Xiangyu Zheng 21331025 CS7CS4 Machine Learning

(a)(i)

I use four arrays to record the x and y coordinates of -1 and 1:

px,py,nx,ny = [],[],[],[]  
for d in dataArr[1:]:  
 nums = d.split(',')  
 if nums[2] == '1':  
 px.append(float(nums[0]))  
 py.append(float(nums[1]))  
 else:  
 nx.append(float(nums[0]))  
 ny.append(float(nums[1]))

This is the distribution graph of all -1s and 1s:

图表, 散点图

描述已自动生成

(a)(ii)

My model uses the following parameters:

1. penalty=‘l2’, To prevent over-fitting, the model features few, so choose L2
2. multi\_class=‘ovr’, Two classification
3. max\_iter=1000, Algorithm convergence maximum number of iterations
4. fit\_intercept=True, Add variance to the model
5. tol=0.0001, Convergence error range
6. coefficient of x:0.13313
7. coefficient of y:-4.50365
8. intercept:1.65408

….

The coefficients of x and y are obtained through the cost function:

According to the decision boundary formula

the larger the feature coefficient, The greater the impact on the forecast.

Features with too many or too few dimensions will lead to reduced predictions, because learning too many useless features will lead to overfitting, and too few dimensions will lead to underfitting. So features with moderate dimensions will increase predictions

(a)(iii)

We have the sigmoid function:

We bring the feature values into the sigmoid function to get:

We can analyze ’s value range:

There are two features in this sample: x1,x2, So we have decision boundaries:

According to the above formula, we can get：

We can first set the x coordinates of 100 decision boundaries:

x1\_plot = np.linspace(-1, 1,100)

We can get the coefficient array and intercept through the trained model:

lr = LR(penalty='l2', max\_iter=1000)

lr.coef\_

lr.intercept\_[0]

Putting the coefficient and intercept into the formula, we can calculate the y coordinate of the decision boundary:

x2\_plot = (-lr.coef\_[0][0] \* x1\_plot - lr.intercept\_[0]) / lr.coef\_[0][1]

Next, we just need to draw it:

plt.plot(x1\_plot, x2\_plot)

图表, 散点图

描述已自动生成

(a)(iv)

As we can see from the graph in (a)(i), it is obvious that the dividing line between 1 and -1 is a curve, but the decision boundary of my model is a straight line, so when forecasting, the data from upper left corner , upper right corner and below the middle of the decision boundary are all predicted incorrectly. So I feel that my model classification effect is not good for these ‘special data’.

(b) (i)

Three models were used for this task:

1.parameters:

1. penalty=’l2’ Use l2 regularization
2. loss=’squared\_hinge’ loss function
3. dual=True Turned into a dual problem
4. C=0.001 Penalty coefficient
5. fit\_intercept=True Calculate the intercept
6. max\_iter=1000 The maximum number of iterations
7. coefficient of x:0.01109
8. coefficient of y:-0.38334
9. intercept:0.23246

2.parameters:

1. penalty=’l2’ Use l2
2. loss=’squared\_hinge’
3. dual=True
4. C=1
5. fit\_intercept=True
6. max\_iter=1000
7. coefficient of x:0.05306
8. coefficient of y:-1.69412
9. intercept:0.60767

3.parameters:

1. penalty=’l2’
2. loss=’squared\_hinge’
3. dual=True
4. C=100
5. fit\_intercept=True Calculate the intercept
6. max\_iter=1000 The maximum number of iterations
7. coefficient of x:0.02683
8. coefficient of y:-1.83350
9. intercept:0.58653

(b) (ii)

The following are three LinearSVC diagrams with different Cs, and the decision boundary. The calculation method of the decision boundary is the same as in (a) (iii)

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

图表, 散点图

描述已自动生成

(b) (iii)

According to the svm cost function:

The bigger C is, the smaller the influence on θ, the smaller C is, the bigger the influence on θ’s calculation.

According to the hard-spaced margin line formula:

And soft margin formula:

Lines with soft and hard margin are drawn as follow when C=100 and C=0.001:

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

We can see that when C=0.001, the soft margin is much larger than when C=100, which shows that the model has a large tolerance for sample errors, which will reduce the accuracy of the model's prediction. When C is relatively large, it will cause over-fitting, but it is not reflected in the above figure.

