

Question1

```
1 import time
2 from sklearn.svm import SVC
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.ensemble import AdaBoostClassifier
5 from sklearn.ensemble import RandomForestClassifier
6 from sklearn.tree import DecisionTreeClassifier
7 from sklearn.neural_network import MLPClassifier
8 from sklearn.model_selection import train_test_split
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.datasets import make_classification
```

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.colors import ListedColormap
4 import warnings
5
6 warnings.simplefilter(action='ignore', category=FutureWarning)
7 warnings.filterwarnings("ignore")
```

```
1 def create_dataset(n=1250, nf=2, nr=0, ni=2, random_state=125):
2     '''
3     generate a new dataset with
4     n: total number of samples
5     nf: number of features
6     nr: number of redundant features (these are linear combinatins of
7     informative features)
8     ni: number of informative features (ni + nr = nf must hold)
9     random_state: set for reproducibility
10    '''
11    X, y = make_classification(n_samples=n,
12                              n_features=nf,
13                              n_redundant=nr,
14                              n_informative=ni,
15                              random_state=random_state,
16                              n_clusters_per_class=2)
17    rng = np.random.RandomState(2)
18    X += 3 * rng.uniform(size=X.shape)
19    X = StandardScaler().fit_transform(X)
20    return X, y
```

```
1 def plotter(classifier, X, X_test, y_test, title, ax=None):
2     # plot decision boundary for given classifier
```

```

3     plot_step = 0.02
4     x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
5     y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
6     xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
7                           np.arange(y_min, y_max, plot_step))
8     Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
9     Z = Z.reshape(xx.shape)
10    if ax:
11        ax.contourf(xx, yy, Z, cmap=plt.cm.Paired)
12        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
13        ax.set_title(title)
14    else:
15        plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
16        plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
17        plt.title(title)

```

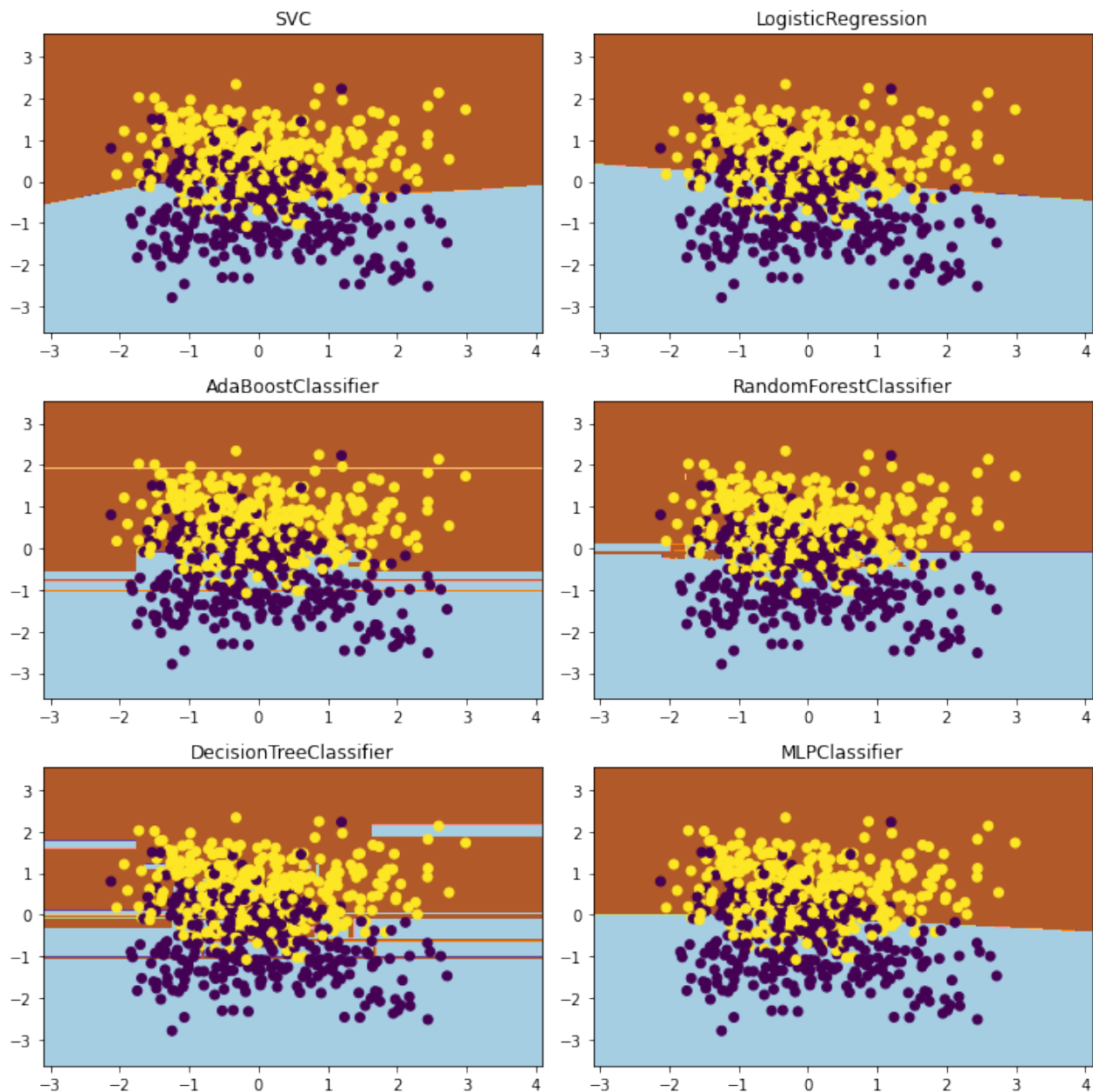
(a)

