**Python集成学习算法对iris数据进行分类**

**相关知识**

1.集成学习概述

集成学习在机器学习算法中具有较高的准确率，不足之处就是模型的训练过程可能比较复杂，效率不是很高。目前接触较多的集成学习主要有2种：基于Boosting的和基于Bagging，前者的代表算法有Adaboost、GBDT、XGBOOST、后者的代表算法主要是随机森林。

2.集成学习的主要思想

集成学习的主要思想是利用一定的手段学习出多个分类器，而且这多个分类器要求是弱分类器，然后将多个分类器进行组合公共预测。核心思想就是如何训练多个弱分类器以及如何将这些弱分类器进行组合。

3.集成学习中弱分类器选择

一般采用弱分类器的原因在于将误差进行均衡，因为一旦某个分类器太强了就会造成后面的结果受其影响太大，严重的会导致后面的分类器无法进行分类。常用的弱分类器可以采用误差率小于0.5的，比如说逻辑回归、SVM、神经网络。

4.多个分类器的生成

可以采用随机选取数据进行分类器的训练，也可以采用不断的调整错误分类的训练数据的权重生成新的分类器。

5.多个弱分类区如何组合

基本分类器之间的整合方式，一般有简单多数投票、权重投票，贝叶斯投票，基于D-S证据理论的整合，基于不同的特征子集的整合。

**系统环境**

Linux Ubuntu 16.04

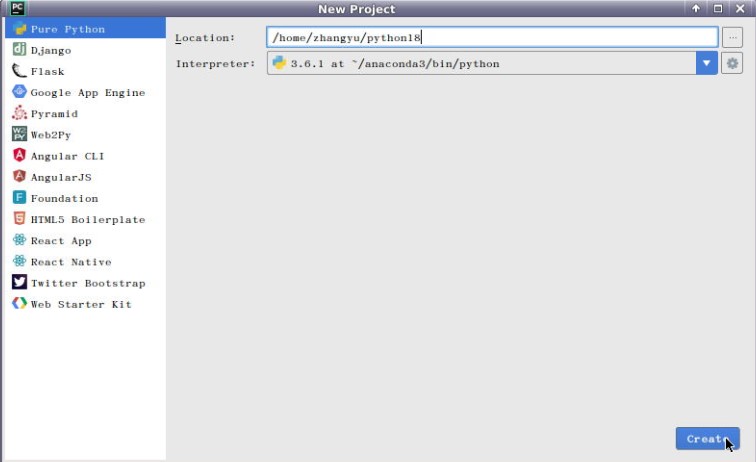
Python3.6

**任务内容**

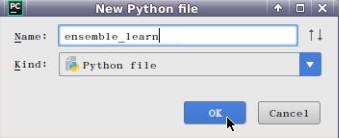
使用ensemble learn算法对iris数据进行分类.

**任务步骤**

1.新建Python Project ，名为python18

[](https://www.ipieuvre.com/doc/exper/28c777f6-91ad-11e9-beeb-00215ec892f4/img/03.jpg)

在python18项目下，新建Python file ，名为ensemble\_learn

[](https://www.ipieuvre.com/doc/exper/28c777f6-91ad-11e9-beeb-00215ec892f4/img/04.jpg)

2.导入 实验所需外包

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. from sklearn **import** clone
4. from sklearn.datasets **import** load\_iris
5. from sklearn.ensemble **import** (RandomForestClassifier, ExtraTreesClassifier,
6. AdaBoostClassifier)
7. from sklearn.tree **import** DecisionTreeClassifier

3.加载数据

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. iris = load\_iris()

4.建立ensemble模型

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. models = [DecisionTreeClassifier(max\_depth=None),
2. RandomForestClassifier(n\_estimators=n\_estimators),
3. ExtraTreesClassifier(n\_estimators=n\_estimators),
4. AdaBoostClassifier(DecisionTreeClassifier(max\_depth=3),
5. n\_estimators=n\_estimators)]

5.特征提取

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. X = iris.data[:, pair]
2. y = iris.target

6.数据规范化

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. mean = X.mean(axis=0)
2. std = X.std(axis=0)
3. X = (X - mean) / std

7.训练模型

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. clf = clone(model)
2. clf = model.fit(X, y)
3. scores = clf.score(X, y)

