谁动了我的特征？——sklearn特征转换行为全记录

目录

---

1 为什么要记录特征转换行为?

使用机器学习算法和模型进行数据挖掘，有时难免事与愿违：我们依仗对业务的理解，对数据的分析，以及工作经验提出了一些特征，但是在模型训练完成后，某些特征可能“身微言轻”——我们认为相关性高的特征并不重要，这时我们便要反思这样的特征提出是否合理；某些特征甚至“南辕北辙”——我们认为正相关的特征结果变成了负相关，造成这种情况很有可能是抽样与整体不相符，模型过于复杂，导致了过拟合。然而，我们怎么判断先前的假设和最后的结果之间的差异呢?

线性模型通常有含有属性coef_，当系数值大于0时为正相关，当系数值小于0时为负相关；另外一些模型含有属性feature_importances_，顾名思义，表示特征的重要性。根据以上两个属性，便可以与先前假设中的特征的相关性（或重要性）进行对比了。但是，理想是丰满的，现实是骨感的。经过复杂的特征转换之后，特征矩阵X已不再是原来的样子：哑变量使特征变多了，特征选择使特征变少了，降维使特征映射到另一个维度中。

累觉不爱了吗？如果，我们能够将最后的特征与原特征对应起来，那么分析特征的系数和重要性又有了意义了。所以，在训练过程（或者转换过程）中，记录下所有特征转换行为是一个有意义的工作。可惜，sklearn暂时并没有提供这样的功能。在这篇博文中，我们尝试对一些常见的转换功能进行行为记录，读者可以在此基础进行进一步的拓展。

---

2 有哪些特征转换的方式?

《使用sklearn做单机特征工程》一文概括了若干常见的转换功能：

类名	功能	说明
StandardScaler	数据预处理（无量纲化）	标准化，基于特征矩阵的列，将特征值转换至服从标准正态分布
MinMaxScaler	数据预处理（无量纲化）	区间缩放，基于最大最小值，将特征值转换到[0, 1]区间上
Normalizer	数据预处理（归一化）	基于特征矩阵的行，将样本向量转换为“单位向量”
Binarizer	数据预处理（二值化）	基于给定阈值，将定量特征按阈值划分
OneHotEncoder	数据预处理（哑编码）	将定性数据编码为定量数据
Imputer	数据预处理（缺失值计算）	计算缺失值，缺失值可填充为均值等
PolynomialFeatures	数据预处理（多项式数据转换）	多项式数据转换
FunctionTransformer	数据预处理（自定义单元数据转换）	使用单变元的函数来转换数据
VarianceThreshold	特征选择（Filter）	方差选择法
SelectKBest	特征选择（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	特征选择（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	特征选择（Embedded）	训练基模型，选择权值系数较高的特征
PCA	降维（无监督）	主成分分析法
LDA	降维（有监督）	线性判别分析法

按照特征数量是否发生变化，这些转换类可分为：

- 无变化：StandardScaler，MinMaxScaler，Normalizer，Binarizer，Imputer，FunctionTransformer*
- 有变化：OneHotEncoder，PolynomialFeatures，VarianceThreshold，SelectKBest，RFE，SelectFromModel，PCA，LDA

FunctionTransformer：自定义的转换函数通常不会使特征数量发生变化*

对于不造成特征数量变化的转换类，我们只需要保持特征不变即可。在此，我们主要研究那些有变化的转换类，其他转换类都默认为无变化。按照映射的形式，可将以上有变化的转换类可分为：

