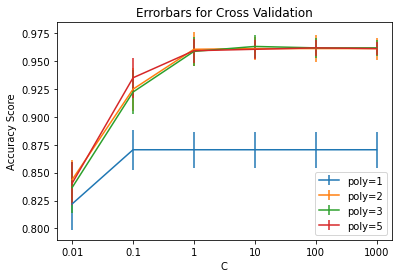
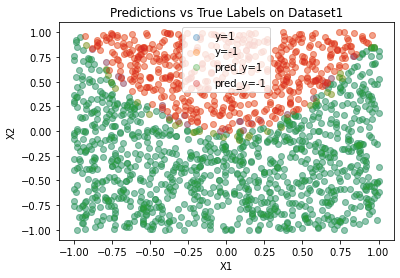
Week4 homework

(i).

(a). Accuracy and precision are metrics for evaluating a classification model. In this task, we use accuracy as the measuring metric, as accuracy considers the overall result including both positive and negative classes. We are choosing the hyperparameter C from a list [0.01, 0.1, 1, 10, 100, 1000], and the maximum order of polynomials from [1, 2, 3, 5]. The best classifier shall have a high mean value of accuracy, while the standard deviation shall not be large. A small standard deviation indicates the model is robust on different datasets. For reproduction, we set all the random states to 0. With these analyses, we started the experiment and generated a cross-validation plot as below:



As the plot shows, the mean value of the accuracy score has little difference between different maximum orders of polynomials selected when C is larger than 1. When observed with care, we can find out that the minimum standard deviation is reached by the model with poly=5 and C=100. Therefore, this model shall be selected as the final model for classification, and we can plot the prediction results as:

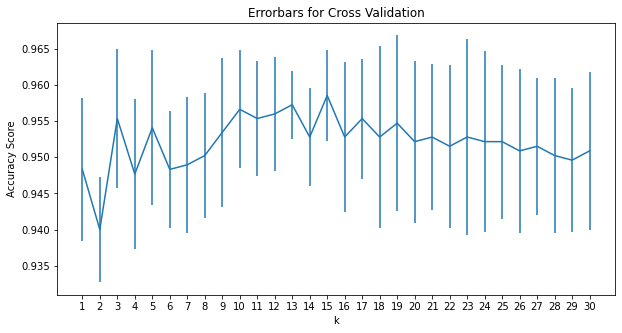


The predictions and true labels are highly overlapped, indicating the model fits the dataset very well.

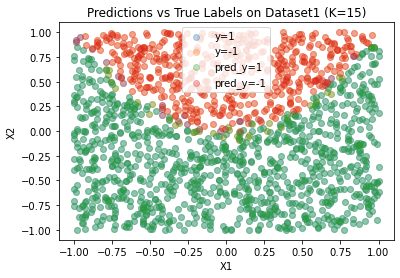
Code:

