★结合Scikit-learn介绍几种常用的特征选择方法 - 罗兵 - 博客园

http://www.cnblogs.com/hhh5460/p/5186226.html

特征选择(排序)对于数据科学家、机器学习从业者来说非常重要。好的特征选择能够提升模型的性能，更能帮助我们理解数据的特点、底层结构，这对进一步改善模型、算法都有着重要作用。
特征选择主要有两个功能：

1. 减少特征数量、降维，使模型泛化能力更强，减少过拟合
2. 增强对特征和特征值之间的理解

拿到数据集，一个特征选择方法，往往很难同时完成这两个目的。通常情况下，我们经常不管三七二十一，选择一种自己最熟悉或者最方便的特征选择方法（往往目的是降维，而忽略了对特征和数据理解的目的）。
在许多机器学习相关的书里，很难找到关于特征选择的内容，因为特征选择要解决的问题往往被视为机器学习的一种副作用，一般不会单独拿出来讨论。
本文将结合Scikit-learn提供的例子介绍几种常用的特征选择方法，它们各自的优缺点和问题。

1 去掉取值变化小的特征 Removing features with low variance

这应该是最简单的特征选择方法了：假设某特征的特征值只有0和1，并且在所有输入样本中，95%的实例的该特征取值都是1，那就可以认为这个特征作用不大。如果100%都是1，那这个特征就没意义了。当特征值都是离散型变量的时候这种方法才能用，如果是连续型变量，就需要将连续变量离散化之后才能用，而且实际当中，一般不太会有95%以上都取某个值的特征存在，所以这种方法虽然简单但是不太好用。可以把它作为特征选择的预处理，先去掉那些取值变化小的特征，然后再从接下来提到的的特征选择方法中选择合适的进行进一步的特征选择。

2 单变量特征选择 Univariate feature selection

单变量特征选择能够对每一个特征进行测试，衡量该特征和响应变量之间的关系，根据得分扔掉不好的特征。对于回归和分类问题可以采用卡方检验等方式对特征进行测试。
这种方法比较简单，易于运行，易于理解，通常对于理解数据有较好的效果（但对特征优化、提高泛化能力来说不一定有效）；这种方法有许多改进的版本、变种。

2.1 Pearson相关系数 Pearson Correlation

皮尔森相关系数是一种最简单的，能帮助理解特征和响应变量之间关系的方法，该方法衡量的是变量之间的线性相关性，结果的取值区间为[-1，1]，-1表示完全的负相关(这个变量下降，那个就会上升)，+1表示完全的正相关，0表示没有线性相关。
Pearson Correlation速度快、易于计算，经常在拿到数据(经过清洗和特征提取之后的)之后第一时间就执行。Scipy的pearsonr方法能够同时计算相关系数和p-value，

复制代码

```
import numpy as np
from scipy.stats import pearsonr

np.random.seed(0)
size = 300
x = np.random.normal(0, 1, size)
print "Lower noise", pearsonr(x, x + np.random.normal(0, 1, size))
print "Higher noise", pearsonr(x, x + np.random.normal(0, 10, size))
```

复制代码

```
Lower noise (0.71824836862138386, 7.3240173129992273e-49)
Higher noise (0.057964292079338148, 0.31700993885324746)
```

这个例子中，我们比较了变量在加入噪音之前和之后的差异。当噪音比较小的时候，相关性很强，p-value很低。
Scikit-learn提供的f_regrssion方法能够批量计算特征的p-value，非常方便，参考sklearn的pipeline
Pearson相关系数的一个明显缺陷是，作为特征排序机制，他只对线性关系敏感。如果关系是非线性的，即便两个变量具有一一对应的关系，Pearson相关性也可能会接近0。

```
x = np.random.uniform(-1, 1, 100000)
print pearsonr(x, x**2)[0]
```

```
-0.00230804707612
```

更多类似的例子参考sample plots。另外，如果仅仅根据相关系数这个值来判断的话，有时候会具有很强的误导性，如Anscombe’ s quartet，最好把数据可视化出来，以免得出错误的结论。

2.2 互信息和最大信息系数 Mutual information and maximal information coefficient (MIC)

以上就是经典的互信息公式了。想把互信息直接用于特征选择其实不是太方便：1、它不属于度量方式，也没有办法归一化，在不同数据及上的结果无法做比较；2、对于连续变量的计算不是很方便（X和Y都是集合，x，y都是离散的取值），通常变量需要先离散化，而互信息的结果对离散化的方式很敏感。
最大信息系数克服了这两个问题。它首先寻找一种最优的离散化方式，然后把互信息取值转换成一种度量方式，取值区间在[0，1]。minepy提供了MIC功能。
反过头来看y=x^2这个例子，MIC算出来的互信息值为1(最大的取值)。

```
from minepy import MINE

m = MINE()
x = np.random.uniform(-1, 1, 10000)
m.compute_score(x, x**2)
print m.mic()
```