8.模型可视化

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

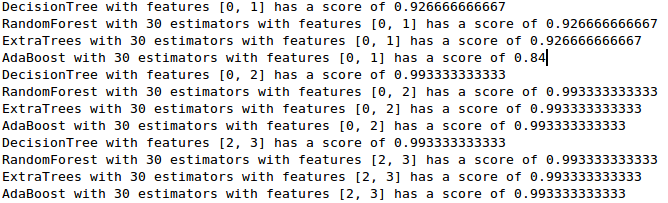
1. # Create a title for each column and the console by using str() and
2. # slicing away useless parts of the string
3. model\_title = str(type(model)).split(".")[-1][:-2][:-len("Classifier")]
4. model\_details = model\_title
5. **if** hasattr(model, "estimators\_"):
6. model\_details += " with {} estimators".format(len(model.estimators\_))
7. print( model\_details + " with features", pair, "has a score of", scores )
8. plt.subplot(3, 4, plot\_idx)
9. **if** plot\_idx <= len(models):
10. # Add a title at the top of each column
11. plt.title(model\_title)
12. # Now plot the decision boundary using a fine mesh as input to a
13. # filled contour plot
14. x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
15. y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
16. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, plot\_step),
17. np.arange(y\_min, y\_max, plot\_step))
18. # Plot either a single DecisionTreeClassifier or alpha blend the
19. # decision surfaces of the ensemble of classifiers
20. **if** isinstance(model, DecisionTreeClassifier):
21. Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])
22. Z = Z.reshape(xx.shape)
23. cs = plt.contourf(xx, yy, Z, cmap=cmap)
24. **else**:
25. # Choose alpha blend level with respect to the number of estimators
26. # that are in use (noting that AdaBoost can use fewer estimators
27. # than its maximum if it achieves a good enough fit early on)
28. estimator\_alpha = 1.0 / len(model.estimators\_)
29. **for** tree **in** model.estimators\_:
30. Z = tree.predict(np.c\_[xx.ravel(), yy.ravel()])
31. Z = Z.reshape(xx.shape)
32. cs = plt.contourf(xx, yy, Z, alpha=estimator\_alpha, cmap=cmap)
33. # Build a coarser grid to plot a set of ensemble classifications
34. # to show how these are different to what we see in the decision
35. # surfaces. These points are regularly space and do not have a black outline
36. xx\_coarser, yy\_coarser = np.meshgrid(np.arange(x\_min, x\_max, plot\_step\_coarser),
37. np.arange(y\_min, y\_max, plot\_step\_coarser))
38. Z\_points\_coarser = model.predict(np.c\_[xx\_coarser.ravel(), yy\_coarser.ravel()]).reshape(xx\_coarser.shape)
39. cs\_points = plt.scatter(xx\_coarser, yy\_coarser, s=15, c=Z\_points\_coarser, cmap=cmap, edgecolors="none")
40. # Plot the training points, these are clustered together and have a
41. # black outline
42. **for** i, c **in** zip(range(n\_classes), plot\_colors):
43. idx = np.where(y == i)
44. plt.scatter(X[idx, 0], X[idx, 1], c=c, label=iris.target\_names[i],
45. cmap=cmap)
46. plot\_idx += 1  # move on to the next plot **in** sequence