1. import sklearn
2. import matplotlib.pyplot as plt
3. import numpy as np
4. from sklearn.linear\_model import LogisticRegression
5. from sklearn.preprocessing import PolynomialFeatures
6. from sklearn.model\_selection import KFold
7. from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve
8. from sklearn.neighbors import KNeighborsClassifier
9. *# Question 1* *Dataset 1 id:5--10--5-1 ; Dataset 2 id:5-5--5-1*
10. *# Load the dataset*
11. file = open("week4.txt", "r+")
12. dataset1, dataset2 = [], []
13. *# Which dataset to write in*
14. flag = False
15. for line in file.readlines():
16. if line.strip() == "# id:5--10--5-1":
17. continue
19. *# Switch the flag*
20. if line.strip() == "# id:5-5--5-1":
21. flag = True
22. continue
24. *# Capture the samples*
25. sample = np.array(line.strip().split(",")).astype(float)
26. if flag:
27. dataset1.append(sample)
28. else:
29. dataset2.append(sample)
30. *# Convert to np type and split X y*
31. dataset1 = np.array(dataset1); dataset2 = np.array(dataset2)
32. X\_1, y\_1= dataset1[:, :2], dataset1[:, 2]
33. index\_pos\_1 = [i for i in range(len(y\_1)) if y\_1[i] == 1]; index\_neg\_1 = [i for i in range(len(y\_1)) if y\_1[i] == -1]
34. X\_2, y\_2 = dataset2[:, :2], dataset2[:, 2]
35. index\_pos\_2 = [i for i in range(len(y\_2)) if y\_2[i] == 1]; index\_neg\_2 = [i for i in range(len(y\_2)) if y\_2[i] == -1]
36. *# A search grid*
37. c\_list = [0.01, 0.1, 1, 10, 100, 1000]
38. poly\_list = [1, 2, 3, 5]
39. score\_mean, score\_std = [], []
40. for poly in poly\_list:
41. *# Create new features*
42. poly\_X\_1 = PolynomialFeatures(poly).fit\_transform(X\_1)
43. means, stds = [], []
44. for c in c\_list:
45. clf = LogisticRegression(random\_state=0, penalty='l2', C=c, max\_iter=1000)
46. scores = []
47. *# Kfold validation*
48. for train, test in KFold(5).split(poly\_X\_1):
49. clf.fit(poly\_X\_1[train], y\_1[train])
50. ypred = clf.predict(poly\_X\_1[test])
51. scores.append(accuracy\_score(y\_1[test], ypred))
52. *# Fetch the mean and std of the scores*
53. means.append(np.array(scores).mean()), stds.append(np.array(scores).std())
54. score\_mean.append(means), score\_std.append(stds)
55. *# Plot the errorbar*
56. plt.figure()
57. plt.title("Errorbars for Cross Validation")
58. for i in range(len(poly\_list)):
59. plt.errorbar(np.arange(len(c\_list)), score\_mean[i], score\_std[i], label="poly=%d"%poly\_list[i])
60. plt.legend()
61. plt.xticks(np.arange(len(c\_list)), c\_list)
62. plt.xlabel("C"), plt.ylabel("Accuracy Score")
63. plt.show()
64. *# We select C=100 and poly=5 for the final model*
65. poly\_X\_1 = PolynomialFeatures(5).fit\_transform(X\_1)
66. clf =  LogisticRegression(random\_state=0, penalty='l2', C=100, max\_iter=1000)
67. clf.fit(poly\_X\_1, y\_1)
68. y\_pred = clf.predict(poly\_X\_1)
69. index\_pos\_pred = [i for i in range(len(y\_pred)) if y\_pred[i] == 1]; index\_neg\_pred = [i for i in range(len(y\_pred)) if y\_pred[i] == -1]
71. *# Plot the first dataset*
72. plt.figure()
73. plt.title("Predictions vs True Labels on Dataset1")
74. plt.scatter(X\_1[index\_pos\_1][:, 0], X\_1[index\_pos\_1][:, 1], label="y=1", alpha=0.3)
75. plt.scatter(X\_1[index\_neg\_1][:, 0], X\_1[index\_neg\_1][:, 1], label='y=-1', alpha=0.3)
76. plt.scatter(X\_1[index\_pos\_pred][:, 0], X\_1[index\_pos\_pred][:, 1], label="pred\_y=1", alpha=0.3)
77. plt.scatter(X\_1[index\_neg\_pred][:, 0], X\_1[index\_neg\_pred][:, 1], label='pred\_y=-1', alpha=0.3)
78. plt.xlabel("X1"), plt.ylabel("X2"), plt.legend()
79. plt.show()

(b). In this problem, we are choosing a value k from 1 to 15. Similar to the last problem, we are seeking a model with a high mean accuracy and low standard deviation. By analyzing the graph, we can assert that 15 shall be the final value for k.



The prediction result is attached below. The fit is also satisfying. From the experiment, we know that the number of neighbors shall not be too small or too large, otherwise the model will overfit on the closest sample, or randomly fit on the surrounding points, causing underfit.



Code:

1. k\_list = np.arange(30)+1
2. score\_mean, score\_std = [], []
3. for k in k\_list:
4. *# Create classifiers*
5. scores = []
6. knn = KNeighborsClassifier(n\_neighbors=k)
7. for train, test in KFold(5).split(X\_1):
8. knn.fit(X\_1[train], y\_1[train])
9. ypred = knn.predict(X\_1[test])
10. scores.append(accuracy\_score(y\_1[test], ypred))
11. *# Fetch the mean and std of the scores*
12. score\_mean.append(np.array(scores).mean()), score\_std.append(np.array(scores).std())
13. *# Plot the errorbar*
14. plt.figure(figsize=[10, 5])
15. plt.title("Errorbars for Cross Validation")
16. plt.errorbar(np.arange(len(k\_list)), score\_mean, score\_std)
17. plt.xticks(np.arange(len(k\_list)), k\_list)
18. plt.xlabel("k"); plt.ylabel("Accuracy Score")
19. plt.show()
20. *# We select K=15 for the final model*
21. knn =  KNeighborsClassifier(n\_neighbors=15).fit(X\_1, y\_1)
22. y\_pred = knn.predict(X\_1)
23. index\_pos\_pred = [i for i in range(len(y\_pred)) if y\_pred[i] == 1]; index\_neg\_pred = [i for i in range(len(y\_pred)) if y\_pred[i] == -1]
25. *# Plot the first dataset*
26. plt.figure()
27. plt.title("Predictions vs True Labels on Dataset1 (K=15)")
28. plt.scatter(X\_1[index\_pos\_1][:, 0], X\_1[index\_pos\_1][:, 1], label="y=1", alpha=0.3)
29. plt.scatter(X\_1[index\_neg\_1][:, 0], X\_1[index\_neg\_1][:, 1], label='y=-1', alpha=0.3)
30. plt.scatter(X\_1[index\_pos\_pred][:, 0], X\_1[index\_pos\_pred][:, 1], label="pred\_y=1", alpha=0.3)
31. plt.scatter(X\_1[index\_neg\_pred][:, 0], X\_1[index\_neg\_pred][:, 1], label='pred\_y=-1', alpha=0.3)
32. plt.xlabel("X1"), plt.ylabel("X2"), plt.legend()
33. plt.show()

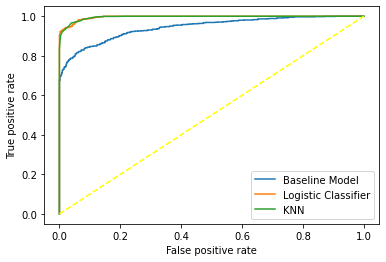
(c). We established a standard logistic classifier as the baseline model. The model is trained with the basic features and the hyperparameters are not tuned except for the random state.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Pred1 | Pred-1 | Logistic | Pred1 | Pred-1 | KNN | Pred1 | Pred-1 |
| True1 | 410 | 108 | True1 | 486 | 32 | True1 | 483 | 35 |
| True-1 | 96 | 953 | True-1 | 25 | 1024 | True-1 | 25 | 1024 |

Code:

1. *# For comparison, a standard logistic classifier is created with the basic features*
2. baseline = LogisticRegression(random\_state=0)
3. baseline.fit(X\_1, y\_1)
4. ypred\_baseline = baseline.predict(X\_1)
5. print("Confusion matrix for the baseline model: ")
6. print(confusion\_matrix(y\_1, ypred\_baseline))
7. *# The other two models*
8. poly\_X\_1 = PolynomialFeatures(5).fit\_transform(X\_1)
9. poly =  LogisticRegression(random\_state=0, penalty='l2', C=100, max\_iter=1000)
10. poly.fit(poly\_X\_1, y\_1)
11. y\_pred\_logistic = poly.predict(poly\_X\_1)
12. print("Confusion matrix for the logistic regression model: ")
13. print(confusion\_matrix(y\_1, y\_pred\_logistic))
14. knn =  KNeighborsClassifier(n\_neighbors=15).fit(X\_1, y\_1)
15. y\_pred\_knn = knn.predict(X\_1)
16. print("Confusion matrix for the KNN model: ")
17. print(confusion\_matrix(y\_1, y\_pred\_knn))

(d). The following plot visualizes the ROC curves of the three models.



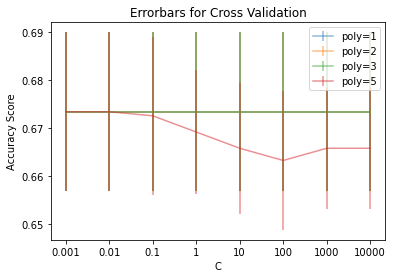
Code:

1. *# Curves of the models*
2. fpr, tpr, \_ = roc\_curve(y\_1, baseline.decision\_function(X\_1))
3. plt.plot(fpr, tpr, label="Baseline Model")
4. fpr, tpr, \_ = roc\_curve(y\_1, poly.decision\_function(poly\_X\_1))
5. plt.plot(fpr, tpr, label="Logistic Classifier")
6. *# For KNN, we shall use the probabilities instead*
7. fpr, tpr, \_ = roc\_curve(y\_1, knn.predict\_proba(X\_1)[:, 1])
8. plt.plot(fpr, tpr, label="KNN")
9. plt.xlabel("False positive rate")
10. plt.ylabel("True positive rate")
11. plt.legend()
12. plt.plot([0, 1], [0, 1], color="yellow",linestyle="--")
13. plt.show()