1.0
MIC的统计能力遭到了一些质疑，当零假设不成立时，MIC的统计就会受到影响。在有的数据集上不存在这个问题，但有的数据集上就存在这个问题。

2.3 距离相关系数 (Distance correlation)

距离相关系数是为了克服Pearson相关系数的弱点而生的。在x和x^2这个例子中，即便Pearson相关系数是0，我们也不能断定这两个变量是独立的（有可能是非线性相关）；但如果距离相关系数是0，那么我们就可以说这两个变量是独立的。
R的energy包里提供了距离相关系数的实现，另外这是Python gist的实现。

```
#R-code
> x = runif (1000, -1, 1)
> dcor(x, x**2)
[1] 0.4943864
```

尽管有MIC和距离相关系数在了，但当变量之间的关系接近线性相关的时候，Pearson相关系数仍然是不可替代的。第一、Pearson相关系数计算速度快，这在处理大规模数据的时候很重要。第二、Pearson相关系数的取值区间是[-1，1]，而MIC和距离相关系数都是[0，1]。这个特点使得Pearson相关系数能够表征更丰富的关系，符号表示关系的正负，绝对值能够表示强度。当然，Pearson相关性有效的前提是两个变量的变化关系是单调的。

2.4 基于学习模型的特征排序 (Model based ranking)

这种方法的思路是直接使用你要用的机器学习算法，针对每个单独的特征和响应变量建立预测模型。其实Pearson相关系数等价于线性回归里的标准化回归系数。假如某个特征和响应变量之间的关系是非线性的，可以用基于树的方法（决策树、随机森林）、或者扩展的线性模型等。基于树的方法比较易于使用，因为他们对非线性关系的建模比较好，并且不需要太多的调试。但要注意过拟合问题，因此树的深度最好不要太大，再就是运用交叉验证。
在波士顿房价数据集上使用sklearn的随机森林回归给出一个单变量选择的例子：

复制代码

```
from sklearn.cross_validation import cross_val_score, ShuffleSplit
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor

#Load boston housing dataset as an example
boston = load_boston()
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]

rf = RandomForestRegressor(n_estimators=20, max_depth=4)
scores = []
for i in range(X.shape[1]):
     score = cross_val_score(rf, X[:, i:i+1], Y, scoring="r2",
                             cv=ShuffleSplit(len(X), 3, .3))
     scores.append((round(np.mean(score), 3), names[i]))
print sorted(scores, reverse=True)
```

复制代码

[(0.636, 'LSTAT'), (0.59, 'RM'), (0.472, 'NOX'), (0.369, 'INDUS'), (0.311, 'PTRATIO'), (0.24, 'TAX'), (0.24, 'CRIM'), (0.185, 'RAD'), (0.16, 'ZN'), (0.087, 'B'), (0.062, 'DIS'), (0.036, 'CHAS'), (0.027, 'AGE')]

3 线性模型和正则化

单变量特征选择方法独立的衡量每个特征与响应变量之间的关系，另一种主流的特征选择方法是基于机器学习模型的方法。有些机器学习方法本身就具有对特征进行打分的机制，或者很容易将其运用到特征选择任务中，例如回归模型，SVM，决策树，随机森林等等。说句题外话，这种方法好像在一些地方叫做wrapper类型，大概意思是说，特征排序模型和机器学习模型是耦盒在一起的，对应的非wrapper类型的特征选择方法叫做filter类型。

下面将介绍如何用回归模型的系数来选择特征。越是重要的特征在模型中对应的系数就会越大，而跟输出变量越是无关的特征对应的系数就会越接近于0。在噪音不多的数据上，或者是数据量远远大于特征数的数据上，如果特征之间相对来说是比较独立的，那么即便是运用最简单的线性回归模型也一样能取得非常好的效果。

复制代码

```
from sklearn.linear_model import LinearRegression
import numpy as np

np.random.seed(0)
size = 5000

#A dataset with 3 features
X = np.random.normal(0, 1, (size, 3))
#Y = X0 + 2*X1 + noise
Y = X[:,0] + 2*X[:,1] + np.random.normal(0, 2, size)
lr = LinearRegression()
lr.fit(X, Y)

#A helper method for pretty-printing linear models
def pretty_print_linear(coefs, names = None, sort = False):
    if names == None:
        names = ["X%s" % x for x in range(len(coefs))]
    lst = zip(coefs, names)
    if sort:
        lst = sorted(lst,  key = lambda x:-np.abs(x[0]))
    return " + ".join("%s * %s" % (round(coef, 3), name)
                                  for coef, name in lst)

print "Linear model:", pretty_print_linear(lr.coef_)
```