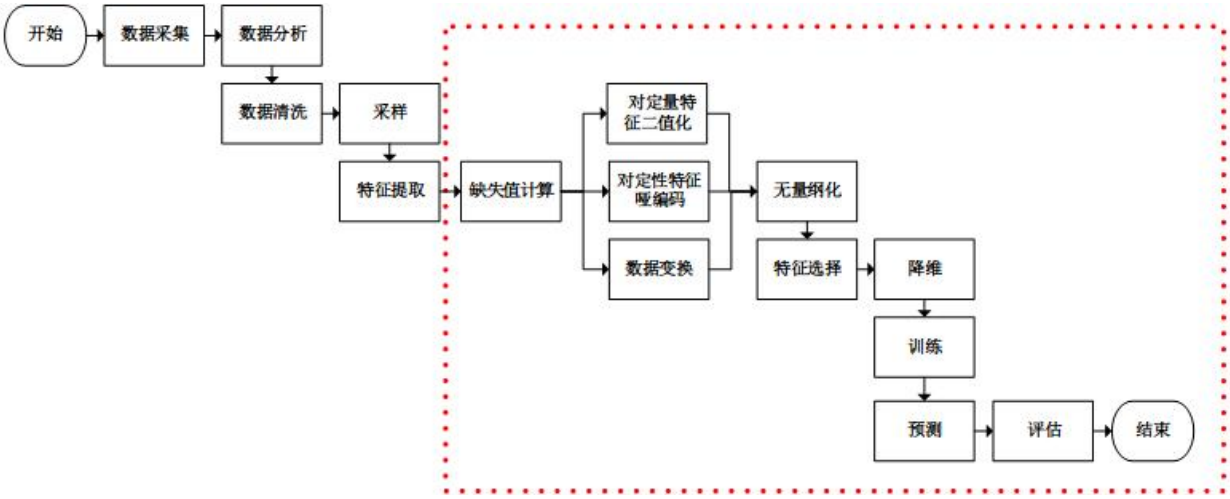
- 一对一：VarianceThreshold，SelectKBest，RFE，SelectFromModel
- 一对多：OneHotEncoder
- 多对多：PolynomialFeatures，PCA，LDA

原特征与新特征为一对一映射通常发生在特征选择时，若原特征被选择则直接变成新特征，否则抛弃。哑编码为典型的一对多映射，需要哑编码的原特征将会转换为多个新特征。多对多的映射中PolynomialFeatures并不要求每一个新特征都与原特征建立映射关系，例如阶为2的多项式转换，第一个新特征只由第一个原特征生成（平方）。降维的本质在于将原特征矩阵X映射到维度更低的空间中，使用的技术通常是矩阵乘法，所以它既要求每一个原特征映射到所有新特征，同时也要求每一个新特征被所有原特征映射。

---

3 特征转换的组合

在《使用sklearn优雅地进行数据挖掘》一文中，我们看到一个基本的数据挖掘场景：



特征转换行为通常是流水线型和并行型结合的。所以，我们考虑重新设计流水线处理类Pipeline和并行处理类FeatureUnion，使其能够根据不同的特征转换类，记录下转换行为“日志”。“日志”的表示形式也是重要的，由上图可知，集成后的特征转换过程呈现无环网状，故使用网络来描述“日志”是合适的。在网络中，节点表示特征，有向连线表示特征转换。

为此，我们新增两个类型Feature和Transfrom来构造网络结构，Feature类型表示网络中的节点，Transform表示网络中的有向边。python的networkx库可以很好地表述网络和操作网络，我这是要重新造轮子吗？其实并不是，现在考虑代表新特征的节点怎么命名的问题，显然，不能与网络中任意节点同名，否则会发生混淆。然而，由于sklearn的训练过程存在并行过程（线程），直接使用network来构造网络的话，将难以处理节点重复命名的问题。所以，我才新增两个新的类型来描述网络结构，这时网络中的节点名是可以重复的。最后，对这网络进行广度遍历，生成基于networkx库的网络，因为这个过程是串行的，故可以使用“当前节点数”作为新增节点的序号了。这两个类的代码（feature.py）设计如下：

```
1 import numpy as np
2
3 class Transform(object):
4     def __init__(self, label, feature):
5         super(Transform, self).__init__()
6         #边标签名，使用networkx等库画图时将用到
7         self.label = label
8         #该边指向的节点
9         self.feature = feature
10
11 class Feature(object):
12     def __init__(self, name):
13         super(Feature, self).__init__()
14         #节点名称，该名称在网络中不唯一，在某些映射中，该名称需要直接传给新特征
15         self.name = name
```

```
        #节点标签名，该名称在网络中唯一，使用networkx等库画图时将用到
        self.label = '%s[%d]' % (self.name, id(self))
        #从本节点发出的有向边列表
        self.transformList = np.array([])

    #建立从self到feature的有向边
    def transform(self, label, feature):
        self.transformList = np.append(self.transformList, Transform(label, feature))

    #深度遍历输出以本节点为源节点的网络
    def printTree(self):
        print self.label
        for transform in self.transformList:
            feature = transform.feature
            print '--%s-->' % transform.label,
            feature.printTree()

    def __str__(self):
        return self.label
```