```

1     # data set
2     data, target = create_dataset()
3
4     # split
5     X_train, X_test, Y_train, Y_test = train_test_split(data, target,
6                                                         train_size=0.5, test_size=0.5, random_state=15)
7
8     # classifier
9     classified_list = []
10
11    SVCModel = SVC()
12    SVCModel.fit(X_train, Y_train)
13    classified_list.append(SVCModel)
14
15    LogisticRegressionModel = LogisticRegression()
16    LogisticRegressionModel.fit(X_train, Y_train)
17    classified_list.append(LogisticRegressionModel)
18
19    AdaBoostClassifierModel = AdaBoostClassifier()
20    AdaBoostClassifierModel.fit(X_train, Y_train)
21    classified_list.append(AdaBoostClassifierModel)
22
23    RandomForestClassifierModel = RandomForestClassifier()
24    RandomForestClassifierModel.fit(X_train, Y_train)
25    classified_list.append(RandomForestClassifierModel)
26
27    DecisionTreeClassifierModel = DecisionTreeClassifier()
28    DecisionTreeClassifierModel.fit(X_train, Y_train)
29    classified_list.append(DecisionTreeClassifierModel)

```

```
30 MLPClassifierModel = MLPClassifier()
31 MLPClassifierModel.fit(X_train, Y_train)
32 classified_list.append(MLPClassifierModel)
33
34 classified_name_list = ['SVC', 'LogisticRegression', 'AdaBoostClassifier',
35                        'RandomForestClassifier',
36                        'DecisionTreeClassifier', 'MLPClassifier']
37
38 fig, ax = plt.subplots(3, 2, figsize=(10, 10))
39 for i, ax in enumerate(ax.flat):
40     plotter(classifier=classified_list[i], X=X_train, X_test=X_test,
41             y_test=Y_test, title=classified_name_list[i],
42             ax=ax)
41 plt.tight_layout()
42 plt.show()
```



(b)

```

1  # data set
2  data, target = create_dataset()
3
4  X_train, X_test, Y_train, Y_test = train_test_split(data, target,
5  train_size=0.8, test_size=0.2, random_state=45)
6  x_y_set = np.hstack((X_train, Y_train.reshape(1000, 1)))
7
8  size_list = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
9
10 # np.random.shuffle(x_y_set)
11
12 # Decision Tree

```