复制代码

Linear model: 0.984 * X0 + 1.995 * X1 + -0.041 * X2

在这个例子当中，尽管数据中存在一些噪音，但这种特征选择模型仍然能够很好的体现出数据的底层结构。当然这也是因为例子中的这个问题非常适合用线性模型来解：特征和响应变量之间全都是线性关系，并且特征之间均是独立的。

在很多实际的数据当中，往往存在多个互相关联的特征，这时候模型就会变得不稳定，数据中细微的变化就可能导致模型的巨大变化（模型的变化本质上是系数，或者叫参数，可以理解成W），这会让模型的预测变得困难，这种现象也称为多重共线性。例如，假设我们有个数据集，它的真实模型应该是Y=X1+X2，当我们观察的时候，发现Y' =X1+X2+e，e是噪音。如果X1和X2之间存在线性关系，例如X1约等于X2，这个时候由于噪音e的存在，我们学到的模型可能就不是Y=X1+X2了，有可能是Y=2X1，或者Y=-X1+3X2。

下边这个例子当中，在同一个数据上加入了一些噪音，用随机森林算法进行特征选择。

复制代码

```
from sklearn.linear_model import LinearRegression

size = 100
np.random.seed(seed=5)

X_seed = np.random.normal(0, 1, size)
X1 = X_seed + np.random.normal(0, .1, size)
```

```
X2 = X_seed + np.random.normal(0, .1, size)
X3 = X_seed + np.random.normal(0, .1, size)

Y = X1 + X2 + X3 + np.random.normal(0,1, size)
X = np.array([X1, X2, X3]).T

lr = LinearRegression()
lr.fit(X,Y)
print "Linear model:", pretty_print_linear(lr.coef_)
```

复制代码

Linear model: -1.291 * X0 + 1.591 * X1 + 2.747 * X2

系数之和接近3，基本上和上上个例子的结果一致，应该说学到的模型对于预测来说还是不错的。但是，如果从系数的字面意思上去解释特征的重要性的话，X3对于输出变量来说具有很强的正面影响，而X1具有负面影响，而实际上所有特征与输出变量之间的影响是均等的。

同样的方法和套路可以用到类似的线性模型上，比如逻辑回归。

3.1 正则化模型

正则化就是把额外的约束或者惩罚项加到已有模型（损失函数）上，以防止过拟合并提高泛化能力。损失函数由原来的E(X,Y)变为E(X,Y)+alpha||w||，w是模型系数组成的向量（有些地方也叫参数parameter，coefficients），||·||一般是L1或者L2范数，alpha是一个可调的参数，控制着正则化的强度。当用在线性模型上时，L1正则化和L2正则化也称为Lasso和Ridge。

3.2 L1正则化/Lasso

L1正则化将系数w的l1范数作为惩罚项加到损失函数上，由于正则项非零，这就迫使那些弱的特征所对应的系数变成0。因此L1正则化往往会使学到的模型很稀疏（系数w经常为0），这个特性使得L1正则化成为一种很好的特征选择方法。

Scikit-learn为线性回归提供了Lasso，为分类提供了L1逻辑回归。

下面的例子在波士顿房价数据上运行了Lasso，其中参数alpha是通过grid search进行优化的。

复制代码

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston

boston = load_boston()
scaler = StandardScaler()
X = scaler.fit_transform(boston["data"])
Y = boston["target"]
names = boston["feature_names"]

lasso = Lasso(alpha=.3)
lasso.fit(X, Y)

