导航

博客园
首　页
新随笔
管　理

< 2018年11月 >

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

公告

昵称：多一点
园龄：2年6个月
粉丝：18
关注：99
+加关注

搜索

谷歌搜索

我的标签

python(42)
pandas(20)
机器学习(16)
每日一linux命令(13)

Python中常用包——sklearn主要模块和基本使用方法

在从事数据科学的人中，最常用的工具就是R和Python了，每个工具都有其利弊，但是Python在各方面都相对胜出一些，这是因为scikit-learn库实现了很多机器学习算法。

加载数据(Data Loading)

我们假设输入时一个特征矩阵或者csv文件。
首先，数据应该被载入内存中。
scikit-learn的实现使用了NumPy中的arrays，所以，我们要使用NumPy来载入csv文件。
以下是从UCI机器学习数据仓库中下载的数据。
样例：

```
1 import numpy as np
 2 import urllib
 3 # url with dataset
 4 url = "http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data"
 5 # download the file
 6 raw_data = urllib.urlopen(url)
 7 # load the CSV file as a numpy matrix
 8 dataset = np.loadtxt(raw_data, delimiter=",")
 9 # separate the data from the target attributes
10 X = dataset[:,0:7]
11 y = dataset[:,8]
```

我们要使用该数据集作为例子，将特征矩阵作为X，目标变量作为y。

linux(13)
sql(11)
爬虫(9)
machine learning(8)
matplotlib(4)
numpy(4)
更多

随笔分类(253)

linux(29)
machine learning(53)
mysql(12)
numpy(1)
pandas(34)
python(120)
visualization(4)

随笔档案(196)

2018年11月 (1)
2018年10月 (3)
2018年9月 (4)
2018年8月 (14)
2018年7月 (7)
2018年6月 (4)
2018年5月 (6)
2018年4月 (4)
2018年3月 (3)
2018年2月 (3)
2018年1月 (22)
2017年12月 (18)
2017年11月 (1)
2017年10月 (5)
2017年9月 (16)
2017年8月 (25)
2017年7月 (32)
2017年6月 (15)
2017年5月 (1)
2017年4月 (7)

注意事项：

（1）可以用浏览器打开那个url，把数据文件保存在本地，然后直接用 np.loadtxt(‘data.txt’, delemiter=”,”) 就可以加载数据了；

（2）X = dataset[:, 0:7]的意思是：把dataset中的所有行，所有1-7列的数据都保存在X中；

数据归一化(Data Normalization)

大多数机器学习算法中的梯度方法对于数据的缩放和尺度都是很敏感的，在开始跑算法之前，我们应该进行归一化或者标准化的过程，这使得特征数据缩放到0-1范围中。scikit-learn提供了归一化的方法，具体解释参考 http://scikit-learn.org/stable/modules/preprocessing.html：

样例：

```
from sklearn import preprocessing
#scale the data attributes
scaled_X = preprocessing.scale(X)

# normalize the data attributes
normalized_X = preprocessing.normalize(X)

# standardize the data attributes
standardized_X = preprocessing.scale(X)
```

特征选择(Feature Selection)

在解决一个实际问题的过程中，选择合适的特征或者构建特征的能力特别重要。这成为特征选择或者特征工程。

特征选择时一个很需要创造力的过程，更多的依赖于直觉和专业知识，并且有很多现成的算法来进行特征的选择。

下面的树算法(Tree algorithms)计算特征的信息量：

样例：

2016年9月 (1)
2016年7月 (3)
2016年6月 (1)

积分与排名

积分 - 94378
排名 - 4318

最新评论

1. Re:pandas 数据类型转换
非常详细，受教了
--慧命

2. Re:基于pandas python sklearn 的美团某商家的评论分类(文本分类)
重复数据的时候可以使用tilearr=[[-2.0016, -0.3718], [1.669 , -0.4386]]np.tile(arr, (2, 1)) # 延第一个维度复制2倍.....
--多一点

3. Re:numpy 中的reshape，flatten，ravel
数据平展，多维数组变成一维数组
对于列表：a=[[1,2,3], [4,5,6],['a','b']]可以使用列表生成器,可以达到类似的效果[y for x in a for y in x]numpy中repeat可以实现同样的效果a.....
--多一点

4. Re:sohu_news搜狐新闻类型分类（系列1）
问题：business类的样本量过少，分类器没有能够学习到足够多，导致该类的误分类比较多，方案1：可以单独

```
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X, y)
# display the relative importance of each attribute
print(model.feature_importances_)
```

输出每个特征的重要程度：

```
[ 0.13784722  0.15383598  0.25451389  0.17476852  0.02847222  0.12314815  0.12741402]
```

算法的使用

scikit-learn实现了机器学习的大部分基础算法，让我们快速了解一下。

逻辑回归（官方文档）

大多数问题都可以归结为二元分类问题。这个算法的优点是可以给出数据所在类别的概率。

样例：

```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)
print('MODEL')
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print('RESULT')
print(metrics.classification_report(expected, predicted))
print('CONFUSION MATRIX')
print(metrics.confusion_matrix(expected, predicted))
```

复制该类的样本量达到某一数量，可以将该问题解决方案2：采用from sklearn.model.....

