(a) (i)I use a iteration when y's value equals 1 I give the color "red" , else give the color "green", and I set the x axis as 'x_1' , set the y axis as 'x_2' , because on the plot the x-axis is the value of parameter 'x1 ', the y-axis is the value of the second parameter 'x1' , and the parameter of 'str' is the title of the plot .Since I have to plot many times using the same pattern , so I define a plot function in the "funcfile.py" :

```
def scattersth(x1, x2, y, str):
    for i in range(len(x1)):
        if y[i] == 1:
            plt.scatter(x1[i], x2[i], color='red', s=5)
        else:
            plt.scatter(x1[i], x2[i], color='green', s=5)
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    plt.title(str)
```

```
plt.subplot(331)
scattersth(x1, x2, y, "original data")
```

And just call it to plot the original data , and the subplot means this is the first image of 3*3=9 (all images), because I have many plots, this helps to see clearly:

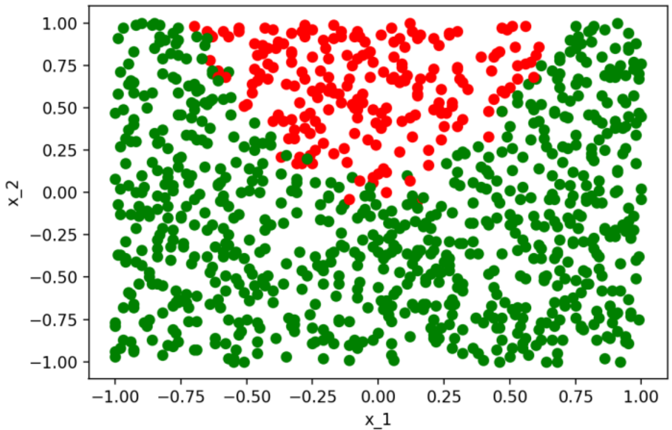


Figure 1: the original data , as we can see from the plot the red points mean the value of y is 1 , green ones mean -1 , we can see clearly there's a boundary of the two colors' points.

(ii)

```
#train the logistic regression classifier on the data
mul_lr = linear_model.LogisticRegression()
mul_lr.fit(x,y)
y_pre = mul_lr.predict(x)
print("linear_model.LogisticRegression :intercept {0}, slope {1} , cost {2}".format(mul_lr.intercept_, mul_lr.coef_,mul_lr.score
```

I use sklearn to train a logistic regression classifier on the data and the result is intercept : [-2.29502037], slope : [[-0.22934877 3.51212233]]

And the score of this model is 0.8268268268268268, which means this model's accuracy is 82.68% when predicts the data.

(iii)Because this requires different color with 2 datasets , so I define a function "scattersthothercolor" to plot the two datasets and could clearly see the different y value of the original ones and the prediction ones . When y=1 plot red point , y=-1 plot the green point , y_pre=1 plot the yellow point , y_pre=-1 plot the black point , in case the overlap so I set the size of point of the prediction data 1 while the original data is 5 , then we can see clearly the yellow points with red circle are the data when y's value equals y_pre , the whole green points are the data when y's value is -1 while the y_pre is 1 ; the black points with red circle is when y=1 and y_pre=-1 , the black points with green circle is when y=-1 and y_pre=-1.

```python
def scattersthothercolor(x1, x2, y,y_pre , str):
    for i in range(len(x1)):
        if y[i] == 1 :
            plt.scatter(x1[i], x2[i], color='red', s=5)
        if y_pre[i] == 1:
            plt.scatter(x1[i], x2[i], color='yellow', s=1)
        if y[i] == -1:
            plt.scatter(x1[i], x2[i], color='green', s=5)
        if y_pre[i] == -1 :
            plt.scatter(x1[i], x2[i], color='black', s=1)
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    plt.title(str)
```

