特征选择 (feature\_selection)

[toc]

本文主要参考sklearn(0.18版为主，部分0.17)的1.13节的官方文档，以及一些工程实践整理而成。

　　当数据预处理完成后，我们需要选择有意义的特征输入机器学习的算法和模型进行训练。通常来说，从两个方面考虑来选择特征：

特征是否发散：如果一个特征不发散，例如方差接近于0，也就是说样本在这个特征上基本上没有差异，这个特征对于样本的区分并没有什么用。

特征与目标的相关性：这点比较显见，与目标相关性高的特征，应当优选选择。除移除低方差法外，本文介绍的其他方法均从相关性考虑。

根据特征选择的形式又可以将特征选择方法分为3种：

Filter：过滤法，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。

Wrapper：包装法，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。

Embedded：嵌入法，先使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。类似于Filter方法，但是是通过训练来确定特征的优劣。

特征选择主要有两个目的：

减少特征数量、降维，使模型泛化能力更强，减少过拟合；

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，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）。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。

**Filter**

1. 移除低方差的特征 (Removing features with low variance)

　　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。**

>>> from sklearn.feature\_selection import VarianceThreshold

>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]

>>> sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))

>>> sel.fit\_transform(X)

array([[0, 1],

[1, 0],

[0, 0],

[1, 1],

[1, 0],

[1, 1]])

果然, VarianceThreshold 移除了第一列特征，第一列中特征值为0的概率达到了5/6.

2. 单变量特征选择 (Univariate feature selection)

　　单变量特征选择的原理是分别单独的计算每个变量的某个统计指标，根据该指标来判断哪些指标重要，剔除那些不重要的指标。

**对于分类问题(y离散)，可采用：**

**卡方检验，f\_classif, mutual\_info\_classif，互信息**

**对于回归问题(y连续)，可采用：**

**皮尔森相关系数，f\_regression, mutual\_info\_regression，最大信息系数**

　　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。

　　单变量特征选择基于单变量的统计测试来选择最佳特征。它可以看作预测模型的一项预处理。==Scikit-learn将特征选择程序用包含 transform 函数的对象来展现==：

SelectKBest 移除得分前 k 名以外的所有特征(取top k)

SelectPercentile 移除得分在用户指定百分比以后的特征(取top k%)

对每个特征使用通用的单变量统计检验： 假正率(false positive rate) SelectFpr, 伪发现率(false discovery rate) SelectFdr, 或族系误差率 SelectFwe.

GenericUnivariateSelect 可以设置不同的策略来进行单变量特征选择。同时不同的选择策略也能够使用超参数寻优，从而让我们找到最佳的单变量特征选择策略。

**将特征输入到评分函数，返回一个单变量的f\_score(F检验的值)或p-values(P值，假设检验中的一个标准，P-value用来和显著性水平作比较)，注意SelectKBest 和 SelectPercentile只有得分，没有p-value。**

For classification: chi2, f\_classif, mutual\_info\_classif

For regression: f\_regression, mutual\_info\_regression

Notice:

　　The methods based on F-test estimate the degree of linear dependency between two random variables. (F检验用于评估两个随机变量的线性相关性)On the other hand, mutual information methods can capture any kind of statistical dependency, but being nonparametric, they require more samples for accurate estimation.(另一方面，互信息的方法可以捕获任何类型的统计依赖关系，但是作为一个非参数方法，估计准确需要更多的样本)

Feature selection with sparse data:

　　If you use sparse data (i.e. data represented as sparse matrices), chi2, mutual\_info\_regression, mutual\_info\_classif will deal with the data without making it dense.(如果你使用稀疏数据(比如，使用稀疏矩阵表示的数据), 卡方检验(chi2)、互信息回归(mutual\_info\_regression)、互信息分类(mutual\_info\_classif)在处理数据时可保持其稀疏性.)

Examples:

Univariate Feature Selection

Comparison of F-test and mutual information

2.1 卡方(Chi2)检验

**经典的卡方检验是检验定性自变量对定性因变量的相关性。比如，我们可以对样本进行一次chi2 测试来选择最佳的两项特征：**

>>> from sklearn.datasets import load\_iris

>>> from sklearn.feature\_selection import SelectKBest

>>> from sklearn.feature\_selection import chi2

>>> iris = load\_iris()

>>> X, y = iris.data, iris.target

>>> X.shape

(150, 4)

>>> X\_new = SelectKBest(chi2, k=2).fit\_transform(X, y)

>>> X\_new.shape

(150, 2)

2.2 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，+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.**

import numpy as np

from scipy.stats import pearsonr

np.random.seed(0)

size = 300

x = np.random.normal(0, 1, size)

# pearsonr(x, y)的输入为特征矩阵和目标向量

print("Lower noise", pearsonr(x, x + np.random.normal(0, 1, size)))

print("Higher noise", pearsonr(x, x + np.random.normal(0, 10, size)))

>>>

# 输出为二元组(sorce, p-value)的数组

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f\_score和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。例如：

