本周对于随机森林算法进行了学习，阅读了一篇论文，《音乐情感识别研究进展\_陈晓鸥》

仍在查找处理解码MIDI文件将数据转换为矩阵的Pyhton库，查找到较为不错处理 一般音频文件的Python库。

一、关于随机森林算法

1.使用随机森林算法的原因：

随机森林算法几乎不需要输入的准备。它们不需要测算就能够处理二分特征、分类特征、数值特征的数据。随机森林算法能完成隐含特征的选择，并且提供一个很好的特征重要度的选择指标。

随机森林算法训练速度快。性能优化过程刚好又提高了模型的准确性，这种精彩表现并不常有，反之亦然。这种旨在多样化子树的子设定随机特征，同时也是一种突出的性能优化！调低给定任意节点的特征划分，能让你简单的处理带有上千属性的数据集。（如果数据集有很多行的话，这种方法同样的也可以适用于行采样）

随机森林算法很难被打败。针对任何给定的数据集，尽管你常能找到一个优于它的模型（比较典型的是神经网络或者一些增益算法( boosting algorithm），但这类算法肯定不多，而且通常建这样的模型并调试好要比随机森林算法模型要耗时的更多。这也是为何随机森林算法作为基准模型表现出色的原因。

建立一个差劲的随机森林模型真的很难！因为随机森林算法对指定使用的超参数（hyper-parameters ）并不十分敏感。为了要得到一个合适的模型，它们不需要做很多调整。只需使用大量的树，模型就不会产生很多偏差。大多数的随机森林算法的实现方法的参数设置初始值也都是合理的。

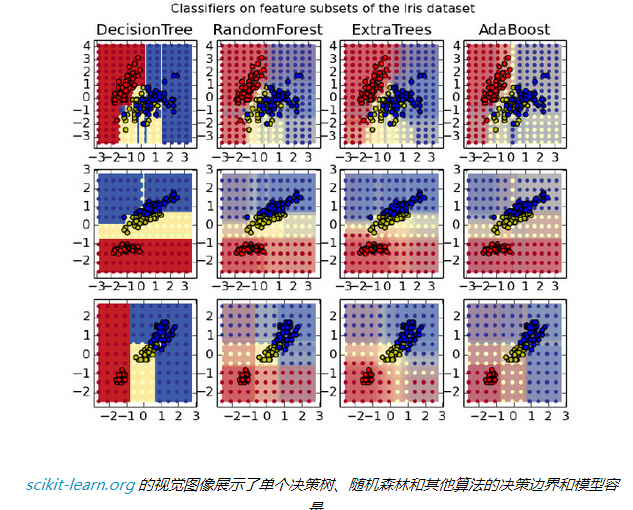
通用性。随机森林算法可以应用于很多类别的模型任务。它们可以很好的处理回归问题，也能对分类问题应付自如（甚至可以产生合适的标准概率值）。虽然我从没亲自尝试，但它们还可以用于[聚类](http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm" \l "cluster) [分析](http://labs.genetics.ucla.edu/horvath/RFclustering/RFclustering/RandomForestHorvath.pdf)问题。

简洁性。对于随机森林来说，如果不是结论模型很简洁，就是学习算法本身很简洁。基本的随机森林学习算法仅用几行代码就可以写出来了。也许这样说有点讽刺。但是也精当得体。

提供了数量庞大的精彩、免费、开源实现方法。你能在几乎所有主要的机器学习资料库和工具箱中找到一个好的实现方法。我现在知道的是，随机森林算法在R 、scikit-learn 和 Weka 上都有非常好的实现方法。

如果以上的理由还不够的话，那么随机森林算法还能很容易进行并行生长。Boosted model 或者大规模神经网络可做不到这一点。

scikit-learn给出了清晰的视觉图展示决策森林的模型容量。



2. 随机森林算法的相关库

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

import numpy as np

from sklearn.datasets import load\_iris

iris=load\_iris()

#print iris#iris的４个属性是：萼片宽度　萼片长度　花瓣宽度　花瓣长度　标签是花的种类：setosa versicolour virginica

print iris['target'].shape

rf=RandomForestRegressor()#这里使用了默认的参数设置

rf.fit(iris.data[:150],iris.target[:150])#进行模型的训练

#

#随机挑选两个预测不相同的样本

instance=iris.data[[100,109]]

print instance

print 'instance 0 prediction；',rf.predict(instance[0])

print 'instance 1 prediction；',rf.predict(instance[1])

print iris.target[100],iris.target[109]

