**[Python](http://lib.csdn.net/base/python" \o "Python知识库" \t "_blank)**[**机器学习**](http://lib.csdn.net/base/machinelearning)**库scikit-learn实践**

[zouxy09@qq.com](mailto:zouxy09@qq.com" \t "_blank)

<http://blog.csdn.net/zouxy09>

**一、概述**

       机器学习[**算法**](http://lib.csdn.net/base/datastructure)在近几年[**大数据**](http://lib.csdn.net/base/hadoop)点燃的热火熏陶下已经变得被人所“熟知”，就算不懂得其中各算法理论，叫你喊上一两个著名算法的名字，你也能昂首挺胸脱口而出。当然了，算法之林虽大，但能者还是有限，能适应某些环境并取得较好效果的算法会脱颖而出，而表现平平者则被历史所淡忘。随着机器学习社区的发展和实践验证，这群脱颖而出者也逐渐被人所认可和青睐，同时获得了更多社区力量的支持、改进和推广。

       以最广泛的分类算法为例，大致可以分为线性和非线性两大派别。线性算法有著名的逻辑回归、朴素贝叶斯、最大熵等，非线性算法有随机森林、决策树、神经网络、核机器等等。线性算法举的大旗是训练和预测的效率比较高，但最终效果对特征的依赖程度较高，需要数据在特征层面上是线性可分的。因此，使用线性算法需要在特征工程上下不少功夫，尽量对特征进行选择、变换或者组合等使得特征具有区分性。而非线性算法则牛逼点，可以建模复杂的分类面，从而能更好的拟合数据。

       那在我们选择了特征的基础上，哪个机器学习算法能取得更好的效果呢？谁也不知道。实践是检验哪个好的不二标准。那难道要苦逼到写五六个机器学习的代码吗？No，机器学习社区的力量是强大的，码农界的共识是不重复造轮子！因此，对某些较为成熟的算法，总有某些优秀的库可以直接使用，省去了大伙调研的大部分时间。

       基于目前使用python较多，而python界中远近闻名的机器学习库要数[scikit-learn](http://scikit-learn.org/stable/index.html)莫属了。这个库优点很多。简单易用，接口抽象得非常好，而且文档支持实在感人。本文中，我们可以封装其中的很多机器学习算法，然后进行一次性[**测试**](http://lib.csdn.net/base/softwaretest)，从而便于分析取优。当然了，针对具体算法，超参调优也非常重要。

**二、Scikit-learn的python实践**

**2.1、Python的准备工作**

       Python一个备受欢迎的点是社区支持很多，有非常多优秀的库或者模块。但是某些库之间有时候也存在依赖，所以要安装这些库也是挺繁琐的过程。但总有人忍受不了这种繁琐，都会开发出不少自动化的工具来节省各位客官的时间。其中，个人总结，安装一个python的库有以下三种方法：

1）Anaconda

       这是一个非常齐全的python发行版本，最新的版本提供了多达195个流行的python包，包含了我们常用的numpy、scipy等等科学计算的包。有了它，妈妈再也不用担心我焦头烂额地安装一个又一个依赖包了。Anaconda在手，轻松我有！下载地址如下：<http://www.continuum.io/downloads>

2）Pip

       使用过Ubuntu的人，对apt-get的爱只有自己懂。其实对Python的库的下载和安装可以借助pip工具的。需要安装什么库，直接下载和安装一条龙服务。在pip官网<https://pypi.python.org/pypi/pip>下载安装即可。未来的需求就在#pip install xx 中。

3）源码包

       如果上述两种方法都没有找到你的库，那你直接把库的源码下载回来，解压，然后在目录中会有个setup.py文件。执行#python setup.py install 即可把这个库安装到python的默认库目录中。

**2.2、Scikit-learn的测试**

[scikit-learn](http://scikit-learn.org/stable/index.html)已经包含在Anaconda中。也可以在官方下载源码包进行安装。本文代码里封装了如下机器学习算法，我们修改数据加载函数，即可一键测试：