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。

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

　　经典的互信息（互信息为随机变量X与Y之间的互信息I(X;Y)为单个事件之间互信息的数学期望）也是评价定性自变量对定性因变量的相关性的，互信息计算公式如下：

I(X;Y)=E[I(xi;yj)]=∑xiϵX∑yjϵYp(xi,yj)logp(xi,yj)p(xi)p(yj)

　　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=x2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。

2.4 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x2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。

> 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。

**2.5 基于模型的特征排序 (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。**假如 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(这里使用了交叉验证)：

from sklearn.cross\_validation import cross\_val\_score, ShuffleSplit

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(n\_estimators=20, max\_depth=4)

scores = []

# **单独采用每个特征进行建模，并进行交叉验证**

**for i in range(X.shape[1]):**

**score = cross\_val\_score(rf, X[:, i:i+1], Y, scoring="r2", # 注意X[:, i]和X[:, i:i+1]的区别**

**cv=ShuffleSplit(len(X), 3, .3))**

**scores.append((format(np.mean(score), '.3f'), names[i]))**

print(sorted(scores, reverse=True))

[('0.620', 'LSTAT'), ('0.591', 'RM'), ('0.467', 'NOX'), ('0.342', 'INDUS'), ('0.305', 'TAX'), ('0.240', 'PTRATIO'), ('0.206', 'CRIM'), ('0.187', 'RAD'), ('0.184', 'ZN'), ('0.135', 'B'), ('0.082', 'DIS'), ('0.020', 'CHAS'), ('0.002', 'AGE')]

Wrapper

3. 递归特征消除 (Recursive Feature Elimination)

**递归消除特征法使用一个基模型来进行多轮训练，每轮训练后，移除若干权值系数的特征，再基于新的特征集进行下一轮训练。**

**sklearn官方解释：对特征含有权重的预测模型(例如，线性模型对应参数coefficients)，RFE通过递归减少考察的特征集规模来选择特征。首先，预测模型在原始特征上训练，每个特征指定一个权重。之后，那些拥有最小绝对值权重的特征被踢出特征集。如此往复递归，直至剩余的特征数量达到所需的特征数量。**

　　RFECV 通过交叉验证的方式执行RFE，以此来选择最佳数量的特征：对于一个数量为d的feature的集合，他的所有的子集的个数是2的d次方减1(包含空集)。指定一个外部的学习算法，比如SVM之类的。通过该算法计算所有子集的validation error。**选择error最小的那个子集作为所挑选的特征。**

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

#递归特征消除法，返回特征选择后的数据

#参数estimator为基模型

#参数n\_features\_to\_select为选择的特征个数

RFE(estimator=LogisticRegression(), n\_features\_to\_select=2).fit\_transform(iris.data, iris.target)

示例:

Recursive feature elimination: 一个递归特征消除的示例，展示了在数字分类任务中，像素之间的相关性。

Recursive feature elimination with cross-validation: 一个递归特征消除示例，通过交叉验证的方式自动调整所选特征的数量。

Embedded

4. 使用SelectFromModel选择特征 (Feature selection using SelectFromModel)

　　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。其实Pearson相关系数等价于线性回归里的标准化回归系数。**

**SelectFromModel 作为meta-transformer，能够用于拟合后任何拥有coef\_或feature\_importances\_ 属性的预测模型。 如果特征对应的coef\_ 或 feature\_importances\_ 值低于设定的阈值threshold，那么这些特征将被移除。除了手动设置阈值，也可通过字符串参数调用内置的启发式算法(heuristics)来设置阈值，包括：平均值(“mean”), 中位数(“median”)以及他们与浮点数的乘积，如”0.1\*mean”。**

Examples

Feature selection using SelectFromModel and LassoCV: 在阈值未知的前提下，选择了Boston dataset中两项最重要的特征。

4.1 基于L1的特征选择 (L1-based feature selection)

**使用L1范数作为惩罚项的线性模型(Linear models)会得到稀疏解：大部分特征对应的系数为0。当你希望减少特征的维度以用于其它分类器时，可以通过 feature\_selection.SelectFromModel 来选择不为0的系数。特别指出，常用于此目的的稀疏预测模型有 linear\_model.Lasso（回归）， linear\_model.LogisticRegression 和 svm.LinearSVC（分类）:**

>>> from sklearn.svm import LinearSVC

>>> from sklearn.datasets import load\_iris

>>> from sklearn.feature\_selection import SelectFromModel

>>> iris = load\_iris()

>>> X, y = iris.data, iris.target

>>> X.shape

(150, 4)

>>> lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)

>>> model = SelectFromModel(lsvc, prefit=True)

>>> X\_new = model.transform(X)

>>> X\_new.shape

(150, 3)

　　使用feature\_selection库的SelectFromModel类结合带L1以及L2惩罚项的逻辑回归模型:

from sklearn.feature\_selection import SelectFromModel

#带L1和L2惩罚项的逻辑回归作为基模型的特征选择

#参数threshold为权值系数之差的阈值

SelectFromModel(LR(threshold=0.5, C=0.1)).fit\_transform(iris.data, iris.target)