(b) (iv)

Through the three linearSVC and logistic regression graphs, we can get that the coefficient of x and intercept in the logistic regression model are larger, and the proportion of x is more when it predicts. In the linearSVC model, as the value of C becomes larger, the coefficient of x and intercept become larger and larger, while the coefficient of y becomes smaller and smaller, the prediction accuracy of the model seems to be getting better and better.

(c) (i)

The following are the parameters of the trained model:

1. penalty=‘l2’, To prevent over-fitting, the model features few, so choose L2
2. multi\_class=‘ovr’, Two classification
3. max\_iter=1000, Algorithm convergence maximum number of iterations
4. fit\_intercept=True, Add variance to the model
5. tol=0.0001, Convergence error range
6. coefficient of x:-0.03959
7. coefficient of x:-6.11030
8. coefficient of :5.57693
9. coefficient of :-0.96831
10. intercept:0.45464

(ii)

The following is the point distribution diagram and the predicted distribution diagram after adding 2 features. Basically, the classification is successful, and the dividing line becomes a curve, which basically fits the data perfectly.

图表, 散点图

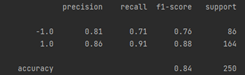
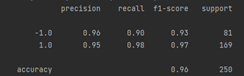
描述已自动生成

(iii)

I chose the logistic regression model in (a) for comparison.

First print out the classification reports of the two models:

print(classification\_report(y\_test, lr.predict(X\_test)))

Logistic regression with 2 features

Logistic regression with 4 features

First we pass 4 parameters:

1. precision

Indicates the actual number of positive samples among the samples predicted to be positive

1. recall

Indicates the proportion of samples that are actually positive that are judged to be positive

1. fi-score

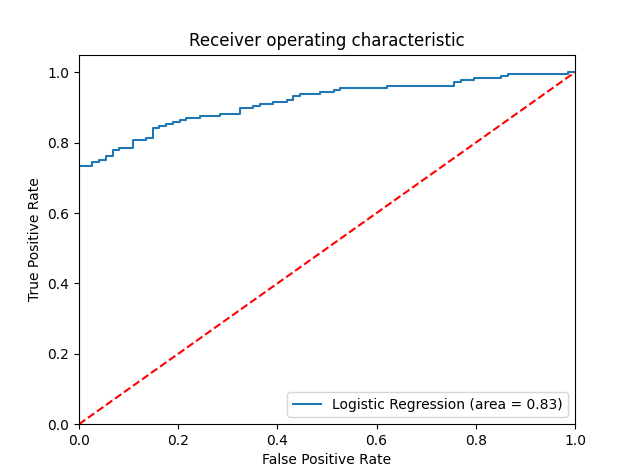
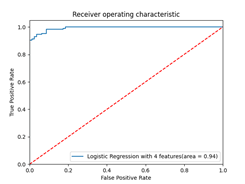
Weighted harmonic average of Precision and Recall

1. accuracy

It refers to the ratio of the sample book correctly classified by the classifier to the total number of samples for a given test data set, that is, the probability of correct prediction.

We can see that no matter which number it is, the performance of the logistic regression model with four feature is much better

Next we look at the ROC diagram:

Logistic regression with 2 features

Logistic regression with 4 features

The area under the curve is also shown in the figure. The logistic regression model with 4 features is larger, indicating better performance.

(iv)

First we added 2 features, so our decision boundary formula is:

(1)

Because and are the squares of and , we have:

(2)

Because we know the x coordinate, we need to solve for the y coordinate. The y coordinate is the value of . We want the equation to have no square of , so we first solve for the value of :

(3)

We substitute formula 3 into formula 2 to get:

(4)

After simplifying the formula, we get:

(5)

What we are asking for is , so there is:

(6)

Finally we have:

(7)

Next we draw the decision boundary

First, we first generate 200 x coordinates, the interval is -1 to 1:

x1\_plot = np.linspace(-1, 1, 200)

Then we calculate the coordinate array of x2 according to the formula:

x2\_plot=[]  
for i,v in enumerate(x1\_plot):  
 a = (-lr.coef\_[0][0]\*v-lr.coef\_[0][2]\*math.pow(v,2)) / lr.coef\_[0][1]  
 x2\_plot.append(a)

x2\_plot = np.array(x2\_plot)