**[python]** [view plain](http://blog.csdn.net/zouxy09/article/details/48903179) [copy](http://blog.csdn.net/zouxy09/article/details/48903179)

1. classifiers = {'NB':naive\_bayes\_classifier,
2. 'KNN':knn\_classifier,
3. 'LR':logistic\_regression\_classifier,
4. 'RF':random\_forest\_classifier,
5. 'DT':decision\_tree\_classifier,
6. 'SVM':svm\_classifier,
7. 'SVMCV':svm\_cross\_validation,
8. 'GBDT':gradient\_boosting\_classifier
9. }

train\_test.py

**[python]** [view plain](http://blog.csdn.net/zouxy09/article/details/48903179) [copy](http://blog.csdn.net/zouxy09/article/details/48903179)

1. #!usr/bin/env python
2. #-\*- coding: utf-8 -\*-
4. **import** sys
5. **import** os
6. **import** time
7. **from** sklearn **import** metrics
8. **import** numpy as np
9. **import** cPickle as pickle
11. reload(sys)
12. sys.setdefaultencoding('utf8')
14. # Multinomial Naive Bayes Classifier
15. **def** naive\_bayes\_classifier(train\_x, train\_y):
16. **from** sklearn.naive\_bayes **import** MultinomialNB
17. model = MultinomialNB(alpha=0.01)
18. model.fit(train\_x, train\_y)
19. **return** model

22. # KNN Classifier
23. **def** knn\_classifier(train\_x, train\_y):
24. **from** sklearn.neighbors **import** KNeighborsClassifier
25. model = KNeighborsClassifier()
26. model.fit(train\_x, train\_y)
27. **return** model

30. # Logistic Regression Classifier
31. **def** logistic\_regression\_classifier(train\_x, train\_y):
32. **from** sklearn.linear\_model **import** LogisticRegression
33. model = LogisticRegression(penalty='l2')
34. model.fit(train\_x, train\_y)
35. **return** model

38. # Random Forest Classifier
39. **def** random\_forest\_classifier(train\_x, train\_y):
40. **from** sklearn.ensemble **import** RandomForestClassifier
41. model = RandomForestClassifier(n\_estimators=8)
42. model.fit(train\_x, train\_y)
43. **return** model

46. # Decision Tree Classifier
47. **def** decision\_tree\_classifier(train\_x, train\_y):
48. **from** sklearn **import** tree
49. model = tree.DecisionTreeClassifier()
50. model.fit(train\_x, train\_y)
51. **return** model

54. # GBDT(Gradient Boosting Decision Tree) Classifier
55. **def** gradient\_boosting\_classifier(train\_x, train\_y):
56. **from** sklearn.ensemble **import** GradientBoostingClassifier
57. model = GradientBoostingClassifier(n\_estimators=200)
58. model.fit(train\_x, train\_y)
59. **return** model