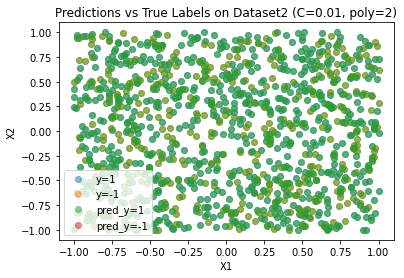
(e). A confusion matrix can illustrate the number of samples wrongly classified, and people can also learn about similar classes that are difficult to separate. In the table attached in problem (c), we show the confusion matrices. We can see the two models have more correct predictions than the baseline model, while the logistic classifier is slightly more accurate than the KNN model. When evaluating with ROC curves, we know that a model is more ideal when it has a curve closer to the position [0, 1], the point indicating 0% false positive with 100% true positive. It’s shown in the graph that both logistic and KNN models are closer to the point than the baseline model is. Moreover, KNN generated a curve that is closest to the ideal point. There is no significant difference between the two models, and I would suggest using a logistic classifier as it is slightly better than the KNN model.

(ii). The code used in this question is similar to those applied in the last problem, thus the codes for this question are not attached in the report.

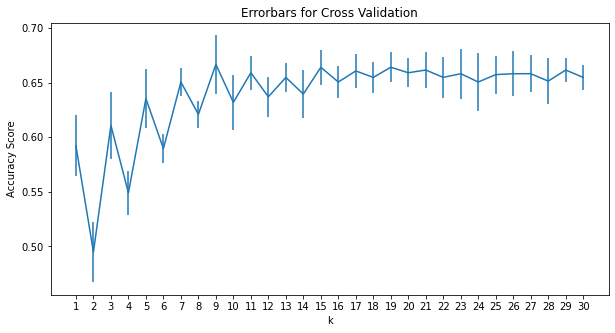
(a). When implementing a logistic classifier, we kept the search grid and generated a plot for cross-validation.

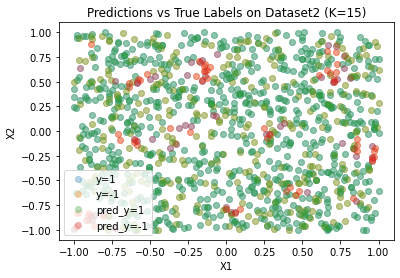


Although all the accuracies are low, we can still observe an accuracy drop when increasing the maximum order of polynomials, suggesting the existence of overfitting. The accuracy also suffers a decrease when the value of C rises. Therefore, we chose C=0.01 and poly=2 for the final logistic classifier. The prediction results are shown in the following plot, and it can be analyzed that this dataset is too complex to be classified by a logistic classifier.



(b). A significant rise in the mean accuracy is shown when selecting the number of neighbors for the KNN model. Using the following plot as an indication, we also select K=15 for the final model.



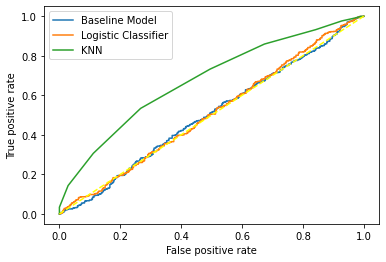


There lie differences between the KNN model and the logistic classifier, but it is hard to tell which model can perform better, as they both have low mean accuracies.

(c). The baseline model is also the logistic classifier with no hyperparameters tuned.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Pred1 | Pred-1 | Logistic | Pred1 | Pred-1 | KNN | Pred1 | Pred-1 |
| True1 | 0 | 386 | True1 | 0 | 386 | True1 | 61 | 325 |
| True-1 | 0 | 796 | True-1 | 0 | 796 | True-1 | 54 | 742 |

(d). The following plot visualizes the ROC curves of the three models.



(e). Although we cannot draw a conclusion about which model to use from the first two problems, the table from (c) and the plot from (d) can aid us to make a decision. The baseline model and the logistic classifier are making random predictions by classifying all the samples into the negative class. The ROC curves of the two models also support our discovery. On the other hand, the KNN model generates a curve closer to the ideal point, and several positive samples are correctly classified. In conclusion, the KNN model is more capable for this dataset.