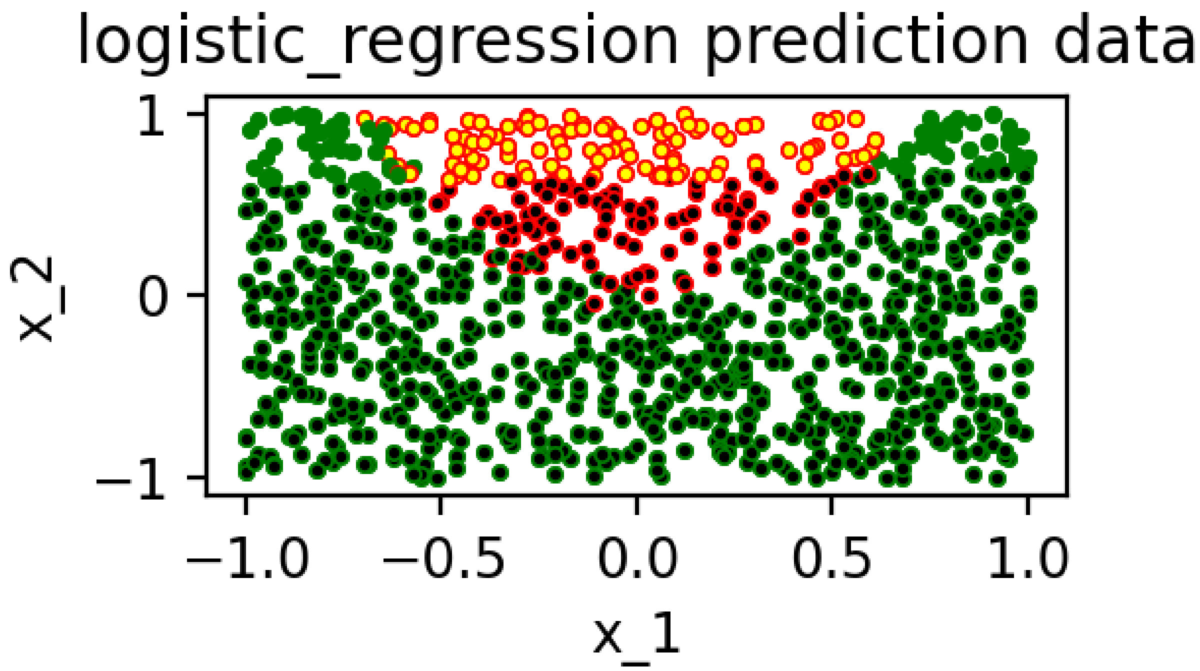


Figure 2 the prediction data with the original data

We usually use $w_1x_1 + w_2x_2 + b = 0$ to define decision boundary, when $h_w(x) = w_1x_1 + w_2x_2 + b > 0$ we classify it to y=1 , and the parameters b is the intercept and [w1,w2]=slop , so we can get the x2 equals -(w1x1+b)/w2 , so I define a function to draw the linear decision boundary in "funcfile.py" , the parameter "intercept" is the b , the parameter "coef" is the array of [w1 , w2] :

```
def plotsth(x, intercept, coef):
    plt.plot(x, -(intercept+coef[0][0]*x)/coef[0][1], color='blue', label='decision boundary')
    plt.legend()
```

So just recall it and the output image comes.