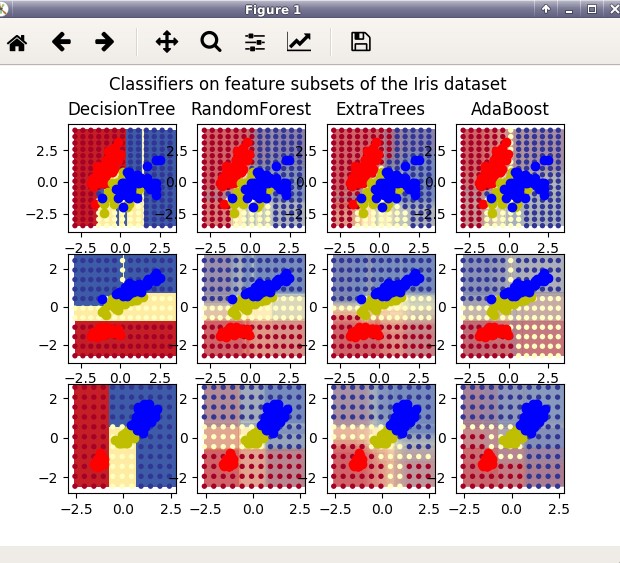
完整代码：

[view plain](https://www.ipieuvre.com/e/187/311/9539) [copy](https://www.ipieuvre.com/e/187/311/9539)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. from sklearn **import** clone
4. from sklearn.datasets **import** load\_iris
5. from sklearn.ensemble **import** (RandomForestClassifier, ExtraTreesClassifier,
6. AdaBoostClassifier)
7. from sklearn.tree **import** DecisionTreeClassifier
8. # Parameters
9. n\_classes = 3
10. n\_estimators = 30
11. plot\_colors = "ryb"
12. cmap = plt.cm.RdYlBu
13. plot\_step = 0.02  # fine step width **for** decision surface contours
14. plot\_step\_coarser = 0.5  # step widths **for** coarse classifier guesses
15. RANDOM\_SEED = 13  # fix the seed on each iteration
16. # Load data
17. iris = load\_iris()
18. plot\_idx = 1
19. models = [DecisionTreeClassifier(max\_depth=None),
20. RandomForestClassifier(n\_estimators=n\_estimators),
21. ExtraTreesClassifier(n\_estimators=n\_estimators),
22. AdaBoostClassifier(DecisionTreeClassifier(max\_depth=3),
23. n\_estimators=n\_estimators)]
24. **for** pair **in** ([0, 1], [0, 2], [2, 3]):
25. **for** model **in** models:
26. # We only take the two corresponding features
27. X = iris.data[:, pair]
28. y = iris.target
29. # Shuffle
30. idx = np.arange(X.shape[0])
31. np.random.seed(RANDOM\_SEED)
32. np.random.shuffle(idx)
33. X = X[idx]
34. y = y[idx]
35. # Standardize
36. mean = X.mean(axis=0)
37. std = X.std(axis=0)
38. X = (X - mean) / std
39. # Train
40. clf = clone(model)
41. clf = model.fit(X, y)
42. scores = clf.score(X, y)
43. # Create a title for each column and the console by using str() and
44. # slicing away useless parts of the string
45. model\_title = str(type(model)).split(".")[-1][:-2][:-len("Classifier")]
46. model\_details = model\_title
47. **if** hasattr(model, "estimators\_"):
48. model\_details += " with {} estimators".format(len(model.estimators\_))
49. print( model\_details + " with features", pair, "has a score of", scores )
50. plt.subplot(3, 4, plot\_idx)
51. **if** plot\_idx <= len(models):
52. # Add a title at the top of each column
53. plt.title(model\_title)
54. # Now plot the decision boundary using a fine mesh as input to a
55. # filled contour plot
56. x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
57. y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
58. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, plot\_step),
59. np.arange(y\_min, y\_max, plot\_step))
60. # Plot either a single DecisionTreeClassifier or alpha blend the
61. # decision surfaces of the ensemble of classifiers
62. **if** isinstance(model, DecisionTreeClassifier):
63. Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])
64. Z = Z.reshape(xx.shape)
65. cs = plt.contourf(xx, yy, Z, cmap=cmap)
66. **else**:
67. # Choose alpha blend level with respect to the number of estimators
68. # that are in use (noting that AdaBoost can use fewer estimators
69. # than its maximum if it achieves a good enough fit early on)
70. estimator\_alpha = 1.0 / len(model.estimators\_)
71. **for** tree **in** model.estimators\_:
72. Z = tree.predict(np.c\_[xx.ravel(), yy.ravel()])
73. Z = Z.reshape(xx.shape)
74. cs = plt.contourf(xx, yy, Z, alpha=estimator\_alpha, cmap=cmap)
75. # Build a coarser grid to plot a set of ensemble classifications
76. # to show how these are different to what we see in the decision
77. # surfaces. These points are regularly space and do not have a black outline
78. xx\_coarser, yy\_coarser = np.meshgrid(np.arange(x\_min, x\_max, plot\_step\_coarser),
79. np.arange(y\_min, y\_max, plot\_step\_coarser))
80. Z\_points\_coarser = model.predict(np.c\_[xx\_coarser.ravel(), yy\_coarser.ravel()]).reshape(xx\_coarser.shape)
81. cs\_points = plt.scatter(xx\_coarser, yy\_coarser, s=15, c=Z\_points\_coarser, cmap=cmap, edgecolors="none")
82. # Plot the training points, these are clustered together and have a
83. # black outline
84. **for** i, c **in** zip(range(n\_classes), plot\_colors):
85. idx = np.where(y == i)
86. plt.scatter(X[idx, 0], X[idx, 1], c=c, label=iris.target\_names[i],
87. cmap=cmap)
88. plot\_idx += 1  # move on to the next plot **in** sequence
89. plt.suptitle("Classifiers on feature subsets of the Iris dataset")
90. plt.axis("tight")
91. plt.show()

运行结果：

[](https://www.ipieuvre.com/doc/exper/28c777f6-91ad-11e9-beeb-00215ec892f4/img/01.png)

[](https://www.ipieuvre.com/doc/exper/28c777f6-91ad-11e9-beeb-00215ec892f4/img/02.jpg)