62. # SVM Classifier
63. **def** svm\_classifier(train\_x, train\_y):
64. **from** sklearn.svm **import** SVC
65. model = SVC(kernel='rbf', probability=True)
66. model.fit(train\_x, train\_y)
67. **return** model
69. # SVM Classifier using cross validation
70. **def** svm\_cross\_validation(train\_x, train\_y):
71. **from** sklearn.grid\_search **import** GridSearchCV
72. **from** sklearn.svm **import** SVC
73. model = SVC(kernel='rbf', probability=True)
74. param\_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
75. grid\_search = GridSearchCV(model, param\_grid, n\_jobs = 1, verbose=1)
76. grid\_search.fit(train\_x, train\_y)
77. best\_parameters = grid\_search.best\_estimator\_.get\_params()
78. **for** para, val **in** best\_parameters.items():
79. **print** para, val
80. model = SVC(kernel='rbf', C=best\_parameters['C'], gamma=best\_parameters['gamma'], probability=True)
81. model.fit(train\_x, train\_y)
82. **return** model
84. **def** read\_data(data\_file):
85. **import** gzip
86. f = gzip.open(data\_file, "rb")
87. train, val, test = pickle.load(f)
88. f.close()
89. train\_x = train[0]
90. train\_y = train[1]
91. test\_x = test[0]
92. test\_y = test[1]
93. **return** train\_x, train\_y, test\_x, test\_y
95. **if** \_\_name\_\_ == '\_\_main\_\_':
96. data\_file = "mnist.pkl.gz"
97. thresh = 0.5
98. model\_save\_file = None
99. model\_save = {}
101. test\_classifiers = ['NB', 'KNN', 'LR', 'RF', 'DT', 'SVM', 'GBDT']
102. classifiers = {'NB':naive\_bayes\_classifier,
103. 'KNN':knn\_classifier,
104. 'LR':logistic\_regression\_classifier,
105. 'RF':random\_forest\_classifier,
106. 'DT':decision\_tree\_classifier,
107. 'SVM':svm\_classifier,
108. 'SVMCV':svm\_cross\_validation,
109. 'GBDT':gradient\_boosting\_classifier
110. }
112. **print** 'reading training and testing data...'
113. train\_x, train\_y, test\_x, test\_y = read\_data(data\_file)
114. num\_train, num\_feat = train\_x.shape
115. num\_test, num\_feat = test\_x.shape
116. is\_binary\_class = (len(np.unique(train\_y)) == 2)
117. **print** '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Info \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'
118. **print** '#training data: %d, #testing\_data: %d, dimension: %d' % (num\_train, num\_test, num\_feat)
120. **for** classifier **in** test\_classifiers:
121. **print** '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* %s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' % classifier
122. start\_time = time.time()
123. model = classifiers[classifier](train\_x, train\_y)
124. **print** 'training took %fs!' % (time.time() - start\_time)
125. predict = model.predict(test\_x)
126. **if** model\_save\_file != None:
127. model\_save[classifier] = model
128. **if** is\_binary\_class:
129. precision = metrics.precision\_score(test\_y, predict)
130. recall = metrics.recall\_score(test\_y, predict)
131. **print** 'precision: %.2f%%, recall: %.2f%%' % (100 \* precision, 100 \* recall)
132. accuracy = metrics.accuracy\_score(test\_y, predict)
133. **print** 'accuracy: %.2f%%' % (100 \* accuracy)
135. **if** model\_save\_file != None:
136. pickle.dump(model\_save, open(model\_save\_file, 'wb'))

**四、测试结果**

       本次使用[mnist](http://yann.lecun.com/exdb/mnist)手写体库进行实验：<http://deeplearning.net/data/mnist/mnist.pkl.gz>。共5万训练样本和1万测试样本。

       代码运行结果如下：

**[python]** [view plain](http://blog.csdn.net/zouxy09/article/details/48903179) [copy](http://blog.csdn.net/zouxy09/article/details/48903179)

1. reading training **and** testing data...
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Info \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. #training data: 50000, #testing\_data: 10000, dimension: 784
4. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NB \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
5. training took 0.287000s!
6. accuracy: 83.69%
7. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KNN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
8. training took 31.991000s!
9. accuracy: 96.64%
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LR \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
11. training took 101.282000s!
12. accuracy: 91.99%
13. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RF \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
14. training took 5.442000s!
15. accuracy: 93.78%
16. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
17. training took 28.326000s!
18. accuracy: 87.23%
19. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SVM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
20. training took 3152.369000s!
21. accuracy: 94.35%
22. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GBDT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
23. training took 7623.761000s!
24. accuracy: 96.18%

       在这个数据集中，由于数据分布的团簇性较好（如果对这个[**数据库**](http://lib.csdn.net/base/mysql)了解的话，看它的t-SNE映射图就可以看出来。由于任务简单，其在deep learning界已被认为是toy dataset），因此KNN的效果不赖。GBDT是个非常不错的算法，在kaggle等大数据比赛中，状元探花榜眼之列经常能见其身影。三个臭皮匠赛过诸葛亮，还是被验证有道理的，特别是三个臭皮匠还能力互补的时候！

       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。