**PythonScikit-learn分类算法综合**

**相关知识**

常见的分类算法有以下几种：

（1）逻辑回归

逻辑回归（Logistic regression）可以用来回归，也可以用来分类。分类时主要是二分类，预测离散因变量与一组解释变量之间的关系，最常用的是二值型logistic。即因变量的取值只包含两个类别 例如：好、坏 ；发生、不发生。主要在流行病学中应用较多，比较常用的情形是探索某疾病的危险因素，根据危险因素预测某疾病发生的概率，等等。例如，想探讨胃癌发生的危险因素，可以选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群肯定有不同的体征和生活方式等。这里的因变量就是是否胃癌，即“是”或“否”，为两分类变量，自变量就可以包括很多了，例如年龄、性别、饮食习惯、幽门螺杆菌感染等。自变量既可以是连续的，也可以是分类的。通过logistic回归分析，就可以大致了解到底哪些因素是胃癌的危险因素。

（2）决策树

决策树是用于分类和预测的主要技术之一，决策树学习是以实例为基础的归纳学习算法，它着眼于从一组无次序、无规则的实例中推理出以决策树表示的分类规则。构造决策树的目的是找出属性和类别间的关系，用它来预测将来未知类别的记录的类别。它采用自顶向下的递归方式，在决策树的内部节点进行属性的比较，并根据不同属性值判断从该节点向下的分支，在决策树的叶节点得到结论。

主要的决策树算法有ID3、C4.5（C5.0）、CART、PUBLIC、SLIQ和SPRINT算法等。它们在选择测试属性采用的技术、生成的决策树的结构、剪枝的方法以及时刻，能否处理大数据集等方面都有各自的不同之处。

（3）贝叶斯

贝叶斯（Bayes）分类算法是一类利用概率统计知识进行分类的算法，如朴素贝叶斯（Naive Bayes）算法。这些算法主要利用Bayes定理来预测一个未知类别的样本属于各个类别的可能性，选择其中可能性最大的一个类别作为该样本的最终类别。由于贝叶斯定理的成立本身需要一个很强的条件独立性假设前提，而此假设在实际情况中经常是不成立的，因而其分类准确性就会下降。为此就出现了许多降低独立性假设的贝叶斯分类算法，如TAN（Tree Augmented Naive Bayes)算法，它是在贝叶斯网络结构的基础上增加属性对之间的关联来实现的。

（4）人工神经网络

人工神经网络（Artificial Neural Networks，ANN）是一种应用类似于大脑神经突触连接的结构进行信息处理的数学模型。在这种模型中，大量的节点（或称”神经元”，或”单元”）之间相互连接构成网络，即”神经网络”，以达到处理信息的目的。神经网络通常需要进行训练，训练的过程就是网络进行学习的过程。训练改变了网络节点的连接权的值使其具有分类的功能，经过训练的网络就可用于对象的识别。

目前，神经网络已有上百种不同的模型，常见的有BP网络、径向基RBF网络、Hopfield网络、随机神经网络（Boltzmann机）、竞争神经网络（Hamming网络，自组织映射网络）等。但是当前的神经网络仍普遍存在收敛速度慢、计算量大、训练时间长和不可解释等缺点。

（5）k-近邻

k-近邻(kNN，k-Nearest Neighbors)算法是一种基于实例的分类方法。该方法就是找出与未知样本x距离最近的k个训练样本，看这k个样本中多数属于哪一类，就把x归为那一类。k-近邻方法是一种懒惰学习方法，它存放样本，直到需要分类时才进行分类，如果样本集比较复杂，可能会导致很大的计算开销，因此无法应用到实时性很强的场合。

（6）支持向量机

支持向量机（SVM，Support Vector Machine）是Vapnik根据统计学习理论提出的一种新的学习方法，它的最大特点是根据结构风险最小化准则，以最大化分类间隔构造最优分类超平面来提高学习机的泛化能力，较好地解决了非线性、高维数、局部极小点等问题。对于分类问题，支持向量机算法根据区域中的样本计算该区域的决策曲面，由此确定该区域中未知样本的类别。