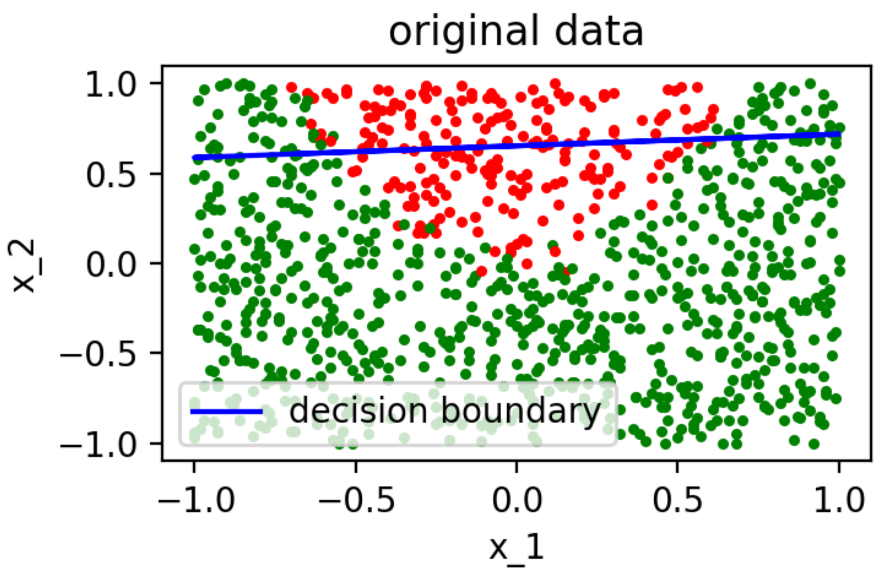


Figure 3: the decision boundary of the trained model in the original data

(iv) This prediction can't reflect the feature of all orignal data , I think it just return whichever class is the most likely, and the model can be improved to some extent , because the model is linear and through the plot of original data I think it's more like a curve using quadratic function to plot the decision boundary.

(b) (i)I pick 6 numbers of C to train the model , and print the parameters, after that predict the data then plot it , using the “add_subplot” function.

```
fig2 = plt.figure()
svc_c = [0.001,1,10,100,500,1000]
for i in range(6):
    linear_svc_model = LinearSVC(C=svc_c[i]).fit(x, y)
    print("linear_svc_model{0} :when C={1},intercept {2}, slope {3} ,score {4}".format(i+1,svc_c[i], linear_svc_model.intercept_,
                                                                                     linear_svc_model.coef_, linear_svc_model.score(x,y)))
    y_svc_pre = linear_svc_model.predict(x)
    linear_svc_model_subplot = fig2.add_subplot(2, 3, i+1)
    scattersth(x1, x2, y_svc_pre, "predictiondata when c is {0}".format(svc_c[i]))
    plotsth(x1, linear_svc_model.intercept_, linear_svc_model.coef_)

plt.show()
```

I found when C is too small or too large , the score of the model would decrese.

linear_svc_model1 :when C=0.001,intercept [-0.38304573], slope [[-0.02813036 0.31868212]] ,score 0.7877877877877878

linear_svc_model2 :when C=1,intercept [-0.78348024], slope [[-0.08562309 1.2261786]] ,score 0.8248248248248248

linear_svc_model3 :when C=10,intercept [-0.78858122], slope [[-0.08627484 1.23652484]] ,score 0.8268268268268268

linear_svc_model4 :when C=100,intercept [-0.80587576], slope [[-0.05154838 1.24861954]] ,score 0.8288288288288288

linear_svc_model5 :when C=500,intercept [-0.85731115], slope [[0.21140994 1.02179456]] ,score 0.7977977977977978

linear_svc_model6 :when C=1000,intercept [-1.17827911], slope [[0.03957859 1.56852242]] ,score 0.8118118118118118

(ii)