Finally, we draw the decision boundary and get the picture::

plt.plot(x1\_plot, x2\_plot)

图表, 散点图

描述已自动生成

Appendix

code

(a)(i)

1. **import** numpy as np
2. **import** pandas as pd
3. **import** matplotlib.pyplot as plt
5. # Read data
6. data = open("data")
7. dataArr = str(data.read()).split('\n')
9. #Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
10. px,py,nx,ny = [],[],[],[]
11. **for** d **in** dataArr[1:]:
12. nums = d.split(',')
13. **if** nums[2] == '1':
14. px.append(float(nums[0]))
15. py.append(float(nums[1]))
16. **else**:
17. nx.append(float(nums[0]))
18. ny.append(float(nums[1]))
20. #Draw a picture
21. fig,square = plt.subplots(figsize=(7, 7))
22. square.scatter(px,py,s=10,c='r',marker ='o',label = '1')
23. square.scatter(nx,ny,s=10,c='b',marker ='x',label = '-1')
24. square.legend()
25. square.set\_xlabel("x-axis")
26. square.set\_ylabel("y-axis")
27. plt.show()

(a)(ii and iii)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** sklearn.linear\_model **import** LogisticRegression as LR
4. **from** sklearn.model\_selection **import** train\_test\_split
5. **from** sklearn.metrics **import** roc\_curve, classification\_report, roc\_auc\_score
7. # Read data
8. dataText = open("data")
9. dataArr = str(dataText.read()).split('\n')
11. # Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
12. px, py, nx, ny = [], [], [], []
13. data, target = [], []
14. **for** d **in** dataArr[1:]:
15. nums = d.split(',')
16. dxy = []
17. dxy.append(float(nums[0]))
18. dxy.append(float(nums[1]))
19. data.append(dxy)
20. target.append(float(nums[2]))
21. **if** nums[2] == '1':
22. px.append(float(nums[0]))
23. py.append(float(nums[1]))
24. **else**:
25. nx.append(float(nums[0]))
26. ny.append(float(nums[1]))
28. # Divide the data set into a training set and a test set
29. X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, target)
30. # Training model
31. lr = LR(penalty='l2', max\_iter=1000, tol=0.0001)
32. lr.fit(X\_train, y\_train)
34. # Prediction data set
35. y\_pred = lr.predict(data)
36. y\_test\_pred = lr.predict(X\_test)
38. # Print classification report
39. **print**(classification\_report(y\_test, y\_test\_pred))
40. fig, square = plt.subplots(figsize=(7, 7))
42. # Get accuracy and other data
43. y\_prob = lr.predict\_proba(X\_test)[:, 1]
44. logit\_roc\_auc = roc\_auc\_score(y\_test, lr.predict(X\_test))
45. fpr\_lr, tpr\_lr, threshold\_lr = roc\_curve(y\_test, y\_prob)
47. # Draw the ROC curve graph
48. plt.figure()
49. plt.plot(fpr\_lr, tpr\_lr, label='Logistic Regression (area = %0.2f)' % logit\_roc\_auc)
50. plt.plot([0, 1], [0, 1], 'r--')
51. plt.xlim([0.0, 1.0])
52. plt.ylim([0.0, 1.05])
53. plt.xlabel('False Positive Rate')
54. plt.ylabel('True Positive Rate')
55. plt.title('Receiver operating characteristic')
56. plt.legend(loc="lower right")
57. plt.show()
59. # Draw a dot plot
60. y\_pred = list(y\_pred)
61. ppx, ppy, pnx, pny = [], [], [], []
62. **for** i, v **in** enumerate(data):
63. **if** y\_pred[i] == 1:
64. ppx.append(v[0])
65. ppy.append(v[1])
66. **else**:
67. pnx.append(v[0])
68. pny.append(v[1])
69. square.scatter(px, py, s=30, c='r', marker='o', label='1')
70. square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
71. square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
72. square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')
74. # Draw the decision boundary
75. x1\_plot = np.linspace(-1, 1, 100)
76. x2\_plot = (-lr.coef\_[0][0] \* x1\_plot - lr.intercept\_[0]) / lr.coef\_[0][1]
77. plt.plot(x1\_plot, x2\_plot)
79. square.legend()
80. plt.title("Logistic Regression")
81. square.set\_xlabel("x-axis")
82. square.set\_ylabel("y-axis")
83. plt.show()