（7）基于关联规则的分类

关联规则挖掘是数据挖掘中一个重要的研究领域。近年来，对于如何将关联规则挖掘用于分类问题，学者们进行了广泛的研究。关联分类方法形如condset→C的规则，其中condset是项(或属性-值对)的集合，而C是类标号，这种形式的规则称为类关联规则（class association rules，CARS）。关联分类方法一般由两步组成：第一步用关联规则挖掘算法从训练数据集中挖掘出所有满足指定支持度和置信度的类关联规则；第二步使用启发式方法从挖掘出的类关联规则中挑选出一组高质量的规则用于分类。属于关联分类的算法主要包括CBA ，ADT，CMAR 等。

（8）集成学习（Ensemble Learning）

实际应用的复杂性和数据的多样性往往使得单一的分类方法不够有效。因此，学者们对多种分类方法的融合即集成学习进行了广泛的研究。集成学习已成为国际机器学习界的研究热点，并被称为当前机器学习四个主要研究方向之一。集成学习是一种机器学习范式，它试图通过连续调用单个的学习算法，获得不同的基学习器，然后根据规则组合这些学习器来解决同一个问题，可以显著的提高学习系统的泛化能力。组合多个基学习器主要采用（加权）投票的方法，常见的算法有装袋 （Bagging），提升/推进 （Boosting）等。集成学习由于采用了投票平均的方法组合多个分类器，所以有可能减少单个分类器的误差，获得对问题空间模型更加准确的表示，从而提高分类器的分类准确度。

以上简单介绍了各种主要的分类方法，应该说其都有各自不同的特点及优缺点。而用来比较和评估分类方法的标准主要有：

（1）预测的准确率。模型正确地预测新样本的类标号的能力；

（2）计算速度。包括构造模型以及使用模型进行分类的时间；

（3）强壮性。模型对噪声数据或空缺值数据正确预测的能力；

（4）可伸缩性。对于数据量很大的数据集，有效构造模型的能力；

（5）模型描述的简洁性和可解释性。模型描述愈简洁、愈容易理解，则愈受欢迎。

**系统环境**

Linux Ubuntu 16.04

Python3.6

**任务内容**

本次使用mnist手写体库进行实验。共5万训练样本和1万测试样本。

**任务步骤**

1.首先在Linux上新建/data/python10目录，并切换到该目录下。

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. sudo mkdir -p /data/python10/

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. cd /data/python10/

修改/data/python10目录下的所有文件及子目录的所有者和所属的组

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

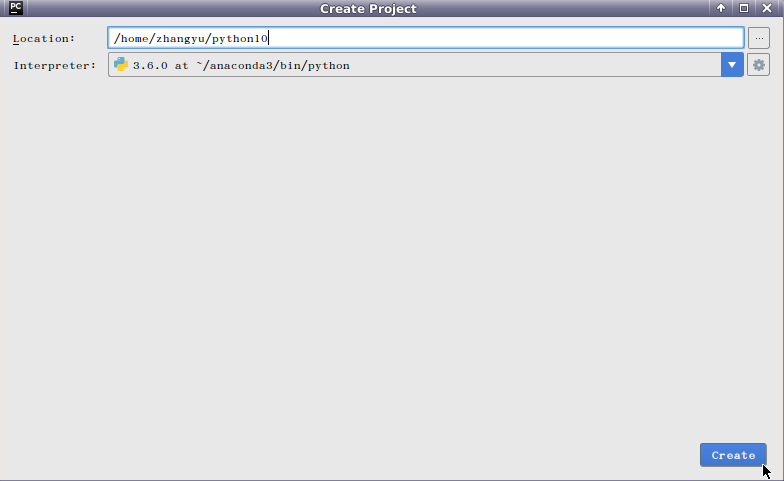
1. sudo chown -R zhangyu.zhangyu /data/python10

2.使用wge命令，从网址 http://192.168.1.100:60000/allfiles/python10/目录下，将实验所需数据下载到linux本地/data/python10目录下。

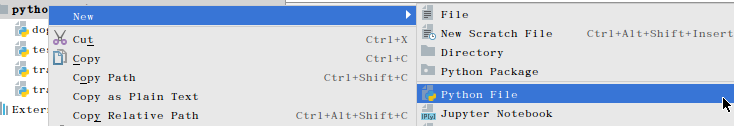
[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. sudo wget http://192.168.1.100:60000/allfiles/python10/mnist.pkl.gz