4 sklearn源码分析

我们可以统一地记录不改变特征数量的转换行为：在“日志”网络中，从代表原特征的节点，引伸出连线连上唯一的代表新特征的节点。然而，对于改变特征数量的转换行为来说，需要针对每个转换类编写不同的“日志”记录（网络生成）代码。为不改变特征数量的转换行为设计代码（default.py）如下：

```
import numpy as np
from feature import Feature

def doWithDefault(model, featureList):
    leaves = np.array([])

    n_features = len(featureList)

    #为每一个输入的原节点，新建一个新节点，并建立映射
    for i in range(n_features):
        feature = featureList[i]
        newFeature = Feature(feature.name)
        feature.transform(model.__class__.__name__, newFeature)
        leaves = np.append(leaves, newFeature)

    #返回新节点列表，之所以该变量取名叫leaves，是因为其是网络的边缘节点
    return leaves
```

4.1 一对一映射

映射形式为一对一时，转换类通常为特征选择类。在这种映射下，原特征要么只转化为一个新特征，要么不转化。通过分析sklearn源码不难发现，特征选择类都混入了特质sklearn.feature_selection.base.SelectorMixin，因此这些类都有方法get_support来获取哪些特征转换信息：

所以，在设计“日志”记录模块时，判断转换类是否混入了该特征，若是则直接调用get_support方法来得到被筛选的特征的掩码或者下标，如此我们便可从被筛选的特征引伸出连线连上新特征。为此，我们设计代码（one2one.py）如下：

```
import numpy as np
from sklearn.feature_selection.base import SelectorMixin
from feature import Feature

def doWithSelector(model, featureList):
```

```
    assert(isinstance(model, SelectorMixin))

    leaves = np.array([])

    n_features = len(featureList)

    #新节点的掩码
    mask_features = model.get_support()

    for i in range(n_features):
        feature = featureList[i]
        #原节点被选择，生成新节点，并建立映射
        if mask_features[i]:
            newFeature = Feature(feature.name)
            feature.transform(model.__class__.__name__, newFeature)
            leaves = np.append(leaves, newFeature)
        #原节点被抛弃，生成一个名为Abandomed的新节点，建立映射，但是这个特征不加入下一步继续生长的节点列表
        else:
            newFeature = Feature('Abandomed')
            feature.transform(model.__class__.__name__, newFeature)

    return leaves
```

4.2 一对多映射

OneHotEncoder是典型的一对多映射转换类，其提供了两个属性结合两个参数来表示转换信息：

- n_values：定性特征的值数量，若为auto则直接从训练集中获取，若为整数则表示所有定性特征的值数量+1，若为数组则分别表示每个定性特征的数量+1
- categorical_features：定性特征的掩码或下标
- active_features_：有效值（在n_values为auto时有用），假设A属性取值范围为（1，2，3），但是实际上训练样本中只有（1，2），假设B属性取值范围为（2，3，4），训练样本中只有（2，4），那么有效值为（1，2，5，7）。是不是感到奇怪了，为什么有效值不是（1，2，2，4）？OneHotEncoder在这里做了巧妙的设计：有效值被转换成了一个递增的序列，这样方便于配合属性n_features快速地算出每个原特征转换成了哪些新特征，转换依据的真实有效值是什么。
- feature_indices_：每个定性特征的有效值范围，例如第i个定性特征，其有效值范围为feature_indices_[i]至feature_indices_[i+1]，sklearn官方文档在此描述有误，该数组的长度应为n_features+1。在上例中，feature_indices_等于（0，3，8）。故下标为0的定性特征，其有效值范围为大于0小于3，则有效值为1和2；下标为1的定性特征，其有效值范围为大于3小于8，则有效值为5和7。下标为0的定性特征，其两个真实有效值为1-0=1和2-0=2；下标为1的定性特征，其两个真实有效值为5-3=2和7-3=4。这样一来就可以得到（1，2，2，4）的真实有效值了。

