Xiangyu Zheng 21331025 CS7CS4 Machine Learning week4 assignment

(i)(a)

I first determine the value of the maximum order of the polynomial: set C=0.1, C=1, C=10 and use ten-fold cross-validation to draw error bars of degree=1 to 9, you can get:

for i in range(1, 10):  
 poly\_train = PolynomialFeatures(degree=i).fit\_transform(train)  
 LR = LogisticRegression(penalty='l2', C=C, max\_iter=10000, tol=0.0001)  
 scores = cross\_val\_score(LR, poly\_train, target, cv=10)  
 f1\_score = cross\_val\_score(LR, poly\_train, target, scoring='f1', cv=10)

图表, 折线图

描述已自动生成

图表, 箱线图

描述已自动生成

图片包含 图形用户界面

描述已自动生成

As can be seen from the above figure, through the accuracy and the value of f1\_score, I initially chose degree=2 or 3, because starting from C=1 and starting from degree=2, the accuracy of the following polynomials and the value of f1 are not much different .

As you can see from the above figure, starting from C=1, degree=2, 3 have similar performance, so draw their scatter plot and decision boundary under C=1:

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 degree=2, the performance of the classifier is good , and the decision boundary does not change when degree=3. So I chose 2 for degree.

Then I looked for the best C based on the maximum order of polynomial being 2. I set two value ranges from C=0.001 to 1, increasing by 0.05 each time and C=1 to 1000 each time increasing by 20, and you can get error bar:

图表

描述已自动生成

图表, 直方图

描述已自动生成

Through the comparison of the above two figures, we can know that when C=40, the accuracy and f1 values are the highest, and the standard deviation value is the smallest. So I choose C=40 and degree=2 as the best values for my model.

(b)

I first determine the range of K from 1-30, and draw cross-validation error bars. It can be determined that when k=3, the accuracy is the highest and the error is the smallest. Although there are better K values in the future, the overall trend is declining, and the larger the K, the more data needs to be calculated, and the more memory it consumes. So initially choose K=3:图表

描述已自动生成

We draw a scatter plot and the decision boundary of the knn model according to K=1 and k=3:

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 k=1, the decision boundary fits each point -1 and 1, and there is an over-fitting phenomenon. The decision boundary of k=3 is better. According to the data of the error bar, when k=3, it is the best choice.

(c)

Three models for calculating confusion matrix:

1.Logistic regression, the heighest order of the polynomial is 2, using L2 regularization, C=40

2.KNN, K=3

3.DummyClassifier, using random prediction strategy

poly\_features = PolynomialFeatures(degree=2)  
poly\_train = poly\_features.fit\_transform(X\_train)

LR = LogisticRegression(penalty='l2', C=40)

knn = KNeighborsClassifier(n\_neighbors=3)

dummy = DummyClassifier(strategy="uniform")

According to the confusion matrix:

|  |  |
| --- | --- |
| True positives | False positives |
| False negatives | True negatives |

I divided the training set into 20% of the test set, and then used the confusion\_matrix function in sklearn to calculate the confusion matrix, and I could get:

confusion\_matrix(y\_test, poly\_pred)

1.Logistic regression

|  |  |
| --- | --- |
| True positives:70 | False positives:3 |
| False negatives:5 | True negatives:133 |

2.KNN

|  |  |
| --- | --- |
| True positives:70 | False positives:3 |
| False negatives:5 | True negatives:133 |

3.DummyClassifier

|  |  |
| --- | --- |
| True positives:33 | False positives:40 |
| False negatives:68 | True negatives:70 |

(d)

First, according to the ROC accuracy formula:

Use the function in skleanr to get the accuracy:

knn\_roc\_auc = roc\_auc\_score(y\_test, knn.predict(X\_test))

Then use the predict\_proba function to get the probability of predicting the target of each value in the test set

y\_prob\_knn = knn.predict\_proba(X\_test)[:, 1]

Then use the roc\_curve function and the target and prediction probability matrix of the parameter test set to get the false positive rate and true positive rate arrays

fpr\_lr\_knn, tpr\_lr\_knn, threshold\_lr\_knn = roc\_curve(y\_test, y\_prob\_knn)

Finally, draw the roc curves of the three models together, and get the following figure:

图表, 折线图

描述已自动生成

(e)

We can clearly see from the data in c and d that the performance of knn classifier and logistic regression is significantly better than that of Dummy Classifier based on random classification, but the performance of knn and logistic regression is equivalent. I would recommend the knn classifier under the data of this two classification, because the parameters of knn are relatively simple, it is only necessary to determine the value of k through cross-validation, while it is more troublesome to find the best hyperparameters through cross-validation in logistic regression.