print "Lasso model: ", pretty_print_linear(lasso.coef_, names, sort = True)
```

复制代码

Lasso model: -3.707 * LSTAT + 2.992 * RM + -1.757 * PTRATIO + -1.081 * DIS + -0.7 * NOX + 0.631 * B + 0.54 * CHAS + -0.236 * CRIM + 0.081 * ZN + -0.0 * INDUS + -0.0 * AGE + 0.0 * RAD + -0.0 * TAX

可以看到，很多特征的系数都是0。如果继续增加alpha的值，得到的模型就会越来越稀疏，即越来越多的特征系数会变成0。

然而，L1正则化像非正则化线性模型一样也是不稳定的，如果特征集合中具有相关联的特征，当数据发生细微变化时也有可能导致很大的模型差异。

3.3 L2正则化/Ridge regression

L2正则化将系数向量的L2范数添加到了损失函数中。由于L2惩罚项中系数是二次方的，这使得L2和L1有着诸多差异，最明显的一点就是，L2正则化会让系数的取值变得平均。对于关联特征，这意味着他们能够获得更相近的对应系数。还是以Y=X1+X2为例，假设X1和X2具有很强的关联，如果用L1正则化，不论学到的模型是Y=X1+X2还是Y=2X1，惩罚都是一样的，都是2*alpha*。*但是对于L2来说，第一个模型的惩罚项是2*alpha，但第二个模型的是4*alpha。可以看出，系数之和为常数时，各系数相等时惩罚是最小的，所以才有了L2会让各个系数趋于相同的特点。

可以看出，L2正则化对于特征选择来说一种稳定的模型，不像L1正则化那样，系数会因为细微的数据变化而波动。所以L2正则

化和L1正则化提供的价值是不同的，L2正则化对于特征理解来说更加有用：表示能力强的特征对应的系数是非零。
回过头来看看3个互相关联的特征的例子，分别以10个不同的种子随机初始化运行10次，来观察L1和L2正则化的稳定性。

复制代码

```
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
size = 100

#We run the method 10 times with different random seeds
for i in range(10):
    print "Random seed %s" % i
    np.random.seed(seed=i)
    X_seed = np.random.normal(0, 1, size)
    X1 = X_seed + np.random.normal(0, .1, size)
    X2 = X_seed + np.random.normal(0, .1, size)
    X3 = X_seed + np.random.normal(0, .1, size)
    Y = X1 + X2 + X3 + np.random.normal(0, 1, size)
    X = np.array([X1, X2, X3]).T


    lr = LinearRegression()
    lr.fit(X,Y)
    print "Linear model:", pretty_print_linear(lr.coef_)

    ridge = Ridge(alpha=10)
    ridge.fit(X,Y)
    print "Ridge model:", pretty_print_linear(ridge.coef_)
    print
```

复制代码

Random seed 0 Linear model: 0.728 * X0 + 2.309 * X1 + -0.082 * X2 Ridge model: 0.938 * X0 + 1.059 * X1 + 0.877 * X2
Random seed 1 Linear model: 1.152 * X0 + 2.366 * X1 + -0.599 * X2 Ridge model: 0.984 * X0 + 1.068 * X1 + 0.759 * X2
Random seed 2 Linear model: 0.697 * X0 + 0.322 * X1 + 2.086 * X2 Ridge model: 0.972 * X0 + 0.943 * X1 + 1.085 * X2
Random seed 3 Linear model: 0.287 * X0 + 1.254 * X1 + 1.491 * X2 Ridge model: 0.919 * X0 + 1.005 * X1 + 1.033 * X2
Random seed 4 Linear model: 0.187 * X0 + 0.772 * X1 + 2.189 * X2 Ridge model: 0.964 * X0 + 0.982 * X1 + 1.098 * X2
Random seed 5 Linear model: -1.291 * X0 + 1.591 * X1 + 2.747 * X2 Ridge model: 0.758 * X0 + 1.011 * X1 + 1.139 * X2
Random seed 6 Linear model: 1.199 * X0 + -0.031 * X1 + 1.915 * X2 Ridge model: 1.016 * X0 + 0.89 * X1 + 1.091 * X2
Random seed 7 Linear model: 1.474 * X0 + 1.762 * X1 + -0.151 * X2 Ridge model: 1.018 * X0 + 1.039 * X1 + 0.901 * X2
Random seed 8 Linear model: 0.084 * X0 + 1.88 * X1 + 1.107 * X2 Ridge model: 0.907 * X0 + 1.071 * X1 + 1.008 * X2
Random seed 9 Linear model: 0.714 * X0 + 0.776 * X1 + 1.364 * X2 Ridge model: 0.896 * X0 + 0.903 * X1 + 0.98 * X2
可以看出，不同的数据上线性回归得到的模型（系数）相差甚远，但对于L2正则化模型来说，结果中的系数非常的稳定，差别较小，都比较接近于1，能够反映出数据的内在结构。

4 随机森林

随机森林具有准确率高、鲁棒性好、易于使用等优点，这使得它成为了目前最流行的机器学习算法之一。随机森林提供了两种特征选择的方法：mean decrease impurity和mean decrease accuracy。

4.1 平均不纯度减少 mean decrease impurity

随机森林由多个决策树构成。决策树中的每一个节点都是关于某个特征的条件，为的是将数据集按照不同的响应变量一分为二。利用不纯度可以确定节点（最优条件），对于分类问题，通常采用基尼不纯度或者信息增益，对于回归问题，通常采用的是方差或者最小二乘拟合。当训练决策树的时候，可以计算出每个特征减少了多少树的不纯度。对于一个决策树森林来说，可以算出每个特征平均减少了多少不纯度，并把它平均减少的不纯度作为特征选择的值。
下边的例子是sklearn中基于随机森林的特征重要度度量方法：

复制代码