综上，我们设计处理OneHotEncoder类的代码（one2many.py）如下：

```
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from feature import Feature

def doWithOneHotEncoder(model, featureList):
    assert(isinstance(model, OneHotEncoder))
    assert(hasattr(model, 'feature_indices_'))

    leaves = np.array([])

    n_features = len(featureList)

    #定性特征的掩码
    if model.categorical_features == 'all':
        mask_features = np.ones(n_features)
    else:
        mask_features = np.zeros(n_features)
        mask_features[self.categorical_features] = 1

    #定性特征的数量
    n_qualitativeFeatures = len(model.feature_indices_) - 1
    #如果定性特征的取值个数是自动的，即从训练数据中生成
```

```
    if model.n_values == 'auto':
        #定性特征的有效取值列表
        n_activeFeatures = len(model.active_features_)
    #变量j为定性特征的下标，变量k为有效值的下标
    j = k = 0
    for i in range(n_features):
        feature = featureList[i]
        #如果是定性特征
        if mask_features[i]:
            if model.n_values == 'auto':
                #为属于第j个定性特征的每个有效值生成一个新节点，建立映射关系
                while k < n_activeFeatures and model.active_features_[k] < model.feature_indices_[j+1]:
                    newFeature = Feature(feature.name)
                    feature.transform('%s[%d]' % (model.__class__.__name__, model.active_features_[k] -
model.feature_indices_[j]), newFeature)
                    leaves = np.append(leaves, newFeature)
                    k += 1
            else:
                #为属于第j个定性特征的每个有效值生成一个新节点，建立映射关系
                for k in range(model.feature_indices_[j]+1, model.feature_indices_[j+1]):
                    newFeature = Feature(feature.name)
                    feature.transform('%s[%d]' % (model.__class__.__name__, k - model.feature_indices_[j]),
newFeature)
                    leaves = np.append(leaves, newFeature)
            j += 1
        #如果不是定性特征，则直接根据原节点生成新节点
        else:
            newFeature = Feature(feature.name)
            feature.transform('%s[r]' % model.__class__.__name__, newFeature)
            leaves = append(leaves, newFeatures)

    return leaves
```

4.3 多对多映射

PCA类是典型的多对多映射的转换类，其提供了参数n_components_来表示转换后新特征的个数。之前说过降维的转换类，其既要求每一个原特征映射到所有新特征，也要求每一个新特征被所有原特征映射。故，我们设计处理PCA类的代码（many2many.py）如下：

```
import numpy as np
from sklearn.decomposition import PCA
from feature import Feature

def doWithPCA(model, featureList):
    leaves = np.array([])

    n_features = len(featureList)

    #按照主成分数生成新节点
    for i in range(model.n_components_):
        newFeature = Feature(model.__class__.__name__)
        leaves = np.append(leaves, newFeature)

    #为每一个原节点与每一个新节点建立映射
    for i in range(n_features):
        feature = featureList[i]
        for j in range(model.n_components_):
            newFeature = leaves[j]
            feature.transform(model.__class__.__name__, newFeature)

    return leaves
```