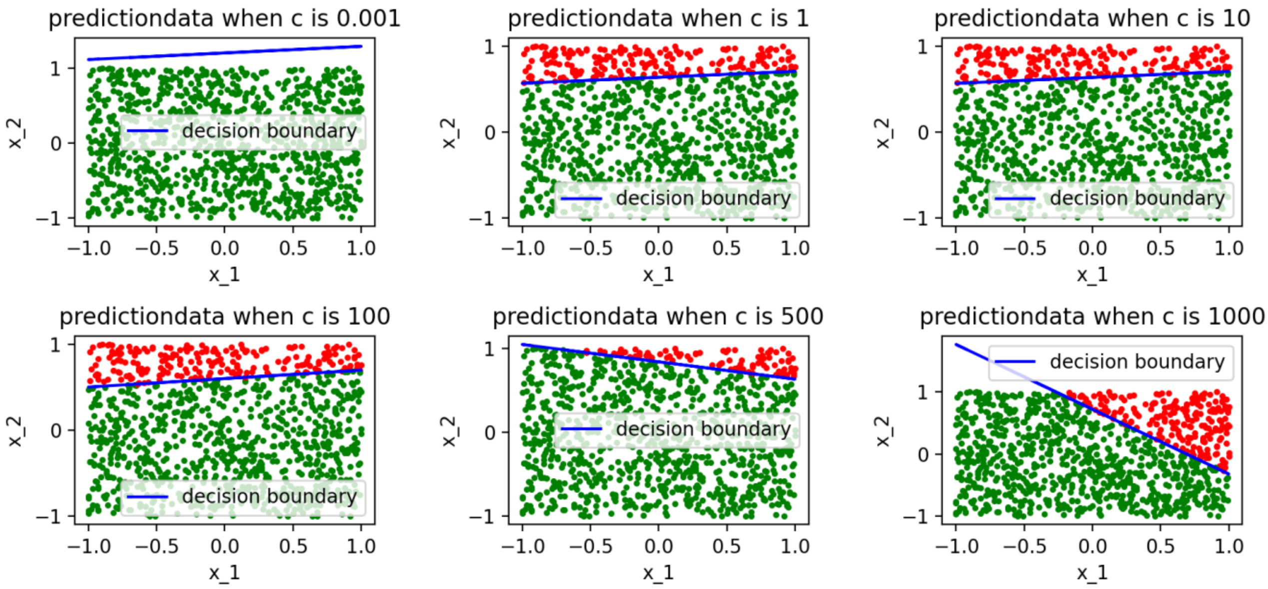


Figure 4 the prediction_data with different value of C

I find the if C is too small it's underfitting while too large is overfitting , and they are all linear prediction models. When the value of c limits to a range the prediction data doesn't vary so much , once out of that bound it varies much.

(iii) As we can see from the SVM cost function :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \max(0, 1 - y^{(i)}\theta^T x^{(i)}) + \theta^T\theta/C$$

The less value of C , the model would be underfitting, the higher the value of C , the less the model is regularized , and the less important penalty is ,that means you can tolerate higher cost function values. If C is so big that the SVM predictions would be changing at each run time , because it is over-fitting which means it doesn't generalize well and doesn't predict well for data outside the training set.

(c)

(i) To get the parameter:

```
x_new = np.array(df.iloc[:, 0:4])
mul_lr1 = linear_model.LogisticRegression()
mul_lr1.fit(x_new, y)
print("new linear_model.LogisticRegression :intercept {0}, slope {1} ".format(mul_lr1.intercept_, mul_lr1.coef_))
```

(new linear_model.LogisticRegression)

intercept [-0.80227213]

slope [[-0.22141638   5.35919863 -8.05566943 -0.26219426]]

(ii) I use subplot function and recall the "scattersth" function to plot the two sets of data.

```
y_pre1 = mul_lr1.predict(x_new)
plt.subplot(121)
scattersth(x1, x2, y_pre1, "new linear_model.LogisticRegression")
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')
plt.legend()
plt.subplot(122)
scattersth(x1, x2, y, "original data")
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')
plt.legend()
```

When add two features the graph , as we can see from the baseline , the new model's prediction doesn't change much compared with the original data.