```
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
import numpy as np

#Load boston housing dataset as an example
boston = load_boston()
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]
rf = RandomForestRegressor()
rf.fit(X, Y)
print "Features sorted by their score:"
print sorted(zip(map(lambda x: round(x, 4), rf.feature_importances_), names),
             reverse=True)
```

复制代码

Features sorted by their score: [(0.5298, 'LSTAT'), (0.4116, 'RM'), (0.0252, 'DIS'), (0.0172, 'CRIM'), (0.0065, 'NOX'), (0.0035, 'PTRATIO'), (0.0021, 'TAX'), (0.0017, 'AGE'), (0.0012, 'B'), (0.0008, 'INDUS'), (0.0004, 'RAD'), (0.0001, 'CHAS'), (0.0, 'ZN')]

这里特征得分实际上采用的是Gini Importance。使用基于不纯度的方法的时候，要记住：1、这种方法存在偏向，对具有更多类别的变量会更有利；2、对于存在关联的多个特征，其中任意一个都可以作为指示器（优秀的特征），并且一旦某个特征被选择之后，其他特征的重要度就会急剧下降，因为不纯度已经被选中的那个特征降下来了，其他的特征就很难再降低那么多不纯度了，这样一来，只有先被选中的那个特征重要度很高，其他的关联特征重要度往往较低。在理解数据时，这就会造成误解，导致错误的认为先被选中的特征是很重要的，而其余的特征是不重要的，但实际上这些特征对响应变量的作用确实非常接近的（这跟Lasso是很像的）。

特征随机选择方法稍微缓解了这个问题，但总的来说并没有完全解决。下面的例子中，X0、X1、X2是三个互相关联的变量，在没有噪音的情况下，输出变量是三者之和。

复制代码

```
size = 10000
np.random.seed(seed=10)
X_seed = np.random.normal(0, 1, size)
X0 = X_seed + np.random.normal(0, .1, size)
X1 = X_seed + np.random.normal(0, .1, size)
X2 = X_seed + np.random.normal(0, .1, size)
X = np.array([X0, X1, X2]).T
Y = X0 + X1 + X2

rf = RandomForestRegressor(n_estimators=20, max_features=2)
rf.fit(X, Y);
print "Scores for X0, X1, X2:", map(lambda x:round (x,3),
                                    rf.feature_importances_)
```

复制代码

Scores for X0, X1, X2: [0.278, 0.66, 0.062]

当计算特征重要性时，可以看到X1的重要度比X2的重要度要高出10倍，但实际上他们真正的重要度是一样的。尽管数据量已经很大且没有噪音，且用了20棵树来做随机选择，但这个问题还是会存在。

需要注意的一点是，关联特征的打分存在不稳定的现象，这不仅仅是随机森林特有的，大多数基于模型的特征选择方法都存在这个问题。

4.2 平均精确率减少 Mean decrease accuracy

另一种常用的特征选择方法就是直接度量每个特征对模型精确率的影响。主要思路是打乱每个特征的特征值顺序，并且度量顺序变动对模型的精确率的影响。很明显，对于不重要的变量来说，打乱顺序对模型的精确率影响不会太大，但是对于重要的变量来说，打乱顺序就会降低模型的精确率。

这个方法sklearn中没有直接提供，但是很容易实现，下面继续在波士顿房价数据集上进行实现。

复制代码

```
from sklearn.cross_validation import ShuffleSplit
from sklearn.metrics import r2_score
from collections import defaultdict

X = boston["data"]
Y = boston["target"]

rf = RandomForestRegressor()
scores = defaultdict(list)