5 实践

到此，我们可以专注改进流水线处理和并行处理的模块了。为了不破坏Pipeline类和FeatureUnion类的核心功能，我们分别派生出两个类PipelineExt和FeatureUnionExt。其次，为这两个类增加私有方法getFeatureList，这个方法有只有一个参数featureList表示输入流水线处理或并行处理的特征列表（元素为feature.Feature类的对象），输出经过流水线处理或并行处理后的特征列表。设计内部方法_doWithModel，其被getFeatureList方法调用，其提供了一个公共的入口，将根据流水线上或者并行中的转换类的不同，具体调用不同的处理方法（这些不同的处理方法在one2one.py，one2many.py，many2many.py中定义）。在者，我们还需要一个initRoot方法来初始化网络结构，返回一个根节点。最后，我们尝试用networkx库读取自定义的网络结构，基于matplotlib的对网络进行图形化显示。以上部分的代码（ple.py）如下：

```
from sklearn.feature_selection.base import SelectorMixin
from sklearn.preprocessing import OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline, FeatureUnion, _fit_one_transformer, _fit_transform_one, _transform_one
from sklearn.externals.joblib import Parallel, delayed
from scipy import sparse
import numpy as np
import networkx as nx
from matplotlib import pyplot as plt
from default import doWithDefault
from one2one import doWithSelector
from one2many import doWithOneHotEncoder
from many2many import doWithPCA
from feature import Feature

#派生Pipeline类
class PipelineExt(Pipeline):
    def _pre_get_featues(self, featureList):
        leaves = featureList
        for name, transform in self.steps[:-1]:
            leaves = _doWithModel(transform, leaves)
        return leaves

    #定义getFeatureList方法
    def getFeatureList(self, featureList):
        leaves = self._pre_get_featues(featureList)
        model = self.steps[-1][-1]
        if hasattr(model, 'fit_transform') or hasattr(model, 'transform'):
            leaves = _doWithModel(model, leaves)
        return leaves

#派生FeatureUnion类，该类不仅记录了转换行为，同时也支持部分数据处理
class FeatureUnionExt(FeatureUnion):
    def __init__(self, transformer_list, idx_list, n_jobs=1, transformer_weights=None):
        self.idx_list = idx_list
        FeatureUnion.__init__(self, transformer_list=map(lambda trans:(trans[0], trans[1]), transformer_list),
n_jobs=n_jobs, transformer_weights=transformer_weights)

    def fit(self, X, y=None):
        transformer_idx_list = map(lambda trans, idx:(trans[0], trans[1], idx), self.transformer_list,
self.idx_list)
        transformers = Parallel(n_jobs=self.n_jobs)(
            delayed(_fit_one_transformer)(trans, X[:,idx], y)
            for name, trans, idx in transformer_idx_list)
        self._update_transformer_list(transformers)
        return self

    def fit_transform(self, X, y=None, **fit_params):
        transformer_idx_list = map(lambda trans, idx:(trans[0], trans[1], idx), self.transformer_list,
self.idx_list)
        result = Parallel(n_jobs=self.n_jobs)(
            delayed(_fit_transform_one)(trans, name, X[:,idx], y,
                                        self.transformer_weights, **fit_params)
            for name, trans, idx in transformer_idx_list)
```

```
52
53         Xs, transformers = zip(*result)
54         self._update_transformer_list(transformers)
55         if any(sparse.issparse(f) for f in Xs):
56             Xs = sparse.hstack(Xs).tocsr()
57         else:
58             Xs = np.hstack(Xs)
59         return Xs
60
61     def transform(self, X):
62         transformer_idx_list = map(lambda trans, idx:(trans[0], trans[1], idx), self.transformer_list,
self.idx_list)
63         Xs = Parallel(n_jobs=self.n_jobs)(
64             delayed(_transform_one)(trans, name, X[:,idx], self.transformer_weights)
65             for name, trans, idx in transformer_idx_list)
66         if any(sparse.issparse(f) for f in Xs):
67             Xs = sparse.hstack(Xs).tocsr()
68         else:
69             Xs = np.hstack(Xs)
70         return Xs
71
72     #定义getFeatureList方法
73     def getFeatureList(self, featureList):
74         transformer_idx_list = map(lambda trans, idx:(trans[0], trans[1], idx), self.transformer_list,
self.idx_list)
75         leaves = np.array(Parallel(n_jobs=self.n_jobs)(
76             delayed(_doWithModel)(trans, featureList[idx])
77             for name, trans, idx in transformer_idx_list))
78         leaves = np.hstack(leaves)
79         return leaves
80
81 #定义为每个模型进行转换记录的总入口方法，该方法将根据不同的转换类调用不同的处理方法
82 def _doWithModel(model, featureList):
83     if isinstance(model, SelectorMixin):
84         return doWithSelector(model, featureList)
85     elif isinstance(model, OneHotEncoder):
86         return doWithOneHotEncoder(model, featureList)
87     elif isinstance(model, PCA):
88         return doWithPCA(model, featureList)
89     elif isinstance(model, FeatureUnionExt) or isinstance(model, PipelineExt):
90         return model.getFeatureList(featureList)
91     else:
92         return doWithDefault(model, featureList)
93
94 #初始化网络的根节点，输入参数为原始特征的名称
95 def initRoot(featureNameList):
96     root = Feature('root')
97     for featureName in featureNameList:
98         newFeature = Feature(featureName)
99         root.transform('init', newFeature)
100     return root
```