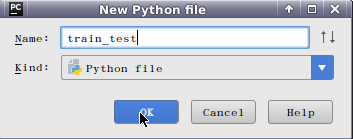
3.新建Python 项目，名为python10

[](https://www.ipieuvre.com/doc/exper/2852ffa6-91ad-11e9-beeb-00215ec892f4/img/01.png)

在python10项目下新建Python file

[](https://www.ipieuvre.com/doc/exper/2852ffa6-91ad-11e9-beeb-00215ec892f4/img/02.png)

名为train\_test.py

[](https://www.ipieuvre.com/doc/exper/2852ffa6-91ad-11e9-beeb-00215ec892f4/img/03.png)

4.提取数据，并将数据分为训练集train与测试集test

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. def read\_data(data\_file):
2. **import** gzip
3. f = gzip.open(data\_file, "rb")
4. train, val, test = pickle.load(f,encoding='latin1')
5. f.close()
6. train\_x = train[0]
7. train\_y = train[1]
8. test\_x = test[0]
9. test\_y = test[1]
10. **return** train\_x, train\_y, test\_x, test\_y

5.创建各种分类模型函数

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. # Multinomial Naive Bayes Classifier
2. def naive\_bayes\_classifier(train\_x, train\_y):
3. from sklearn.naive\_bayes **import** MultinomialNB
4. model = MultinomialNB(alpha=0.01)
5. model.fit(train\_x, train\_y)
6. **return** model
7. # KNN Classifier
8. def knn\_classifier(train\_x, train\_y):
9. from sklearn.neighbors **import** KNeighborsClassifier
10. model = KNeighborsClassifier()
11. model.fit(train\_x, train\_y)
12. **return** model
13. # Logistic Regression Classifier
14. def logistic\_regression\_classifier(train\_x, train\_y):
15. from sklearn.linear\_model **import** LogisticRegression
16. model = LogisticRegression(penalty='l2')
17. model.fit(train\_x, train\_y)
18. **return** model
19. # Random Forest Classifier
20. def random\_forest\_classifier(train\_x, train\_y):
21. from sklearn.ensemble **import** RandomForestClassifier
22. model = RandomForestClassifier(n\_estimators=8)
23. model.fit(train\_x, train\_y)
24. **return** model
25. # Decision Tree Classifier
26. def decision\_tree\_classifier(train\_x, train\_y):
27. from sklearn **import** tree
28. model = tree.DecisionTreeClassifier()
29. model.fit(train\_x, train\_y)
30. **return** model
31. # GBDT(Gradient Boosting Decision Tree) Classifier
32. def gradient\_boosting\_classifier(train\_x, train\_y):
33. from sklearn.ensemble **import** GradientBoostingClassifier
34. model = GradientBoostingClassifier(n\_estimators=200)
35. model.fit(train\_x, train\_y)
36. **return** model
37. # SVM Classifier
38. def svm\_classifier(train\_x, train\_y):
39. from sklearn.svm **import** SVC
40. model = SVC(kernel='rbf', probability=True)
41. model.fit(train\_x, train\_y)
42. **return** model
43. # SVM Classifier using cross validation
44. def svm\_cross\_validation(train\_x, train\_y):
45. from sklearn.grid\_search **import** GridSearchCV
46. from sklearn.svm **import** SVC
47. model = SVC(kernel='rbf', probability=True)
48. param\_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
49. grid\_search = GridSearchCV(model, param\_grid, n\_jobs=1, verbose=1)
50. grid\_search.fit(train\_x, train\_y)
51. best\_parameters = grid\_search.best\_estimator\_.get\_params()
52. **for** para, val **in** best\_parameters.items():
53. print(para, val)
54. model = SVC(kernel='rbf', C=best\_parameters['C'], gamma=best\_parameters['gamma'], probability=True)
55. model.fit(train\_x, train\_y)
56. **return** model