--多一点

5. Re:Python | 多种编码文件（中文）乱码问题解决

不错, 学习了 ,谢谢!

--Ryan_code

阅读排行榜

1. mysql中select into 和sql中的select into 对比(18175)
2. Pandas透视表（pivot_table）详解(12401)
3. RPC服务不可用总结(12347)
4. python使用matplotlib绘制折线图教程(12069)
5. windows下Graphviz安装及入门教程(4916)

推荐排行榜

1. python中matplotlib的颜色及线条控制(2)
2. python绘图：matplotlib和pandas的应用(2)
3. Python数据可视化-seaborn(2)
4. sklearn的快速使用(1)
5. Python中常用包——sklearn主要模块和基本使用方法(1)

结果：

```
MODEL
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
RESULT
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00         4
        1.0       1.00      1.00      1.00         6

avg / total       1.00      1.00      1.00        10

CONFUSION MATRIX
[[4 0]
 [0 6]]
```

输出结果中的各个参数信息，可以参考官方文档。

朴素贝叶斯（官方文档）

这也是著名的机器学习算法，该方法的任务是还原训练样本数据的分布密度，其在多类别分类中有很好的效果。

样例：

```
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y)
print('MODEL')
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print('RESULT')
print(metrics.classification_report(expected, predicted))
print('CONFUSION MATRIX')
print(metrics.confusion_matrix(expected, predicted))
```

结果：

```
MODEL
GaussianNB()
RESULT
             precision    recall  f1-score   support

        0.0       0.80      1.00      0.89         4
        1.0       1.00      0.83      0.91         6

avg / total       0.92      0.90      0.90        10

CONFUSION MATRIX
[[4 0]
 [1 5]]
```

k近邻（官方文档）

k近邻算法常常被用作是分类算法一部分，比如可以用它来评估特征，在特征选择上我们可以用到它。

样例：

```
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
# fit a k-nearest neighbor model to the data
model = KNeighborsClassifier()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

结果:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=5, p=2, weights='uniform')
             precision    recall  f1-score   support

        0.0       0.75      0.75      0.75         4
        1.0       0.83      0.83      0.83         6

avg / total       0.80      0.80      0.80        10

[[3 1]
 [1 5]]
```

决策树（官方文档)

分类与回归树(Classification and Regression Trees ,CART)算法常用于特征含有类别信息的分类或者回归问题，这种方法非常适用于多分类情况。

样例:

```
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

样例:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=None, splitter='best')
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00         4
        1.0       1.00      1.00      1.00         6

avg / total       1.00      1.00      1.00        10

[[4 0]
 [0 6]]
```

支持向量机（官方文档）

SVM是非常流行的机器学习算法，主要用于分类问题，如同逻辑回归问题，它可以使用一对多的方法进行多类别的分类。

样例：

```
from sklearn import metrics
from sklearn.svm import SVC
# fit a SVM model to the data
model = SVC()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

结果

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00         4
        1.0       1.00      1.00      1.00         6

avg / total       1.00      1.00      1.00        10

[[4 0]
 [0 6]]
```

除了分类和回归算法外，scikit-learn提供了更加复杂的算法，比如聚类算法，还实现了算法组合的技术，如Bagging和Boosting算法。

如何优化算法参数

一项更加困难的任务是构建一个有效的方法用于选择正确的参数，我们需要用搜索的方法来确定参数。scikit-learn提供了实现这一目标的函数。

下面的例子是一个进行正则参数选择的程序：

GridSearchCV官方文档1（模块使用） 官方文档2（原理详解）

样例：

```
1 import numpy as np
 2 from sklearn.linear_model import Ridge
 3 from sklearn.grid_search import GridSearchCV
 4 # prepare a range of alpha values to test
 5 alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
 6 # create and fit a ridge regression model, testing each alpha
 7 model = Ridge()
 8 grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))
 9 grid.fit(X, y)
10 print(grid)
11 # summarize the results of the grid search
12 print(grid.best_score_)
13 print(grid.best_estimator_.alpha)
```

结果：

```
GridSearchCV(cv=None, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, solver='auto', tol=0.001),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'alpha': array([  1.00000e+00,   1.00000e-01,   1.00000e-02,   1.00000e-
03,
         1.00000e-04,   0.00000e+00])},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring=None,
       verbose=0)
-5.59572064238
0.0
```

有时随机从给定区间中选择参数是很有效的方法，然后根据这些参数来评估算法的效果进而选择最佳的那个。

RandomizedSearchCV官方文档（模块使用）官方文档2 （原理详解）

样例：

```
 1 import numpy as np
 2 from scipy.stats import uniform as sp_rand
 3 from sklearn.linear_model import Ridge
 4 from sklearn.grid_search import RandomizedSearchCV
 5 # prepare a uniform distribution to sample for the alpha parameter
 6 param_grid = {'alpha': sp_rand()}
 7 # create and fit a ridge regression model, testing random alpha values
 8 model = Ridge()
 9 rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100)
10 rsearch.fit(X, y)
11 print(rsearch)
12 # summarize the results of the random parameter search
13 print(rsearch.best_score_)
14 print(rsearch.best_estimator_.alpha)
```

参考文献：http://www.jianshu.com/p/1c6efdbce226

http://www.cnblogs.com/CheeseZH/p/5250997.html