```

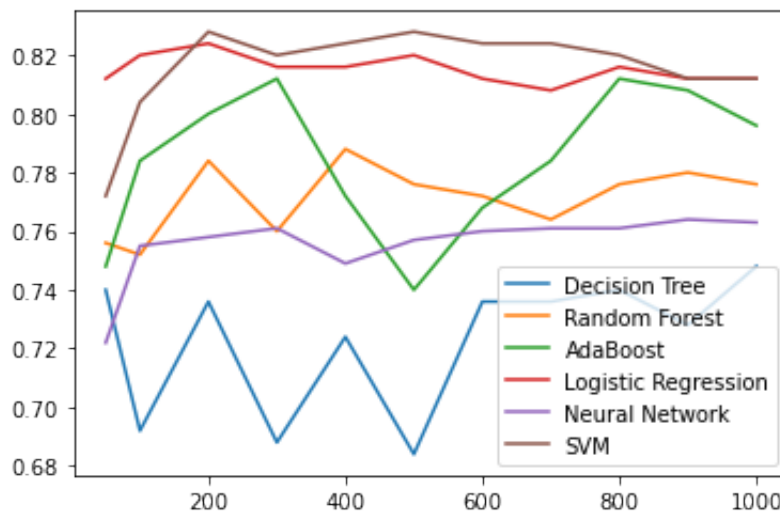
12 DecisionTreeClassifierModel = DecisionTreeClassifier()
13 decision_tree_list = []
14 for i in range(len(size_list)):
15     np.random.shuffle(x_y_set)
16     DecisionTreeClassifierModel.fit(x_y_set[:size_list[i], :2],
x_y_set[:size_list[i], 2])
17     decision_tree_list.append(DecisionTreeClassifierModel.score(X_test,
Y_test))
18
19 # Random Forest
20 RandomForestClassifierModel = RandomForestClassifier()
21 random_forest_list = []
22 for i in range(len(size_list)):
23     np.random.shuffle(x_y_set)
24     RandomForestClassifierModel.fit(x_y_set[:size_list[i], :2],
x_y_set[:size_list[i], 2])
25     random_forest_list.append(RandomForestClassifierModel.score(X_test,
Y_test))
26
27 # AdaBoost
28 AdaBoostClassifierModel = AdaBoostClassifier()
29 ada_boost_list = []
30 for i in range(len(size_list)):
31     np.random.shuffle(x_y_set)
32     AdaBoostClassifierModel.fit(x_y_set[:size_list[i], :2],
x_y_set[:size_list[i], 2])
33     ada_boost_list.append(AdaBoostClassifierModel.score(X_test, Y_test))
34
35 # LogisticRegression
36 LogisticRegressionModel = LogisticRegression()
37 logistic_regression_list = []
38 for i in range(len(size_list)):
39     np.random.shuffle(x_y_set)
40     LogisticRegressionModel.fit(x_y_set[:size_list[i], :2],
x_y_set[:size_list[i], 2])
41     logistic_regression_list.append(LogisticRegressionModel.score(X_test,
Y_test))
42
43 # MLPClassifier
44 MLPClassifierModel = MLPClassifier()
45 neural_network_list = []
46 for i in range(len(size_list)):
47     np.random.shuffle(x_y_set)
48     MLPClassifierModel.fit(x_y_set[:size_list[i], :2],
x_y_set[:size_list[i], 2])
49     neural_network_list.append(MLPClassifierModel.score(X_train, Y_train))
50
51 # SVC
52 SVCModel = SVC()

```

```

53 svc_list = []
54 for i in range(len(size_list)):
55     # np.random.shuffle(x_y_set)
56     SVCModel.fit(x_y_set[:size_list[i], :2], x_y_set[:size_list[i], 2])
57     svc_list.append(SVCModel.score(X_test, Y_test))
58
59 plt.plot(size_list, decision_tree_list, label='Decision Tree')
60 plt.plot(size_list, random_forest_list, label='Random Forest')
61 plt.plot(size_list, ada_boost_list, label='AdaBoost')
62 plt.plot(size_list, logistic_regression_list, label='Logistic Regression')
63 plt.plot(size_list, neural_network_list, label='Neural Network')
64 plt.plot(size_list, svc_list, label='SVM')
65 plt.legend()
66 plt.show()

```



From the graph, i will choose Logistic Regression model because it has higher accuracy and more stable than others.

(c)

```

1  # data set
2  data, target = create_dataset()
3
4  # split
5  # X_train, X_test = train_test_split(data, train_size=0.8, test_size=0.2,
6  # Y_train, Y_test = train_test_split(target, train_size=0.8,
7  #                                     test_size=0.2, random_state=45)
8
9  X_train, X_test, Y_train, Y_test = train_test_split(data, target,
10                                                         train_size=0.8, test_size=0.2, random_state=45)
11 # x_y_set = np.array()

```