#crossvalidate the scores on a number of different random splits of the data
for train_idx, test_idx in ShuffleSplit(len(X), 100, .3):
    X_train, X_test = X[train_idx], X[test_idx]
    Y_train, Y_test = Y[train_idx], Y[test_idx]
    r = rf.fit(X_train, Y_train)
    acc = r2_score(Y_test, rf.predict(X_test))
    for i in range(X.shape[1]):
        X_t = X_test.copy()
        np.random.shuffle(X_t[:, i])
        shuff_acc = r2_score(Y_test, rf.predict(X_t))
        scores[names[i]].append((acc-shuff_acc)/acc)
print "Features sorted by their score:"
print sorted([(round(np.mean(score), 4), feat) for
             feat, score in scores.items()], reverse=True)
```

复制代码

Features sorted by their score: [(0.7276, 'LSTAT'), (0.5675, 'RM'), (0.0867, 'DIS'), (0.0407, 'NOX'), (0.0351, 'CRIM'), (0.0233, 'PTRATIO'), (0.0168, 'TAX'), (0.0122, 'AGE'), (0.005, 'B'), (0.0048, 'INDUS'), (0.0043, 'RAD'), (0.0004, 'ZN'), (0.0001, 'CHAS')]

在这个例子当中，LSTAT和RM这两个特征对模型的性能有着很大的影响，打乱这两个特征的特征值使得模型的性能下降了73%和57%。注意，尽管这些我们是在所有特征上进行了训练得到了模型，然后才得到了每个特征的重要性测试，这并不意味着我们扔掉某个或者某些重要特征后模型的性能就一定会下降很多，因为即便某个特征删掉之后，其关联特征一样可以发挥作用，让模型性能基本上不变。

5 两种顶层特征选择算法

之所以叫做顶层，是因为他们都是建立在基于模型的特征选择方法基础之上的，例如回归和SVM，在不同的子集上建立模型，然后汇总最终确定特征得分。

5.1 稳定性选择 Stability selection

稳定性选择是一种基于二次抽样和选择算法相结合较新的方法，选择算法可以是回归、SVM或其他类似的方法。它的主要思想是在不同的数据子集和特征子集上运行特征选择算法，不断的重复，最终汇总特征选择结果，比如可以统计某个特征被认为是重要特征的频率（被选为重要特征的次数除以它所在的子集被测试的次数）。理想情况下，重要特征的得分会接近100%。稍微弱一点的特征得分会是非0的数，而最无用的特征得分将会接近于0。

sklearn在随机lasso和随机逻辑回归中有对稳定性选择的实现。

复制代码

```
from sklearn.linear_model import RandomizedLasso
from sklearn.datasets import load_boston
boston = load_boston()

#using the Boston housing data.
#Data gets scaled automatically by sklearn's implementation
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]

rlasso = RandomizedLasso(alpha=0.025)
rlasso.fit(X, Y)
```

```
print "Features sorted by their score:"
print sorted(zip(map(lambda x: round(x, 4), rlasso.scores_),
                 names), reverse=True)
```

复制代码

Features sorted by their score: [(1.0, 'RM'), (1.0, 'PTRATIO'), (1.0, 'LSTAT'), (0.62, 'CHAS'), (0.595, 'B'), (0.39, 'TAX'), (0.385, 'CRIM'), (0.25, 'DIS'), (0.22, 'NOX'), (0.125, 'INDUS'), (0.045, 'ZN'), (0.02, 'RAD'), (0.015, 'AGE')]

在上边这个例子当中，最高的3个特征得分是1.0，这表示他们总会被选作有用的特征（当然，得分会收到正则化参数alpha的影响，但是sklearn的随机lasso能够自动选择最优的alpha）。接下来的几个特征得分就开始下降，但是下降的不是特别急剧，这跟纯lasso的方法和随机森林的结果不一样。能够看出稳定性选择对于克服过拟合和对数据理解来说都是有帮助的：总的来说，好的特征不会因为有相似的特征、关联特征而得分为0，这跟Lasso是不同的。对于特征选择任务，在许多数据集和环境下，稳定性选择往往是性能最好的方法之一。

5.2 递归特征消除 Recursive feature elimination (RFE)

递归特征消除的主要思想是反复的构建模型（如SVM或者回归模型）然后选出最好的（或者最差的）的特征（可以根据系数来选），把选出来的特征放到一遍，然后在剩余的特征上重复这个过程，直到所有特征都遍历了。这个过程中特征被消除的次序就是特征的排序。因此，这是一种寻找最优特征子集的贪心算法。

RFE的稳定性很大程度上取决于在迭代的时候底层用哪种模型。例如，假如RFE采用的普通的回归，没有经过正则化的回归是不稳定的，那么RFE就是不稳定的；假如采用的是Ridge，而用Ridge正则化的回归是稳定的，那么RFE就是稳定的。

Sklearn提供了RFE包，可以用于特征消除，还提供了RFECV，可以通过交叉验证来对的特征进行排序。

复制代码

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

boston = load_boston()
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]

#use linear regression as the model
lr = LinearRegression()
#rank all features, i.e continue the elimination until the last one
rfe = RFE(lr, n_features_to_select=1)
rfe.fit(X,Y)

print "Features sorted by their rank:"
print sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), names))
```