6.用训练集数据训练各种分类器模型，并对测试数据进行预测。

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

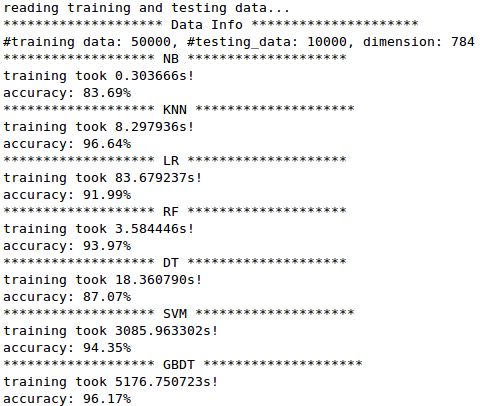
1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. data\_file = "/data/python10/mnist.pkl.gz"
3. thresh = 0.5
4. model\_save\_file = None
5. model\_save = {}
6. test\_classifiers = ['NB', 'KNN', 'LR', 'RF', 'DT', 'SVM', 'GBDT']
7. classifiers = {'NB': naive\_bayes\_classifier,
8. 'KNN': knn\_classifier,
9. 'LR': logistic\_regression\_classifier,
10. 'RF': random\_forest\_classifier,
11. 'DT': decision\_tree\_classifier,
12. 'SVM': svm\_classifier,
13. 'SVMCV': svm\_cross\_validation,
14. 'GBDT': gradient\_boosting\_classifier
15. }
16. print('reading training and testing data...')
17. train\_x, train\_y, test\_x, test\_y = read\_data(data\_file)
18. num\_train, num\_feat = train\_x.shape
19. num\_test, num\_feat = test\_x.shape
20. is\_binary\_class = (len(np.unique(train\_y)) == 2)
21. print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Info \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
22. print('#training data: %d, #testing\_data: %d, dimension: %d' % (num\_train, num\_test, num\_feat))
23. **for** classifier **in** test\_classifiers:
24. print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* %s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' % classifier)
25. start\_time = time.time()
26. model = classifiers[classifier](train\_x, train\_y)
27. print('training took %fs!' % (time.time() - start\_time))
28. predict = model.predict(test\_x)
29. **if** model\_save\_file != None:
30. model\_save[classifier] = model
31. **if** is\_binary\_class:
32. precision = metrics.precision\_score(test\_y, predict)
33. recall = metrics.recall\_score(test\_y, predict)
34. print('precision: %.2f%%, recall: %.2f%%' % (100 \* precision, 100 \* recall))
35. accuracy = metrics.accuracy\_score(test\_y, predict)
36. print('accuracy: %.2f%%' % (100 \* accuracy))
37. **if** model\_save\_file != None:
38. pickle.dump(model\_save, open(model\_save\_file, 'wb'))

完整代码如下：将其写入到train\_test.py文件中运行

[view plain](https://www.ipieuvre.com/e/179/311/9539) [copy](https://www.ipieuvre.com/e/179/311/9539)