```

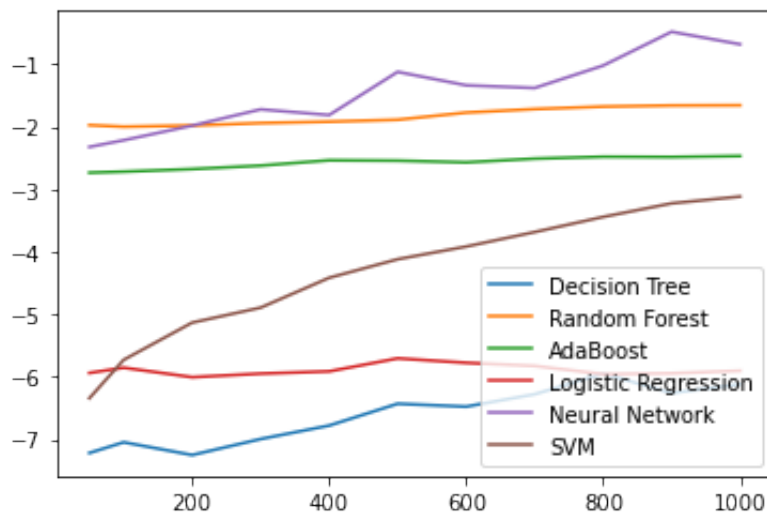
10
11 size_list = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
12
13 # Decision Tree
14 DecisionTreeClassifierModel = DecisionTreeClassifier()
15 decision_tree_list = []
16 dt_time_list = []
17 for i in range(len(size_list)):
18     start = time.time()
19     DecisionTreeClassifierModel.fit(X_train[:size_list[i]],
20     Y_train[:size_list[i]])
21     decision_tree_list.append(DecisionTreeClassifierModel.score(X_test,
22     Y_test))
23     end = time.time()
24     dt_time_list.append(np.log(end - start))
25
26 # Random Forest
27 RandomForestClassifierModel = RandomForestClassifier()
28 random_forest_list = []
29 rf_time_list = []
30 for i in range(len(size_list)):
31     start = time.time()
32     RandomForestClassifierModel.fit(X_train[:size_list[i]],
33     Y_train[:size_list[i]])
34     random_forest_list.append(RandomForestClassifierModel.score(X_test,
35     Y_test))
36     end = time.time()
37     rf_time_list.append(np.log(end - start))
38
39 # AdaBoost
40 AdaBoostClassifierModel = AdaBoostClassifier()
41 ada_boost_list = []
42 ab_time_list = []
43 for i in range(len(size_list)):
44     start = time.time()
45     AdaBoostClassifierModel.fit(X_train[:size_list[i]],
46     Y_train[:size_list[i]])
47     ada_boost_list.append(AdaBoostClassifierModel.score(X_test, Y_test))
48     end = time.time()
49     ab_time_list.append(np.log(end - start))
50
51 # LogisticRegression
52 LogisticRegressionModel = LogisticRegression()
53 logistic_regression_list = []
54 lr_time_list = []
55 for i in range(len(size_list)):
56     start = time.time()
57     LogisticRegressionModel.fit(X_train[:size_list[i]],
58     Y_train[:size_list[i]])

```

```

53     logistic_regression_list.append(LogisticRegressionModel.score(X_test,
54                             Y_test))
55     end = time.time()
56     lr_time_list.append(np.log(end - start))
57
58 # MLPClassifier
59 MLPClassifierModel = MLPClassifier()
60 neural_network_list = []
61 nn_time_list = []
62 for i in range(len(size_list)):
63     start = time.time()
64     MLPClassifierModel.fit(X_train[:size_list[i]], Y_train[:size_list[i]])
65     neural_network_list.append(MLPClassifierModel.score(X_train, Y_train))
66     end = time.time()
67     nn_time_list.append(np.log(end - start))
68
69 # SVC
70 SVCModel = SVC()
71 svc_list = []
72 svc_time_list = []
73 for i in range(len(size_list)):
74     start = time.time()
75     SVCModel.fit(X_train[:size_list[i]], Y_train[:size_list[i]])
76     svc_list.append(SVCModel.score(X_test, Y_test))
77     end = time.time()
78     svc_time_list.append(np.log(end - start))
79
80 plt.plot(size_list, dt_time_list, label='Decision Tree')
81 plt.plot(size_list, rf_time_list, label='Random Forest')
82 plt.plot(size_list, ab_time_list, label='AdaBoost')
83 plt.plot(size_list, lr_time_list, label='Logistic Regression')
84 plt.plot(size_list, nn_time_list, label='Neural Network')
85 plt.plot(size_list, svc_time_list, label='SVM')
86 plt.legend()
87 plt.show()

```

Random Forest and AdaBoost is stable and fast while increasing the number of data, SVM model's time gets more while data amount increases, Decision Tree and Logistic Regression is slow and hardly affected by amount of data.

(d)