(b)(i)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** sklearn.metrics **import** classification\_report
4. **from** sklearn.svm **import** LinearSVC
5. **from** sklearn.model\_selection **import** train\_test\_split
7. # LinearSVC related classes
8. # Read data
9. dataText = open("data")
10. dataArr = str(dataText.read()).split('\n')
12. # Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
13. px, py, nx, ny = [], [], [], []
14. data, target = [], []
15. **for** d **in** dataArr[1:]:
16. nums = d.split(',')
17. dxy = []
18. dxy.append(float(nums[0]))
19. dxy.append(float(nums[1]))
20. data.append(dxy)
21. target.append(float(nums[2]))
22. **if** nums[2] == '1':
23. px.append(float(nums[0]))
24. py.append(float(nums[1]))
25. **else**:
26. nx.append(float(nums[0]))
27. ny.append(float(nums[1]))
29. # Divide the data set into a training set and a test set
30. X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, target)
32. # Training model
33. LinearSVC = LinearSVC(penalty='l2', loss='squared\_hinge', dual=True, C=100, fit\_intercept=True, max\_iter=1000)
34. LinearSVC.fit(X\_train, y\_train)
36. # Prediction data set
37. y\_pred = LinearSVC.predict(data)
38. y\_pred = list(y\_pred)
39. y\_test\_pred = LinearSVC.predict(X\_test)
41. # Print classification report
42. **print**(classification\_report(y\_test, y\_test\_pred))
44. # Draw a dot plot
45. ppx, ppy, pnx, pny = [], [], [], []
46. **for** i, v **in** enumerate(data):
47. **if** y\_pred[i] == 1:
48. ppx.append(v[0])
49. ppy.append(v[1])
50. **else**:
51. pnx.append(v[0])
52. pny.append(v[1])
54. fig, square = plt.subplots(figsize=(7, 7))
55. square.scatter(px, py, s=30, c='r', marker='o', label='1')
56. square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
57. square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
58. square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')
60. # Draw the decision boundary
61. x1\_plot = np.linspace(-1, 1, 200)
62. x2\_plot = (-LinearSVC.coef\_[0][0] \* x1\_plot - LinearSVC.intercept\_[0]) / LinearSVC.coef\_[0][1]
63. plt.plot(x1\_plot, x2\_plot)
65. # Draw the soft margin and hardmargin of LinearSVC
66. # w0\*x0 + w1\*x1 + b = 1
67. h\_up\_y = -LinearSVC.coef\_[0][0] / LinearSVC.coef\_[0][1] \* x1\_plot - LinearSVC.intercept\_[0] / LinearSVC.coef\_[0][
68. 1] + 1 / LinearSVC.coef\_[0][1]
69. # w0\*x0 + w1\*x1 + b = -1
70. h\_down\_y = -LinearSVC.coef\_[0][0] / LinearSVC.coef\_[0][1] \* x1\_plot - LinearSVC.intercept\_[0] / LinearSVC.coef\_[0][
71. 1] - 1 / LinearSVC.coef\_[0][1]
72. up\_index = (h\_up\_y >= -1) & (h\_up\_y <= 1)
73. down\_index = (h\_down\_y >= -1) & (h\_down\_y <= 1)
74. plt.plot(x1\_plot[up\_index], h\_up\_y[up\_index], color='black')
75. plt.plot(x1\_plot[down\_index], h\_down\_y[down\_index], color='black')
76. # w0\*x0 + w1\*x1 + b = 1-
77. h\_up\_y = -LinearSVC.coef\_[0][0] / LinearSVC.coef\_[0][1] \* x1\_plot - LinearSVC.intercept\_[0] / LinearSVC.coef\_[0][
78. 1] + 0.5 / LinearSVC.coef\_[0][1]
79. # w0\*x0 + w1\*x1 + b = -1
80. h\_down\_y = -LinearSVC.coef\_[0][0] / LinearSVC.coef\_[0][1] \* x1\_plot - LinearSVC.intercept\_[0] / LinearSVC.coef\_[0][
81. 1] - 0.5 / LinearSVC.coef\_[0][1]
82. up\_index = (h\_up\_y >= -1) & (h\_up\_y <= 1)
83. down\_index = (h\_down\_y >= -1) & (h\_down\_y <= 1)
84. plt.plot(x1\_plot[up\_index], h\_up\_y[up\_index], color='black', linestyle='--')
85. plt.plot(x1\_plot[down\_index], h\_down\_y[down\_index], color='black', linestyle='--')
87. square.legend()
88. plt.title("LinearSVC with C=0.001")
89. square.set\_xlabel("x-axis")
90. square.set\_ylabel("y-axis")
91. plt.show()