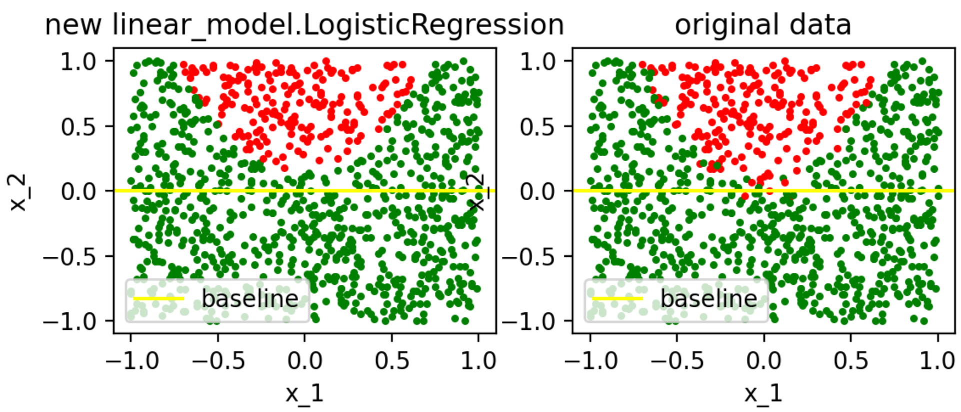
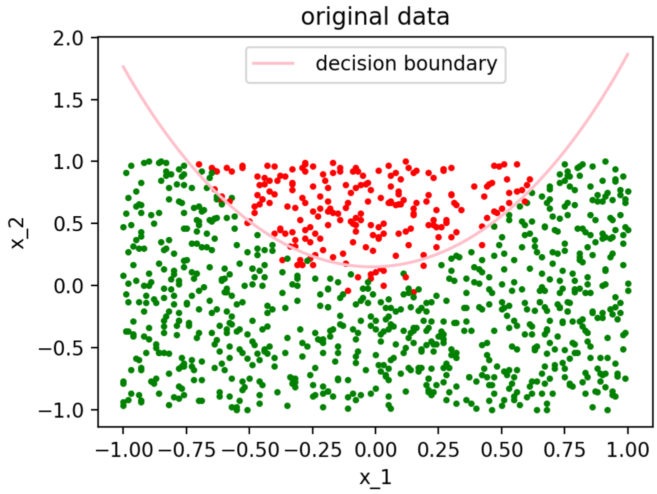


Figure 5 the new model and the original data with the y=0 baseline



(iii)

Figure 6 the decision boundary of the original data

I define each of the coefficient as a,b,c,d , and e is the intercept , then I apply the mathematic quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

,and in this case m is the c in the formula because we can regard the x1 as constant to get the x2_new , and then call the plot function.

```
a = mul_lr1.coef_[0][0]
b = mul_lr1.coef_[0][1]
c = mul_lr1.coef_[0][2]
d = mul_lr1.coef_[0][3]
e = mul_lr1.intercept_
m = e + a*x1_new + c*x1_new*x1_new
x2_new = (-b+(b*b-4*d*m)**0.5)/(2*d)
plt.plot(x1_new, x2_new, label=' decision boundary', color='pink')
scattersth(x1, x2, y, "original data")
plt.legend()
plt.show()
```

Appendix

week2.py

```
import numpy as np
import pandas as pd
from funcfile import *
from sklearn import linear_model
from sklearn.svm import LinearSVC
```

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

#read data
df = pd.read_csv("week2.csv",comment="#")
print(df.head())
x = np.array(df.iloc[:, 0:2])#read the first two colunms as array
y = np.array(df.iloc[:, 4])#read the y value