3.随机森林算法

决策树运行的每一步都涉及到对数据集中的最优分裂点（best split point）进行贪婪选择（greedy selection）。

这个机制使得决策树在没有被剪枝的情况下易产生较高的方差。整合通过提取训练数据库中不同样本（某一问题的不同表现形式）构建的复合树及其生成的预测值能够稳定并降低这样的高方差。这种方法被称作引导聚集算法（bootstrap aggregating），其简称 bagging 正好是装进口袋，袋子的意思，所以被称为「装袋算法」。该算法的局限在于，由于生成每一棵树的贪婪算法是相同的，那么有可能造成每棵树选取的分裂点（split point）相同或者极其相似，最终导致不同树之间的趋同（树与树相关联）。相应地，反过来说，这也使得其会产生相似的预测值，降低原本要求的方差。

我们可以采用限制特征的方法来创建不一样的决策树，使贪婪算法能够在建树的同时评估每一个分裂点。这就是随机森林算法（Random Forest algorithm）。

与装袋算法一样，随机森林算法从训练集里撷取复合样本并训练。其不同之处在于，数据在每个分裂点处完全分裂并添加到相应的那棵决策树当中，且可以只考虑用于存储属性的某一固定子集。

对于分类问题，也就是本教程中我们将要探讨的问题，其被考虑用于分裂的属性数量被限定为小于输入特征的数量之平方根。代码如下：

num\_features\_for\_split = sqrt(total\_input\_features)

这个小更改会让生成的决策树各不相同（没有关联），从而使得到的预测值更加多样化。而多样的预测值组合往往会比一棵单一的决策树或者单一的装袋算法有更优的表现。

import pretty\_midi# Load MIDI file into PrettyMIDI object

midi\_data = pretty\_midi.PrettyMIDI('example.mid')# Print an empirical estimate of its global tempoprint midi\_data.estimate\_tempo()# Compute the relative amount of each semitone across the entire song, a proxy for key

total\_velocity = sum(sum(midi\_data.get\_chroma()))print [sum(semitone)/total\_velocity for semitone in midi\_data.get\_chroma()]# Shift all notes up by 5 semitonesfor instrument in midi\_data.instruments:

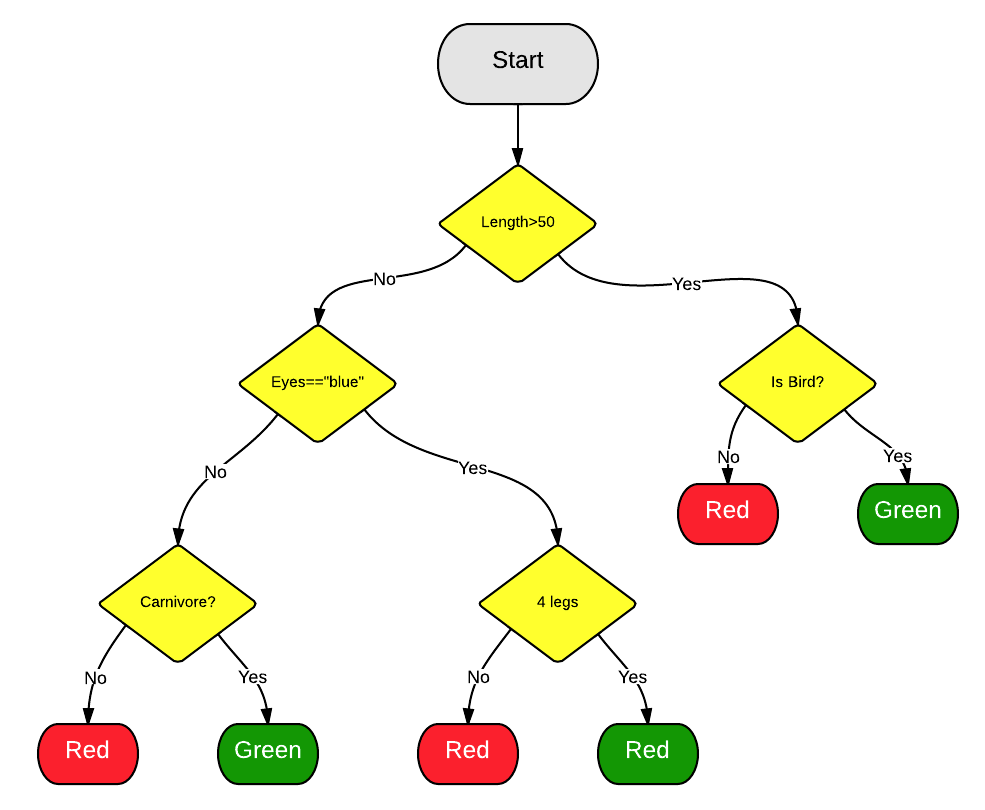
# Don't want to shift drum notes

if not instrument.is\_drum:

for note in instrument.notes:

note.pitch += 5# Synthesize the resulting MIDI data using sine waves

audio\_data = midi\_data.synthesize()