(c)

1. **import** math
2. **import** numpy as np
3. **import** matplotlib.pyplot as plt
4. **from** sklearn.linear\_model **import** LogisticRegression as LR
5. **from** sklearn.metrics **import** classification\_report, roc\_auc\_score, roc\_curve
6. **from** sklearn.model\_selection **import** train\_test\_split
8. # Logistic regression 4 features
9. # Read data
10. dataText = open("newdata.txt")
11. dataArr = str(dataText.read()).split('\n')
13. # Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
14. px, py, nx, ny = [], [], [], []
15. data, target = [], []
16. **for** d **in** dataArr:
17. nums = d.split(',')
18. **print**(nums)
19. dxy = []
20. dxy.append(float(nums[0]))
21. dxy.append(float(nums[1]))
22. dxy.append(float(nums[2]))
23. dxy.append(float(nums[3]))
24. data.append(dxy)
25. target.append(float(nums[4]))
26. **if** nums[4] == '1':
27. px.append(float(nums[0]))
28. py.append(float(nums[1]))
29. **else**:
30. nx.append(float(nums[0]))
31. ny.append(float(nums[1]))
33. # Divide the data set into a training set and a test set
34. X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, target)
36. # Training model
37. lr = LR(penalty='l2', max\_iter=1000, tol=0.0001)
38. lr.fit(X\_train, y\_train)
40. # Prediction data set
41. y\_pred = lr.predict(data)
43. # Print classification report
44. **print**(classification\_report(y\_test, lr.predict(X\_test)))
45. fig, square = plt.subplots(figsize=(7, 7))
47. # Get accuracy and other data
48. y\_prob = lr.predict\_proba(X\_test)[:,1]
49. logit\_roc\_auc = roc\_auc\_score(y\_test, lr.predict(X\_test))
50. fpr\_lr,tpr\_lr,threshold\_lr = roc\_curve(y\_test,y\_prob)
52. # Draw the ROC curve graph
53. plt.figure()
54. plt.plot(fpr\_lr, tpr\_lr, label='Logistic Regression with 4 features(area = %0.2f)' % logit\_roc\_auc)
55. plt.plot([0, 1], [0, 1],'r--')
56. plt.xlim([0.0, 1.0])
57. plt.ylim([0.0, 1.05])
58. plt.xlabel('False Positive Rate')
59. plt.ylabel('True Positive Rate')
60. plt.title('Receiver operating characteristic')
61. plt.legend(loc="lower right")
62. plt.show()
64. # Draw a dot plot
65. y\_pred = list(y\_pred)
66. ppx, ppy, pnx, pny = [], [], [], []
67. **for** i, v **in** enumerate(data):
68. **if** y\_pred[i] == 1:
69. ppx.append(v[0])
70. ppy.append(v[1])
71. **else**:
72. pnx.append(v[0])
73. pny.append(v[1])
74. square.scatter(px, py, s=30, c='r', marker='o', label='1')
75. square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
76. square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
77. square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')
79. # Draw the decision boundary
80. x1\_plot = np.linspace(-1, 1, 200)
81. x2\_plot=[]
82. **for** i,v **in** enumerate(x1\_plot):
83. a = (-lr.coef\_[0][0]\*v-lr.coef\_[0][2]\*math.pow(v,2)) / lr.coef\_[0][1]
84. x2\_plot.append(a)
85. x2\_plot = np.array(x2\_plot)
86. plt.plot(x1\_plot, x2\_plot)
88. square.legend()
89. plt.title("Logistic Regression with 4 features")
90. square.set\_xlabel("x-axis")
91. square.set\_ylabel("y-axis")
92. plt.show()