Abapu
【逻辑回归 (Logistic Regression) 原理 (理论篇） - CSDN App】http://t.csdnimg.cn/TS98K
【LASSO 回归 - CSDN App】http://t.csdnimg.cn/ph0Eo