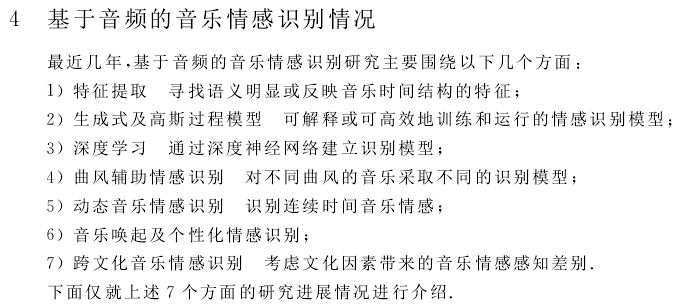


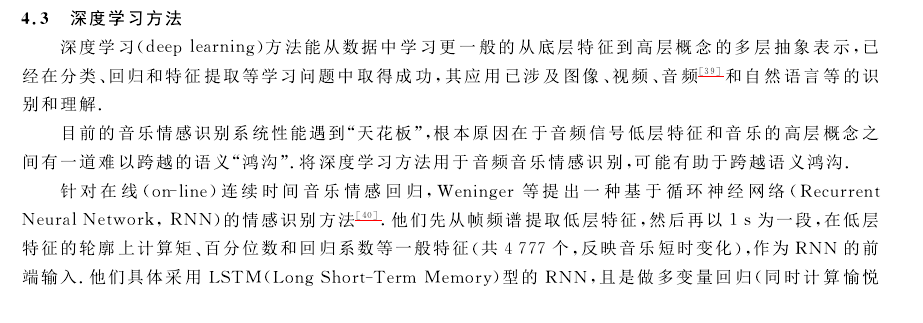
数据集划分：

1.传统的机器学习领域中，由于收集到的数据量往往不多，比较小，所以需要将收集到的数据分为三类：训练集、验证集、测试集。也有人分为两类，就是不需要测试集。

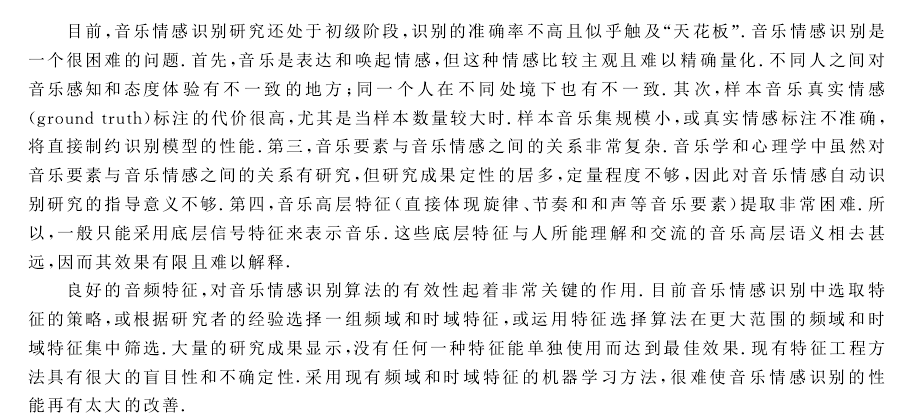
比例根据经验不同而不同，这里给出一个例子，如果是三类，可能是训练集：验证集：测试集=6:2:2；如果是两类，可能是训练集：验证集=7:3。因为数据量不多，所以验证集和测试集需要占的数据比例比较多。

在大数据时代的机器学习或者深度学习领域中，如果还是按照传统的数据划分方式不是十分合理，因为测试集和验证集用于评估模型和选择模型，所需要的数据量和传统的数据量差不多，但是由于收集到的数据远远大于传统机器学习时代的数据量，所以占的比例也就要缩小。比如我们拥有1000000，这么多的数据，训练集：验证集：测试集=98:1:1。如果是两类，也就是相同的道理。









1. 处理MIDI文件的python库

import pretty\_midi# Load MIDI file into PrettyMIDI object

midi\_data = pretty\_midi.PrettyMIDI('example.mid')# Print an empirical estimate of its global tempoprint midi\_data.estimate\_tempo()# Compute the relative amount of each semitone across the entire song, a proxy for key

total\_velocity = sum(sum(midi\_data.get\_chroma()))print [sum(semitone)/total\_velocity for semitone in midi\_data.get\_chroma()]# Shift all notes up by 5 semitonesfor instrument in midi\_data.instruments:

# Don't want to shift drum notes

if not instrument.is\_drum:

for note in instrument.notes:

note.pitch += 5# Synthesize the resulting MIDI data using sine waves

audio\_data = midi\_data.synthesize()

附：

<https://github.com/dokato/musicemotionrecognition>

<https://github.com/vagisha-nidhi/MusicEmotionRecog>