(ii) (a)

We can use the same steps as (i)(a). The best polynomial is not to use the polynomial, and to train with the original data, we can use cross-validation to get the error bar:

图表, 条形图, 直方图

描述已自动生成

图表, 直方图

描述已自动生成

We can see that because there are too few features, C using L2 regularization seems to have lost its effect, regardless of the accuracy and error of C.

Finally, by drawing a scatter plot, we can see that the logistic regression model predicts the data to -1:

图表, 散点图

描述已自动生成

(b)

According to the same steps as (i)(b), we can get the error bar graph of KNN after cross-validation

图表

描述已自动生成

We can see that the best value is K=13, so draw a scatter plot with k=13 and k=1:

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 k=1, it still fits every training point, resulting in overfitting and decreased prediction accuracy.

(c)

I chose 3 models:

1 Logistic regression, C=1, no polynomial is used.

2. KNN, K=13

3.DummyClassifier

Following the same steps as (i)(c), we can get three confusion matrices:

1.Logistic regression

|  |  |
| --- | --- |
| True positives:0 | False positives:54 |
| False negatives:0 | True negatives:87 |

2.KNN

|  |  |
| --- | --- |
| True positives:10 | False positives:44 |
| False negatives:8 | True negatives:79 |

3.DummyClassifier

|  |  |
| --- | --- |
| True positives:27 | False positives:27 |
| False negatives:36 | True negatives:51 |

We can see that logistic regression because all predictions have become one value, resulting in both TP and FN being 0

(d)

In the same steps as (i)(d), we get the ROC curves of the three models:

图表, 折线图

描述已自动生成

(e)

We can see from the data in (c) and (d) that in this training set, compared to the training set in i, the performance of KNN and logistic regression has significantly decreased. Maybe these classifiers are not suitable for doing this For predicting the distribution of data, if you still want to choose one, you still choose KNN. Its performance is slightly better than logistic regression and baseline classifiers.

Appendix

code

ia iia identify C:

1. **import** numpy as np
2. **from** sklearn.linear\_model **import** LogisticRegression
3. **from** sklearn.model\_selection **import** cross\_val\_score
4. **from** sklearn.preprocessing **import** PolynomialFeatures
5. **from** matplotlib.transforms **import** Affine2D
6. **import** matplotlib.pyplot as plt
8. # Get data1 from file
9. data = str(open("data2").read()).split('\n')
10. dataset = []
11. **for** d **in** data[1:]:
12. nums = d.split(',')
13. dxy = []
14. dxy.append(float(nums[0]))
15. dxy.append(float(nums[1]))
16. dxy.append(float(nums[2]))
17. dataset.append(dxy)
18. dataset = np.array(dataset)
20. train = dataset.take([0, 1], axis=1)
21. target = dataset[:, 2]
23. C = 0.001
24. # C = 0.1
25. # C = 1
26. # C = 10
27. # C = 1000
28. Cs, acc\_means, acc\_stds, f1\_means, f1\_stds = [], [], [], [], []
29. **while** C < 1.05:
30. poly = PolynomialFeatures(degree=1)
31. poly\_train = poly.fit\_transform(train)
32. **print**(poly.get\_feature\_names())
33. LR = LogisticRegression(penalty='l2', C=C, max\_iter=10000, tol=0.0001)
34. scores = cross\_val\_score(LR, poly\_train, target, cv=10)
35. # f1\_score = cross\_val\_score(LR, poly\_train, target, scoring='f1', cv=10)
36. # print('C=' + str(C) + ' acc=' + str(scores.mean()) + ' std=' + str(scores.std()))
37. Cs.append(C)
38. acc\_means.append(scores.mean())
39. acc\_stds.append(scores.std())
40. # f1\_means.append(f1\_score.mean())
41. # f1\_stds.append(f1\_score.std())
42. C = C + 0.05
44. plt.figure()
45. fig, ax = plt.subplots(figsize=(12, 5))
46. plt.title('Logistic Regression error bar with C between 0.001 and 1')
47. plt.xlabel('C')
48. plt.ylabel('mean accuracy')
49. # trans1 = Affine2D().translate(-5, 0.0) + ax.transData
50. # trans2 = Affine2D().translate(0.0, 0.0) + ax.transData
51. plt.errorbar(Cs, acc\_means, acc\_stds, mfc='red', color="r",  label='accuracy', alpha=0.5)
52. # plt.errorbar(Cs, f1\_means, f1\_stds, mfc='blue', color="b", transform=trans2, label='f1\_score', alpha=0.5)
53. plt.errorbar(Cs, acc\_means, acc\_stds, mfc='blue', color="b",
54. label='The vertical line is the standard deviation', alpha=0.0)
55. plt.legend(loc=4)
56. plt.show()