复制代码

Features sorted by their rank: [(1.0, 'NOX'), (2.0, 'RM'), (3.0, 'CHAS'), (4.0, 'PTRATIO'), (5.0, 'DIS'), (6.0, 'LSTAT'), (7.0, 'RAD'), (8.0, 'CRIM'), (9.0, 'INDUS'), (10.0, 'ZN'), (11.0, 'TAX'), (12.0, 'B'), (13.0, 'AGE')]

6 一个完整的例子

下面将本文所有提到的方法进行实验对比，数据集采用Friedman #1 回归数据（这篇论文中的数据）。数据是用这个公式产生的：

X1到X5是由单变量分布生成的，e是标准正态变量N(0,1)。另外，原始的数据集中含有5个噪音变量 X5,...,X10，跟响应变量是独立的。我们增加了4个额外的变量X11,...X14，分别是X1,...,X4的关联变量，通过f(x)=x+N(0,0.01)生成，这将产生大于0.999的关联系数。这样生成的数据能够体现出不同的特征排序方法应对关联特征时的表现。

接下来将会在上述数据上运行所有的特征选择方法，并且将每种方法给出的得分进行归一化，让取值都落在0-1之间。对于RFE来说，由于它给出的是顺序而不是得分，我们将最好的5个的得分定为1，其他的特征的得分均匀的分布在0-1之间。

复制代码

```
from sklearn.datasets import load_boston
from sklearn.linear_model import (LinearRegression, Ridge,
                                  Lasso, RandomizedLasso)
from sklearn.feature_selection import RFE, f_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from minepy import MINE

np.random.seed(0)

size = 750
X = np.random.uniform(0, 1, (size, 14))

#"Friedamn #1" regression problem
Y = (10 * np.sin(np.pi*X[:,0]*X[:,1]) + 20*(X[:,2] - .5)**2 +
     10*X[:,3] + 5*X[:,4] + np.random.normal(0,1))
#Add 3 additional correlated variables (correlated with X1-X3)
X[:,10:] = X[:,:4] + np.random.normal(0, .025, (size,4))

names = ["x%s" % i for i in range(1,15)]

ranks = {}

def rank_to_dict(ranks, names, order=1):
    minmax = MinMaxScaler()
    ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
    ranks = map(lambda x: round(x, 2), ranks)
    return dict(zip(names, ranks ))

lr = LinearRegression(normalize=True)
lr.fit(X, Y)
ranks["Linear reg"] = rank_to_dict(np.abs(lr.coef_), names)

ridge = Ridge(alpha=7)
ridge.fit(X, Y)
ranks["Ridge"] = rank_to_dict(np.abs(ridge.coef_), names)


lasso = Lasso(alpha=.05)
lasso.fit(X, Y)
ranks["Lasso"] = rank_to_dict(np.abs(lasso.coef_), names)


rlasso = RandomizedLasso(alpha=0.04)
rlasso.fit(X, Y)
ranks["Stability"] = rank_to_dict(np.abs(rlasso.scores_), names)

#stop the search when 5 features are left (they will get equal scores)
rfe = RFE(lr, n_features_to_select=5)
rfe.fit(X,Y)
ranks["RFE"] = rank_to_dict(map(float, rfe.ranking_), names, order=-1)

rf = RandomForestRegressor()
rf.fit(X,Y)
ranks["RF"] = rank_to_dict(rf.feature_importances_, names)


f, pval  = f_regression(X, Y, center=True)
ranks["Corr."] = rank_to_dict(f, names)

mine = MINE()
mic_scores = []
for i in range(X.shape[1]):
    mine.compute_score(X[:,i], Y)
```

```
    m = mine.mic()
    mic_scores.append(m)

ranks["MIC"] = rank_to_dict(mic_scores, names)


r = {}
for name in names:
    r[name] = round(np.mean([ranks[method][name]
                             for method in ranks.keys()]), 2)

methods = sorted(ranks.keys())
ranks["Mean"] = r
methods.append("Mean")

print "\t%s" % "\t".join(methods)
for name in names:
    print "%s\t%s" % (name, "\t".join(map(str,
                         [ranks[method][name] for method in methods])))
```