#train the logistic regression classifier on the data
mul_lr = linear_model.LogisticRegression()
mul_lr.fit(x,y)
y_pre = mul_lr.predict(x)
print("linear_model.LogisticRegression :intercept {0}, slope {1} ,
score {2}".format(mul_lr.intercept_, mul_lr.coef_,mul_lr.score(x,y)))

x1 = x[:, 0]
x2 = x[:, 1]


plt.subplot(331)
scattersth(x1, x2, y, "original data")
plotsth(x1, mul_lr.intercept_, mul_lr.coef_)

plt.subplot(332)
plt.plot()
scattersthothercolor(x1, x2, y,y_pre, "logistic_regression prediction
data")
plt.show()
#pick 6 numbers of C to train the model , and print the parameters,
#after that predict the data then plot it , using the “add_subplot”
function
fig2 = plt.figure()
svc_c = [0.001,1,10,100,500,1000]
for i in range(6):
    linear_svc_model = LinearSVC(C=svc_c[i]).fit(x, y)
    print("linear_svc_model{0} :when C={1},intercept {2}, slope
{3} ,score {4}".format(i+1,svc_c[i], linear_svc_model.intercept_,

linear_svc_model.coef_, linear_svc_model.score(x,y)))
    y_svc_pre = linear_svc_model.predict(x)
    linear_svc_model_subplot = fig2.add_subplot(2, 3, i+1)
    scattersth(x1, x2, y_svc_pre, "predictiondata when c is
{0}".format(svc_c[i]))
```

```
    plotsth(x1, linear_svc_model.intercept_, linear_svc_model.coef_)

plt.show()


# (c)
x_new = np.array(df.iloc[:, 0:4])
mul_lr1 = linear_model.LogisticRegression()
mul_lr1.fit(x_new, y)
print("new linear_model.LogisticRegression :intercept {0}, slope {1}
".format(mul_lr1.intercept_, mul_lr1.coef_))
y_pre1 = mul_lr1.predict(x_new)
plt.subplot(121)
scattersth(x1, x2, y_pre1, "new linear_model.LogisticRegression")
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')
plt.legend()
plt.subplot(122)
scattersth(x1, x2, y, "original data")
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')
plt.legend()
"""define each of the coefficient as a,b,c,d , and e is the
intercept , then I apply the mathematic quadratic formula """
x1_new = np.linspace(-1, 1, 9999)#to get the axis of x
a = mul_lr1.coef_[0][0]
b = mul_lr1.coef_[0][1]
c = mul_lr1.coef_[0][2]
d = mul_lr1.coef_[0][3]
e = mul_lr1.intercept_
m = e + a*x1_new + c*x1_new*x1_new
x2_new = (-b+(b*b-4*d*m)**0.5)/(2*d)
plt.plot(x1_new, x2_new, label=' decision boundary', color='pink')
scattersth(x1, x2, y, "original data")
plt.legend()
plt.show()
```

"funcfile.py"

```
import matplotlib.pyplot as plt
import sympy as sp

#plot the data
"""when y's value equals 1 color is "red" ,else "green", and set the
x axis as 'x_1' ,
set the y axis as 'x_2' , because on the plot the x-axis is the value
of parameter 'x1 ',
```

```
the y-axis is the value of the second parameter ‘x1’ , and the
parameter of ‘str’ is
the title of the plot"""
def scattersth(x1, x2, y, str):
    for i in range(len(x1)):
        if y[i] == 1:
            plt.scatter(x1[i], x2[i], color='red', s=5)
        else:
            plt.scatter(x1[i], x2[i], color='green', s=5)
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    plt.title(str)

""" plot the two datasets and could clearly see the different y value
of the original ones and the prediction ones"""
def scattersthothercolor(x1, x2, y,y_pre , str):
    for i in range(len(x1)):
        if y[i] == 1 :
            plt.scatter(x1[i], x2[i], color='red', s=5)
        if y_pre[i] == 1:
            plt.scatter(x1[i], x2[i], color='yellow', s=1)
        if y[i] == -1:
            plt.scatter(x1[i], x2[i], color='green', s=5)
        if y_pre[i] == -1 :
            plt.scatter(x1[i], x2[i], color='black', s=1)
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    plt.title(str)

"""plot the line with intercept and coef"""
def plotsth(x, intercept, coef):
    plt.plot(x, -(intercept+coef[0][0]*x)/coef[0][1], color='blue',
label='decision boundary')
    plt.legend()
```