**对于SVM和逻辑回归，参数C控制稀疏性：C越小，被选中的特征越少。对于Lasso，参数alpha越大，被选中的特征越少。**

示例:

Classification of text documents using sparse features: 不同算法使用基于L1的特征选择进行文档分类的对比。

Note:

L1恢复和压缩感知 (L1-recovery and compressive sensing)

　　对于一个好的alpha值，在满足特定条件下， Lasso 仅使用少量观测值就能够完全恢复出非零的系数。特别地，样本的数量需要“足够大”，否则L1模型的表现会充满随机性，所谓“足够大”取决于非零系数的数量，特征数量的对数，噪声的数量，非零系数的最小绝对值以及设计矩阵X的结构。此外，设计矩阵必须拥有特定的属性，比如不能太过相关(correlated)。 对于非零系数的恢复，还没有一个选择alpha值的通用规则 。alpha值可以通过交叉验证来设置(LassoCV or LassoLarsCV)，尽管这也许会导致模型欠惩罚(under-penalized)：引入少量非相关变量不会影响分数预测。相反BIC (LassoLarsIC) 更倾向于设置较大的alpha值。

Reference Richard G. Baraniuk “Compressive Sensing”, IEEE Signal Processing Magazine [120] July 2007

4.2 随机稀疏模型 (Randomized sparse models)

　　基于L1的稀疏模型的局限在于，当面对一组互相关的特征时，它们只会选择其中一项特征。为了减轻该问题的影响可以使用随机化技术，通过\_多次重新估计稀疏模型来扰乱设计矩阵\_，或通过\_多次下采样数据来统计一个给定的回归量被选中的次数\_。——==稳定性选择 (Stability Selection)==

**RandomizedLasso 实现了使用这项策略的Lasso，RandomizedLogisticRegression 使用逻辑回归，适用于分类任务。要得到整个迭代过程的稳定分数，你可以使用 lasso\_stability\_path。**

　　注意到对于非零特征的检测，要使随机稀疏模型比标准F统计量更有效， 那么模型的参考标准需要是稀疏的，换句话说，非零特征应当只占一小部分。

示例:

Sparse recovery: feature selection for sparse linear models: 比较了不同的特征选择方法，并讨论了它们各自适用的场合。

参考文献:

N. Meinshausen, P. Buhlmann, “Stability selection”, Journal of the Royal Statistical Society, 72 (2010)

F. Bach, “Model-Consistent Sparse Estimation through the Bootstrap”

4.3 基于树的特征选择 (Tree-based feature selection)

**基于树的预测模型（见 sklearn.tree 模块，森林见 sklearn.ensemble 模块）能够用来计算特征的重要程度，因此能用来去除不相关的特征（结合 sklearn.feature\_selection.SelectFromModel）:**

>>> from sklearn.ensemble import ExtraTreesClassifier

>>> from sklearn.datasets import load\_iris

>>> from sklearn.feature\_selection import SelectFromModel

>>> iris = load\_iris()

>>> X, y = iris.data, iris.target

>>> X.shape

(150, 4)

>>> clf = ExtraTreesClassifier()

>>> clf = clf.fit(X, y)

>>> clf.feature\_importances\_

array([ 0.04..., 0.05..., 0.4..., 0.4...])

>>> model = SelectFromModel(clf, prefit=True)

>>> X\_new = model.transform(X)

>>> X\_new.shape

(150, 2)

示例:

Feature importances with forests of trees: 从模拟数据中恢复有意义的特征。

Pixel importances with a parallel forest of trees: 用于人脸识别数据的示例。

5. 将特征选择过程融入pipeline (Feature selection as part of a pipeline)

**特征选择常常被当作学习之前的一项预处理。在scikit-learn中推荐使用**

**sklearn.pipeline.Pipeline:**

clf = Pipeline([

('feature\_selection', SelectFromModel(LinearSVC(penalty="l1"))),

('classification', RandomForestClassifier())

])

clf.fit(X, y)

**在此代码片段中，将　sklearn.svm.LinearSVC 和 sklearn.feature\_selection.SelectFromModel 结合来评估特征的重要性，并选择最相关的特征。之后 sklearn.ensemble.RandomForestClassifier 模型使用转换后的输出训练，即只使用被选出的相关特征。你可以选择其它特征选择方法，或是其它提供特征重要性评估的分类器。**更多详情见 sklearn.pipeline.Pipeline 相关示例。

关于更多，参见另一个文档：

《基于模型的特征选择详解 (Embedded & Wrapper)》

小结：

类 所属方式 说明

VarianceThreshold Filter 方差选择法(移除低方差的特征)

SelectKBest Filter 可选关联系数、卡方校验、最大信息系数作为得分计算的方法

RFE Wrapper 递归地训练基模型，将权值系数较小的特征从特征集合中消除

SelectFromModel Embedded 训练基模型，选择权值系数较高的特征