现在，我们需要验证一下成果了，不妨继续使用博文《使用sklearn优雅地进行数据挖掘》中提供的场景来进行测试：

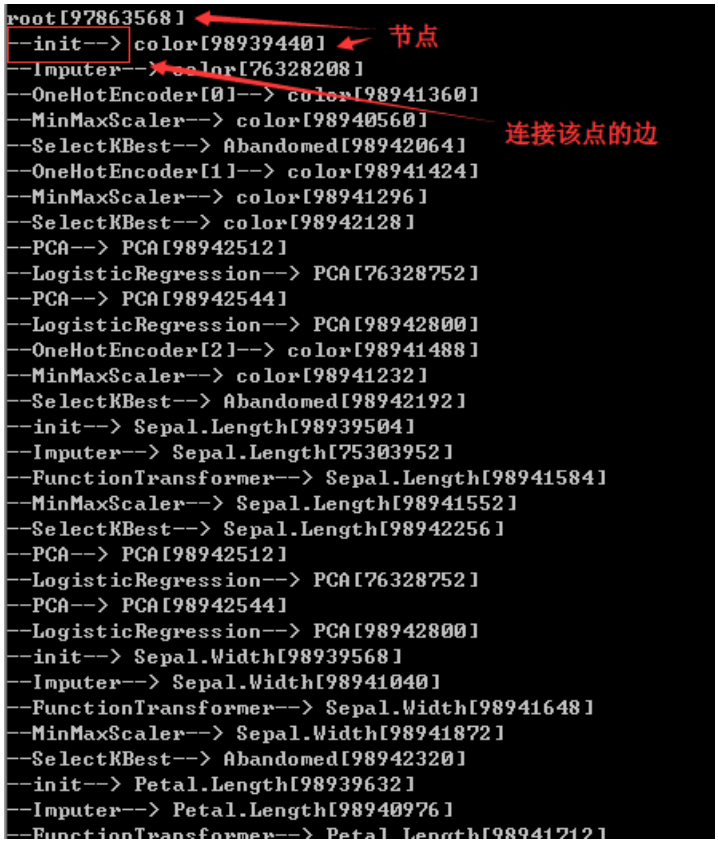
```
1 import numpy as np
2 from sklearn.datasets import load_iris
3 from sklearn.preprocessing import Imputer
4 from sklearn.preprocessing import OneHotEncoder
5 from sklearn.preprocessing import FunctionTransformer
6 from sklearn.preprocessing import Binarizer
7 from sklearn.preprocessing import MinMaxScaler
8 from sklearn.feature_selection import SelectKBest
```

```
from sklearn.feature_selection import chi2
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline, FeatureUnion
from ple import PipelineExt, FeatureUnionExt, initRoot

def datamining(iris, featureList):
    step1 = ('Imputer', Imputer())
    step2_1 = ('OneHotEncoder', OneHotEncoder(sparse=False))
    step2_2 = ('ToLog', FunctionTransformer(np.log1p))
    step2_3 = ('ToBinary', Binarizer())
    step2 = ('FeatureUnionExt', FeatureUnionExt(transformer_list=[step2_1, step2_2, step2_3], idx_list=[[0], [1, 2, 3], [4]]))
    step3 = ('MinMaxScaler', MinMaxScaler())
    step4 = ('SelectKBest', SelectKBest(chi2, k=3))
    step5 = ('PCA', PCA(n_components=2))
    step6 = ('LogisticRegression', LogisticRegression(penalty='l2'))
    pipeline = PipelineExt(steps=[step1, step2, step3, step4, step5, step6])
    pipeline.fit(iris.data, iris.target)
    #最终的特征列表
    leaves = pipeline.getFeatureList(featureList)
    #为最终的特征输出对应的系数
    for i in range(len(leaves)):
        print leaves[i], pipeline.steps[-1][-1].coef_[i]

def main():
    iris = load_iris()
    iris.data = np.hstack((np.random.choice([0, 1, 2], size=iris.data.shape[0]+1).reshape(-1,1), np.vstack((iris.data, np.full(4, np.nan).reshape(1,-1)))))
    iris.target = np.hstack((iris.target, np.array([np.median(iris.target)])))
    root = initRoot(['color', 'Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width'])
    featureList = np.array([transform.feature for transform in root.transformList])

    datamining(iris, featureList)

    root.printTree()

if __name__ == '__main__':
    main()
```