1. #-\*- coding: utf-8 -\*-
2. **import** sys
3. **import** os
4. **import** time
5. from sklearn **import** metrics
6. **import** numpy as np
7. **import** pickle
8. # Multinomial Naive Bayes Classifier
9. def naive\_bayes\_classifier(train\_x, train\_y):
10. from sklearn.naive\_bayes **import** MultinomialNB
11. model = MultinomialNB(alpha=0.01)
12. model.fit(train\_x, train\_y)
13. **return** model
14. # KNN Classifier
15. def knn\_classifier(train\_x, train\_y):
16. from sklearn.neighbors **import** KNeighborsClassifier
17. model = KNeighborsClassifier()
18. model.fit(train\_x, train\_y)
19. **return** model
20. # Logistic Regression Classifier
21. def logistic\_regression\_classifier(train\_x, train\_y):
22. from sklearn.linear\_model **import** LogisticRegression
23. model = LogisticRegression(penalty='l2')
24. model.fit(train\_x, train\_y)
25. **return** model
26. # Random Forest Classifier
27. def random\_forest\_classifier(train\_x, train\_y):
28. from sklearn.ensemble **import** RandomForestClassifier
29. model = RandomForestClassifier(n\_estimators=8)
30. model.fit(train\_x, train\_y)
31. **return** model
32. # Decision Tree Classifier
33. def decision\_tree\_classifier(train\_x, train\_y):
34. from sklearn **import** tree
35. model = tree.DecisionTreeClassifier()
36. model.fit(train\_x, train\_y)
37. **return** model
38. # GBDT(Gradient Boosting Decision Tree) Classifier
39. def gradient\_boosting\_classifier(train\_x, train\_y):
40. from sklearn.ensemble **import** GradientBoostingClassifier
41. model = GradientBoostingClassifier(n\_estimators=200)
42. model.fit(train\_x, train\_y)
43. **return** model
44. # SVM Classifier
45. def svm\_classifier(train\_x, train\_y):
46. from sklearn.svm **import** SVC
47. model = SVC(kernel='rbf', probability=True)
48. model.fit(train\_x, train\_y)
49. **return** model
50. # SVM Classifier using cross validation
51. def svm\_cross\_validation(train\_x, train\_y):
52. from sklearn.grid\_search **import** GridSearchCV
53. from sklearn.svm **import** SVC
54. model = SVC(kernel='rbf', probability=True)
55. param\_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
56. grid\_search = GridSearchCV(model, param\_grid, n\_jobs=1, verbose=1)
57. grid\_search.fit(train\_x, train\_y)
58. best\_parameters = grid\_search.best\_estimator\_.get\_params()
59. **for** para, val **in** best\_parameters.items():
60. print(para, val)
61. model = SVC(kernel='rbf', C=best\_parameters['C'], gamma=best\_parameters['gamma'], probability=True)
62. model.fit(train\_x, train\_y)
63. **return** model
64. def read\_data(data\_file):
65. **import** gzip
66. f = gzip.open(data\_file, "rb")
67. train, val, test = pickle.load(f,encoding='latin1')
68. f.close()
69. train\_x = train[0]
70. train\_y = train[1]
71. test\_x = test[0]
72. test\_y = test[1]
73. **return** train\_x, train\_y, test\_x, test\_y
74. **if** \_\_name\_\_ == '\_\_main\_\_':
75. data\_file = "/data/python10/mnist.pkl.gz"
76. thresh = 0.5
77. model\_save\_file = None
78. model\_save = {}
79. test\_classifiers = ['NB', 'KNN', 'LR', 'RF', 'DT', 'SVM', 'GBDT']
80. classifiers = {'NB': naive\_bayes\_classifier,
81. 'KNN': knn\_classifier,
82. 'LR': logistic\_regression\_classifier,
83. 'RF': random\_forest\_classifier,
84. 'DT': decision\_tree\_classifier,
85. 'SVM': svm\_classifier,
86. 'SVMCV': svm\_cross\_validation,
87. 'GBDT': gradient\_boosting\_classifier
88. }
89. print('reading training and testing data...')
90. train\_x, train\_y, test\_x, test\_y = read\_data(data\_file)
91. num\_train, num\_feat = train\_x.shape
92. num\_test, num\_feat = test\_x.shape
93. is\_binary\_class = (len(np.unique(train\_y)) == 2)
94. print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Info \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
95. print('#training data: %d, #testing\_data: %d, dimension: %d' % (num\_train, num\_test, num\_feat))
96. **for** classifier **in** test\_classifiers:
97. print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* %s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' % classifier)
98. start\_time = time.time()
99. model = classifiers[classifier](train\_x, train\_y)
100. print('training took %fs!' % (time.time() - start\_time))
101. predict = model.predict(test\_x)
102. **if** model\_save\_file != None:
103. model\_save[classifier] = model
104. **if** is\_binary\_class:
105. precision = metrics.precision\_score(test\_y, predict)
106. recall = metrics.recall\_score(test\_y, predict)
107. print('precision: %.2f%%, recall: %.2f%%' % (100 \* precision, 100 \* recall))
108. accuracy = metrics.accuracy\_score(test\_y, predict)
109. print('accuracy: %.2f%%' % (100 \* accuracy))
110. **if** model\_save\_file != None:
111. pickle.dump(model\_save, open(model\_save\_file, 'wb'))

运行结果如下：

[](https://www.ipieuvre.com/doc/exper/2852ffa6-91ad-11e9-beeb-00215ec892f4/img/04.png)

由输出结果可以看到：对于该特征数据集各分类算法的准确率和运行时间朴素贝叶斯算法运行时间最短，但准确率最低。SVM与GBDT算法的运行时间最长，准确率也很高。

还有一个在实际中非常有效的方法，就是融合这些分类器，再进行决策。例如简单的投票，效果都非常不错。建议在实践中，大家都可以尝试下。