```

1 data, target = create_dataset(n=2000, nf=20, nr=12, ni=8, random_state=25)
2 X_train, X_test, Y_train, Y_test = train_test_split(data, target,
3   train_size=0.5, test_size=0.5, random_state=15)
4 DecisionTreeClassifierModel = DecisionTreeClassifier()
5 DecisionTreeClassifierModel.fit(X_train, Y_train)
6
7 train_accuracy = DecisionTreeClassifierModel.score(X_train, Y_train)
8 test_accuracy = DecisionTreeClassifierModel.score(X_test, Y_test)
9
10 print('train_accuracy: ', train_accuracy)
11 print('test_accuracy: ', test_accuracy)

```

```

1 train_accuracy: 1.0
2 test_accuracy: 0.832

```

(e)

```

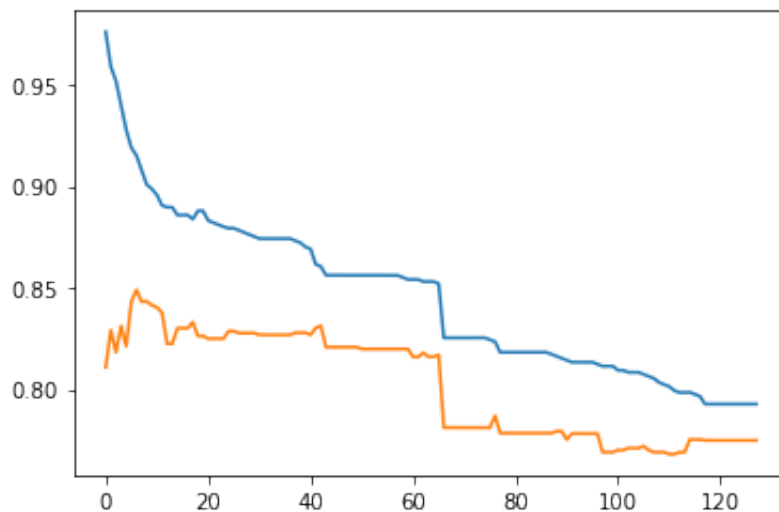
1 from sklearn.metrics import roc_curve
2 from sklearn.metrics import auc
3
4 data, target = create_dataset(n=2000, nf=20, nr=12, ni=8, random_state=25)
5 X_train, X_test, Y_train, Y_test = train_test_split(data, target,
6   train_size=0.5, test_size=0.5, random_state=15)
7 auc_train_list = []

```

```

7  auc_test_list = []
8  for i in range(2, 130):
9      DecisionTreeClassifierModel =
DecisionTreeClassifier(min_samples_leaf=i)
10     DecisionTreeClassifierModel.fit(X_train, Y_train)
11     train_predict = DecisionTreeClassifierModel.predict(X_train)
12     fpr_train, tpr_train, _ = roc_curve(Y_train, train_predict)
13     auc_train_list.append(auc(fpr_train, tpr_train))
14
15     test_predict = DecisionTreeClassifierModel.predict(X_test)
16     fpr_test, tpr_test, _ = roc_curve(Y_test, test_predict)
17     auc_test_list.append(auc(fpr_test, tpr_test))
18
19 plt.plot(auc_train_list)
20 plt.plot(auc_test_list)
21 plt.show()

```



(f)

```

1  data, target = create_dataset(n=2000, nf=20, nr=12, ni=8, random_state=25)
2  X_train, X_test, Y_train, Y_test = train_test_split(data, target,
train_size=0.5, test_size=0.5, random_state=15)
3  X_train_split = np.split(X_train, 10)
4  X_test_split = np.split(X_test, 10)
5  Y_train_split = np.split(Y_train, 10)
6  Y_test_split = np.split(Y_test, 10)
7  score = []
8  x_axis = [i for i in range(2, 96)]
9  test_accuracy = 0
10 train_accuracy = 0
11 max_auc = 0
12 best_k = 0

```