运行程序，最终的特征及对应的系数输出如下：

```
PCA[76328752] [ 1.94335269  5.09215829]
PCA[98942800] [-0.12754262 -1.21307532]
```

输出网络结构的深度遍历（部分截图）：

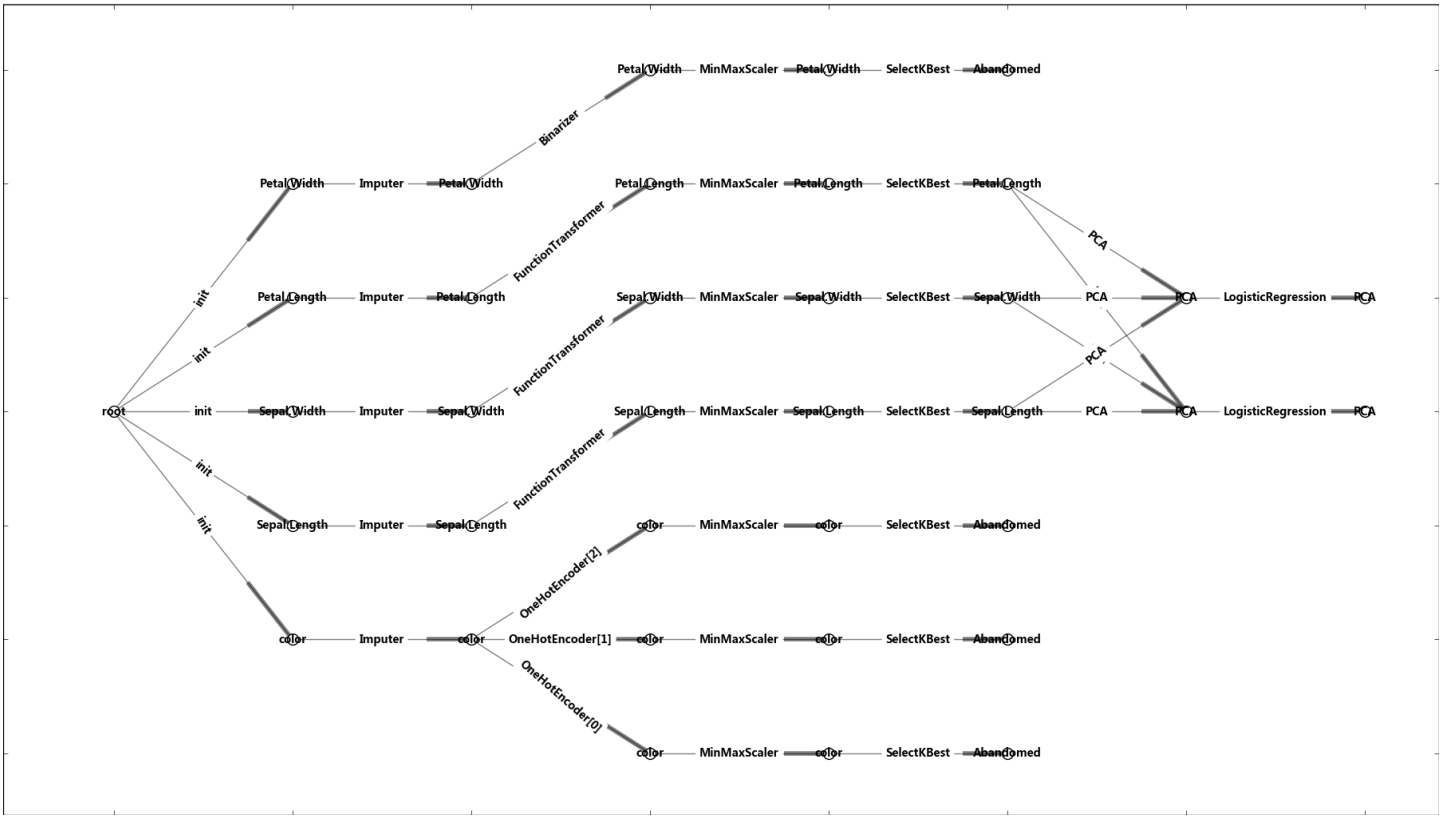


为了更好的展示转换行为构成的网络，我们还可以基于networkx构建有向图，通过matplotlib进行展示（ple.py）：

```
#递归的方式进行深度遍历，生成基于networkx的有向图
def _draw(G, root, nodeLabelDict, edgeLabelDict):
    nodeLabelDict[root.label] = root.name
    for transform in root.transformList:
        G.add_edge(root.label, transform.feature.label)
        edgeLabelDict[(root.label, transform.feature.label)] = transform.label
        _draw(G, transform.feature, nodeLabelDict, edgeLabelDict)

#判断是否图是否存在环
def _isCyclic(root, walked):
    if root in walked:
        return True
    else:
        walked.add(root)
        for transform in root.transformList:
            ret = _isCyclic(transform.feature, walked)
            if ret:
                return True
        walked.remove(root)
        return False

#广度遍历生成瀑布式布局
def fall_layout(root, x_space=1, y_space=1):
    layout = {}
    if _isCyclic(root, set()):
        raise Exception('Graph is cyclic')

    queue = [None, root]
    nodeDict = {}
    levelDict = {}
    level = 0
    while len(queue) > 0:
        head = queue.pop()
        if head is None:
            if len(queue) > 0:
```

```
                level += 1
                queue.insert(0, None)
        else:
            if head in nodeDict:
                levelDict[nodeDict[head]].remove(head)
            nodeDict[head] = level
            levelDict[level] = levelDict.get(level, []) + [head]
            for transform in head.transformList:
                queue.insert(0, transform.feature)

    for level in levelDict.keys():
        nodeList = levelDict[level]
        n_nodes = len(nodeList)
        offset = - n_nodes / 2
        for i in range(n_nodes):
            layout[nodeList[i].label] = (level * x_space, (i + offset) * y_space)

    return layout

def draw(root):
    G = nx.DiGraph()
    nodeLabelDict = {}
    edgeLabelDict = {}

    _draw(G, root, nodeLabelDict, edgeLabelDict)
    #设定网络布局方式为瀑布式
    pos = fall_layout(root)

    nx.draw_networkx_nodes(G,pos,node_size=100, node_color="white")
    nx.draw_networkx_edges(G,pos, width=1,alpha=0.5,edge_color='black')
    #设置网络中节点的标签内容及格式
    nx.draw_networkx_labels(G,pos,labels=nodeLabelDict, font_size=10,font_family='sans-serif')
    #设置网络中边的标签内容及格式
    nx.draw_networkx_edge_labels(G, pos, edgeLabelDict)

    plt.show()
```

以图形界面展示网络的结构：



6 总结

聪明的读者你肯定发现了，记录下特征转换行为的最好时机其实是转换的同时。可惜的是，sklearn目前并不支持这样的功能。在本文中，我将这一功能集中到流水线处理和并行处理的模块当中，只能算是一个临时的手段，但聊胜于无吧。另外，本文也是抛砖引玉，还有其他的转换类，在原特征与新特征之间的映射关系上，百家争鸣。所以，我在Github上新建了个库，包含本文实例中所有的转换类处理的代码，在之后，我会慢慢地填这个坑，直到世界的尽头，抑或sklearn加入该功能。

7 参考资料

1. 《使用sklearn优雅地进行数据挖掘》
2. 《使用sklearn做单机特征工程》
3. sklearn.preprocessing.OneHotEncoder