ia iia identify degree:

1. **import** numpy as np
2. **from** sklearn.linear\_model **import** LogisticRegression
3. **from** sklearn.preprocessing **import** PolynomialFeatures, StandardScaler
4. **from** matplotlib.colors **import** ListedColormap
5. **import** matplotlib.pyplot as plt



10. # Get data1 from file
11. data = str(open("data2").read()).split('\n')
12. dataset = []
13. px, py, nx, ny = [], [], [], []
14. **for** d **in** data[1:]:
15. nums = d.split(',')
16. dxy = []
17. dxy.append(float(nums[0]))
18. dxy.append(float(nums[1]))
19. dxy.append(float(nums[2]))
20. dataset.append(dxy)
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. dataset = np.array(dataset)
30. train = dataset.take([0, 1], axis=1)
31. target = dataset[:, 2]
33. degree = 1
34. poly\_train = PolynomialFeatures(degree=degree).fit\_transform(train)
35. LR = LogisticRegression(penalty='l2', C=1)
36. LR.fit(poly\_train,target)
37. fig, ax = plt.subplots(figsize=(7, 7))

40. axis = [-1, 1, -1, 1]
41. x0, x1 = np.meshgrid(
42. np.linspace(axis[0], axis[1], int((axis[1] - axis[0]) \* 100)).reshape(-1, 1),
43. np.linspace(axis[2], axis[3], int((axis[3] - axis[2]) \* 100)).reshape(-1, 1),
44. )
45. X\_new = np.c\_[x0.ravel(), x1.ravel()]
46. X\_new = PolynomialFeatures(degree=degree).fit\_transform(X\_new)
47. y\_predict = LR.predict(X\_new)
48. zz = y\_predict.reshape(x0.shape)
50. custom\_cmap = ListedColormap(['#EF9A9A', '#FFF59D', '#90CAF9'])
51. ax.contourf(x0, x1, zz, linewidth=1, cmap=custom\_cmap)
52. # plt.contour(x0, x1, zz, [0])
53. ax.scatter(px, py, s=10, c='b', marker='o', label='1')
54. ax.scatter(nx, ny, s=10, c='r', marker='x', label='-1')
55. ax.legend()
56. plt.title('Logistic Regression with C=' + str(degree))
57. ax.set\_xlabel("x-axis")
58. ax.set\_ylabel("y-axis")
59. plt.show()

ib iib cross validation for k:

1. **import** numpy as np
2. **from** sklearn.model\_selection **import** cross\_val\_score
3. **from** sklearn.neighbors **import** KNeighborsClassifier
4. **import** matplotlib.pyplot as plt
6. data = str(open("data2").read()).split('\n')
7. dataset = []
8. **for** d **in** data[1:]:
9. nums = d.split(',')
10. dxy = []
11. dxy.append(float(nums[0]))
12. dxy.append(float(nums[1]))
13. dxy.append(float(nums[2]))
14. dataset.append(dxy)
15. dataset = np.array(dataset)
17. train = dataset.take([0, 1], axis=1)
18. target = dataset[:, 2]
20. Ks, acc\_means, acc\_stds, f1\_means, f1\_stds = [], [], [], [], []
21. **for** K **in** range(1,30):
22. knn = KNeighborsClassifier(n\_neighbors=K)
23. scores = cross\_val\_score(knn, train, target, cv=10)
24. Ks.append(K)
25. acc\_means.append(scores.mean())
26. acc\_stds.append(scores.std())
28. plt.figure()
29. fig, ax = plt.subplots(figsize=(12, 5))
30. ax.errorbar(Ks, acc\_means, acc\_stds, mfc='red', color="r", label='accuracy', alpha=0.5)
31. plt.errorbar(Ks, acc\_means, acc\_stds, mfc='blue', color="b",
32. label='The vertical line is the standard deviation', alpha=0.0)
33. plt.title('KNN error bar with different K' )
34. plt.xlabel('K')
35. plt.ylabel('mean accuracy')
36. plt.legend(loc=4)
37. plt.show()

ib iib draw scatter plot:

1. **import** numpy as np
2. **from** matplotlib.colors **import** ListedColormap
3. **from** sklearn.neighbors **import** KNeighborsClassifier
4. **import** matplotlib.pyplot as plt
6. data = str(open("data2").read()).split('\n')
7. dataset = []
8. px, py, nx, ny = [], [], [], []
9. **for** d **in** data[1:]:
10. nums = d.split(',')
11. dxy = []
12. dxy.append(float(nums[0]))
13. dxy.append(float(nums[1]))
14. dxy.append(float(nums[2]))
15. dataset.append(dxy)
16. **if** nums[2] == '1':
17. px.append(float(nums[0]))
18. py.append(float(nums[1]))
19. **else**:
20. nx.append(float(nums[0]))
21. ny.append(float(nums[1]))
22. dataset = np.array(dataset)
24. train = dataset.take([0, 1], axis=1)
25. target = dataset[:, 2]

28. K=13
29. knn = KNeighborsClassifier(n\_neighbors=K)
30. knn.fit(train,target)
32. fig, ax = plt.subplots(figsize=(7, 7))
33. xp = np.linspace(-1, 1, 300)
34. yp = np.linspace(-1, 1, 300)
35. x1, y1 = np.meshgrid(xp, yp)
36. xy = np.c\_[x1.ravel(), y1.ravel()]
37. y\_pred = knn.predict(xy).reshape(x1.shape)
38. custom\_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
39. ax.contourf(x1, y1, y\_pred, alpha=0.3, cmap=custom\_cmap)
40. ax.scatter(px, py, s=10, c='b', marker='o', label='1')
41. ax.scatter(nx, ny, s=10, c='r', marker='x', label='-1')
42. plt.title('KNN Decision boundary with K=' +str(K))
43. ax.legend(loc=1)
44. ax.set\_xlabel("x-axis")
45. ax.set\_ylabel("y-axis")
46. plt.show()

ic-iic confusion matrix

1. **import** numpy as np
2. **from** sklearn.dummy **import** DummyClassifier
3. **from** sklearn.linear\_model **import** LogisticRegression
4. **from** sklearn.metrics **import** confusion\_matrix
5. **from** sklearn.model\_selection **import** train\_test\_split
6. **from** sklearn.neighbors **import** KNeighborsClassifier
7. **from** sklearn.preprocessing **import** PolynomialFeatures
9. data = str(open("data2").read()).split('\n')
10. dataset = []
11. px, py, nx, ny = [], [], [], []
12. **for** d **in** data[1:]:
13. nums = d.split(',')
14. dxy = []
15. dxy.append(float(nums[0]))
16. dxy.append(float(nums[1]))
17. dxy.append(float(nums[2]))
18. dataset.append(dxy)
19. **if** nums[2] == '1':
20. px.append(float(nums[0]))
21. py.append(float(nums[1]))
22. **else**:
23. nx.append(float(nums[0]))
24. ny.append(float(nums[1]))
25. dataset = np.array(dataset)
27. train = dataset.take([0, 1], axis=1)
28. target = dataset[:, 2]
30. X\_train, X\_test, y\_train, y\_test = train\_test\_split(train, target)
31. knn = KNeighborsClassifier(n\_neighbors=13)
32. knn.fit(X\_train, y\_train)
33. pred = knn.predict(X\_test)
34. knn\_m = confusion\_matrix(y\_test, pred)
35. **print**('confusion matrix：', knn\_m, sep='\n')
37. poly\_features = PolynomialFeatures(degree=1)
38. poly\_train = poly\_features.fit\_transform(X\_train)
39. LR = LogisticRegression(penalty='l2', C=1)
40. LR.fit(poly\_train, y\_train)
41. poly\_test = poly\_features.fit\_transform(X\_test)
42. poly\_pred = LR.predict(poly\_test)
43. lr\_m = confusion\_matrix(y\_test, poly\_pred)
44. **print**('confusion matrix：', lr\_m, sep='\n')
46. dummy = DummyClassifier(strategy="uniform").fit(X\_train, y\_train)
47. dummy\_pred = dummy.predict(X\_test)
48. lr\_m = confusion\_matrix(y\_test, dummy\_pred)
49. **print**('confusion matrix：', lr\_m, sep='\n')

id iid draw roc curve:

1. **import** numpy as np
2. **from** sklearn.dummy **import** DummyClassifier
3. **from** sklearn.linear\_model **import** LogisticRegression
4. **from** sklearn.metrics **import** confusion\_matrix, roc\_auc\_score, roc\_curve
5. **from** sklearn.model\_selection **import** train\_test\_split
6. **from** sklearn.neighbors **import** KNeighborsClassifier
7. **from** sklearn.preprocessing **import** PolynomialFeatures
8. **import** matplotlib.pyplot as plt
10. data = str(open("data2").read()).split('\n')
11. dataset = []
12. px, py, nx, ny = [], [], [], []
13. **for** d **in** data[1:]:
14. nums = d.split(',')
15. dxy = []
16. dxy.append(float(nums[0]))
17. dxy.append(float(nums[1]))
18. dxy.append(float(nums[2]))
19. dataset.append(dxy)
20. **if** nums[2] == '1':
21. px.append(float(nums[0]))
22. py.append(float(nums[1]))
23. **else**:
24. nx.append(float(nums[0]))
25. ny.append(float(nums[1]))
26. dataset = np.array(dataset)
28. train = dataset.take([0, 1], axis=1)
29. target = dataset[:, 2]
31. X\_train, X\_test, y\_train, y\_test = train\_test\_split(train, target)
33. poly\_features = PolynomialFeatures(degree=1)
34. poly\_train = poly\_features.fit\_transform(X\_train)
35. LR = LogisticRegression(penalty='l2', C=1)
36. LR.fit(poly\_train, y\_train)
38. knn = KNeighborsClassifier(n\_neighbors=13)
39. knn.fit(X\_train, y\_train)
41. dummy = DummyClassifier(strategy="uniform").fit(X\_train, y\_train)
42. # dummy\_pred = dummy.predict(X\_test)

45. logit\_roc\_auc = roc\_auc\_score(y\_test, LR.predict(poly\_features.fit\_transform(X\_test)))
46. y\_prob = LR.predict\_proba(poly\_features.fit\_transform(X\_test))[:, 1]
47. fpr\_lr, tpr\_lr, threshold\_lr = roc\_curve(y\_test, y\_prob)

50. dummy\_roc\_auc = roc\_auc\_score(y\_test, dummy.predict(X\_test))
51. y\_prob\_dummy = dummy.predict\_proba(X\_test)[:, 1]
52. fpr\_lr\_dummy, tpr\_lr\_dummy, threshold\_lr\_dummy = roc\_curve(y\_test, y\_prob\_dummy)

55. knn\_roc\_auc = roc\_auc\_score(y\_test, knn.predict(X\_test))
56. y\_prob\_knn = knn.predict\_proba(X\_test)[:, 1]
57. **print**(y\_prob\_knn)
58. fpr\_lr\_knn, tpr\_lr\_knn, threshold\_lr\_knn = roc\_curve(y\_test, y\_prob\_knn)
60. plt.figure()
61. fig,ax = plt.subplots(figsize=(7,7))
62. ax.plot(fpr\_lr, tpr\_lr, label='Logistic Regression (area = %0.2f)' % logit\_roc\_auc)
63. ax.plot(fpr\_lr\_dummy, tpr\_lr\_dummy, label='Dummy Classifier (area = %0.2f)' % dummy\_roc\_auc)
64. ax.plot(fpr\_lr\_knn, tpr\_lr\_knn, label='knn (area = %0.2f)' % knn\_roc\_auc)
65. # ax.plot([0, 1], [0, 1], 'r--')
66. plt.xlim([0.0, 1.0])
67. plt.ylim([0.0, 1.05])
68. plt.xlabel('False Positive Rate')
69. plt.ylabel('True Positive Rate')
70. plt.title('Receiver operating characteristic')
71. plt.legend(loc="lower right")
72. plt.show()