```

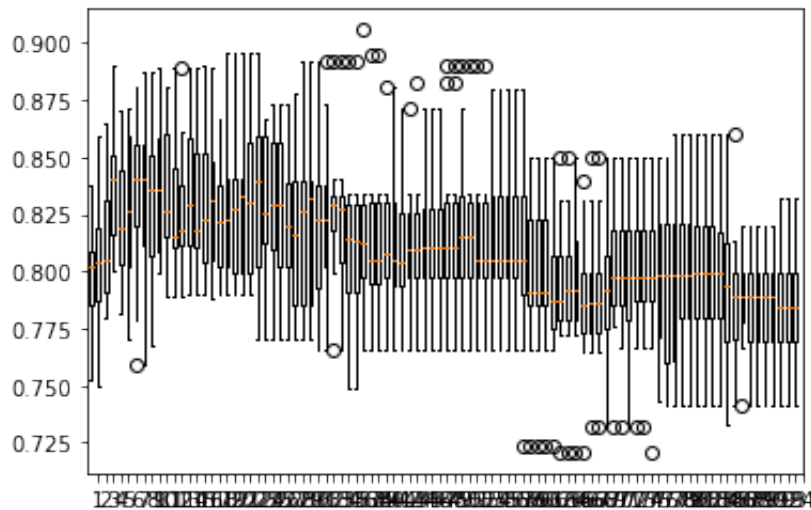
13 for i in range(2, 96):
14     k = 0
15     temp = []
16     for j in range(10):
17         new_x = []
18         new_y = []
19         DecisionTreeClassifierModel =
DecisionTreeClassifier(min_samples_leaf=i)
20         for m in range(10):
21             if m == k:
22                 continue
23             new_x.append(X_train_split[m])
24             new_y.append(Y_train_split[m])
25             new_x = np.concatenate(new_x, axis=0)
26             new_y = np.concatenate(new_y, axis=0)
27             DecisionTreeClassifierModel.fit(new_x, new_y)
28             train_predict =
DecisionTreeClassifierModel.predict(X_train_split[k])
29             ftr, tpr, _ = roc_curve(Y_train_split[k], train_predict)
30             temp_auc = auc(ftr, tpr)
31             if temp_auc > max_auc:
32                 max_auc = temp_auc
33                 train_accuracy = DecisionTreeClassifierModel.score(X_train,
Y_train)
34                 test_accuracy = DecisionTreeClassifierModel.score(X_test,
Y_test)
35                 best_k = i
36                 temp.append(temp_auc)
37                 k += 1
38             score.append(temp)
39         k = 0
40
41 print('min_samples_leaf: ', best_k)
42 print('train_accuracy: ', train_accuracy)
43 print('test_accuracy: ', test_accuracy)
44 plt.boxplot(score)
45 plt.xticks(np.arange(2, 96, 1))
46 plt.show()

```

```

1 min_samples_leaf: 38
2 train_accuracy: 0.862
3 test_accuracy: 0.831

```



(g)

```

1  from sklearn.model_selection import GridSearchCV
2
3  data, target = create_dataset(n=2000, nf=20, nr=12, ni=8, random_state=25)
4  X_train, X_test, Y_train, Y_test = train_test_split(data, target,
5  train_size=0.5, test_size=0.5, random_state=15)
6  X_train_split = np.split(X_train, 10)
7  X_test_split = np.split(X_test, 10)
8  Y_train_split = np.split(Y_train, 10)
9  Y_test_split = np.split(Y_test, 10)
10
11 para = {'min_samples_leaf': [i for i in range(2, 96)]}
12 clf = GridSearchCV(param_grid=para, scoring='roc_auc', cv=10,
13 estimator=DecisionTreeClassifier())
14 clf_res = clf.fit(X_train, Y_train)
15 train_accuracy = clf_res.score(X_train, Y_train)
16 test_accuracy = clf_res.score(X_test, Y_test)
17 # print(clf_res.best_score_)
18 print(clf_res.best_params_)
19 print('train_accuracy: ', train_accuracy)
20 print('test_accuracy: ', test_accuracy)

```

```

1  {'min_samples_leaf': 27}
2  train_accuracy:  0.953077307730773
3  test_accuracy:  0.9007445459310025

```

Because of different data processing methods, gridsearch is random while in (f) we use the k-fold.