复制代码

从以上结果中可以找到一些有趣的发现：
特征之间存在**线性关联**关系，每个特征都是独立评价的，因此X1,...X4的得分和X11,...X14的得分非常接近，而噪音特征X5,...,X10正如预期的那样和响应变量之间几乎没有关系。由于变量X3是二次的，因此X3和响应变量之间看不出有关系（除了MIC之外，其他方法都找不到关系）。这种方法能够衡量出特征和响应变量之间的线性关系，但若想选出优质特征来提升模型的泛化能力，这种方法就不是特别给力了，因为所有的优质特征都不可避免的会被挑出来两次。
Lasso能够挑出一些优质特征，同时让其他特征的系数趋于0。当如需要减少特征数的时候它很有用，但是对于数据理解来说不是很好用。（例如在结果表中，X11,X12,X13的得分都是0，好像他们跟输出变量之间没有很强的联系，但实际上不是这样的）
MIC对特征一视同仁，这一点上和关联系数有点像，另外，它能够找出X3和响应变量之间的非线性关系。
随机森林基于不纯度的排序结果非常鲜明，在得分最高的几个特征之后的特征，得分急剧的下降。从表中可以看到，得分第三的特征比第一的小4倍。而其他的特征选择算法就没有下降的这么剧烈。
Ridge将回归系数均匀的分摊到各个关联变量上，从表中可以看出，X11,...,X14和X1,...,X4的得分非常接近。
稳定性选择常常是一种既能够有助于理解数据又能够挑出优质特征的这种选择，在结果表中就能很好的看出。像Lasso一样，它能找到那些性能比较好的特征（X1，X2，X4，X5），同时，与这些特征关联度很强的变量也得到了较高的得分。

总结

1. 对于理解数据、数据的结构、特点来说，单变量特征选择是个非常好的选择。尽管可以用它对特征进行排序来优化模型，但由于它不能发现冗余（例如假如一个特征子集，其中的特征之间具有很强的关联，那么从中选择最优的特征时就很难考虑到冗余的问题）。
2. 正则化的线性模型对于特征理解和特征选择来说是非常强大的工具。L1正则化能够生成稀疏的模型，对于选择特征子集来说非常有用；相比起L1正则化，L2正则化的表现更加稳定，由于有用的特征往往对应系数非零，因此L2正则化对于数据的理解来说很合适。由于响应变量和特征之间往往是非线性关系，可以采用basis expansion的方式将特征转换到一个更加合适的空间当中，在此基础上再考虑运用简单的线性模型。
3. 随机森林是一种非常流行的特征选择方法，它易于使用，一般不需要feature engineering、调参等繁琐的步骤，并且很多工具包都提供了平均不纯度下降方法。它的两个主要问题，1是重要的特征有可能得分很低（关联特征问题），2是这种方法对特征变量类别多的特征越有利（偏向问题）。尽管如此，这种方法仍然非常值得在你的应用中试一试。
4. 特征选择在很多机器学习和数据挖掘场景中都是非常有用的。在使用的时候要弄清楚自己的目标是什么，然后找到哪种方法适用于自己的任务。当选择最优特征以提升模型性能的时候，可以采用交叉验证的方法来验证某种方法是否比其他方法要好。当用特征选择的方法来理解数据的时候要留心，特征选择模型的稳定性非常重要，稳定性差的模型很容易就会导致错误的结论。对数据进行二次采样然后在子集上运行特征选择算法能够有所帮助，如果在各个子集上的结果是一致的，那就可以说在这个数据集上得出来的结论是可信的，可以用这种特征选择模型的结果来理解数据。

Tips

什么是卡方检验？用方差来衡量某个观测频率和理论频率之间差异性的方法
什么是皮尔森卡方检验？这是一种最常用的卡方检验方法，它有两个用途：1是计算某个变量对某种分布的拟合程度，2是根据两个观测变量的Contingency table来计算这两个变量是否是独立的。主要有三个步骤：第一步用方差和的方式来计算观测频率和理论频率之间卡方值；第二步算出卡方检验的自由度（行数-1乘以列数-1）；第三步比较卡方值和对应自由度的卡方分布，判断显著性。

什么是p-value？简单地说，p-value就是为了验证假设和实际之间一致性的统计学意义的值，即假设检验。有些地方叫右尾概率，根据卡方值和自由度可以算出一个固定的p-value，
什么是响应变量(response value)？简单地说，模型的输入叫做explanatroy variables，模型的输出叫做response variables，其实就是要验证该特征对结果造成了什么样的影响
什么是统计能力(statistical power)?
什么是度量(metric)?
什么是零假设(null hypothesis)?在相关性检验中，一般会取“两者之间无关联”作为零假设，而在独立性检验中，一般会取“两者之间是独立”作为零假设。与零假设相对的是备择假设（对立假设），即希望证明是正确的另一种可能。
什么是多重共线性？
什么是grid search？

That’ s it

References