## 目标

《基于情感的音乐分类系统的研究和实现》

**索引**

音乐情感；Adaboost算法；支持向量机；特征提取；情感模型

《基于循环神经网络的音素识别研究》

**索引**

语音识别；连接时序分类；循环神经网络

## 收获

《基于情感的音乐分类系统的研究和实现》

自动化音乐情感分类的研究工作发展过程：

1999年，E.Schuber在他的博士论文中对音乐的情感的时间序列分析研究。他从基音周期、节奏、响度、频谱质心和恩利五类特征探讨了情感与音乐特征之间的关系以及随时间的变化。随后有更多的研究人员加入到这个队伍中。

2006年，Yuan-Yuan shi，XuanZhu等人运用基于abadoost的双层情感分类模型对音乐进行分类，第一层根据音乐信号的强度和时域特征进行分类，第二层则根据音乐信号的节奏特征进行分类，最后根据两层分类器的分类结果以及每层的权值相称并累加得到最后的分类结果。

2007年MIREX正式提出了Audio Mood Emotion这一科研命题，MIR社区特提供了音乐数据库和ground-truth以促进科研工作者之间的合作以及科研结果的评估。

2008年，Yi-Hsuan Yang，Yu-Cheng Lin等人将对音乐进行情感分类归结为回归问题，根据音乐的音质、旋律和节奏特征，采用SVR回归算法计算出每一首歌曲的arousal和valence值，将其定位于Thayer提出的emotion plane中。根据明确的arousal和valance值区间去检索所自己所需要的音乐歌曲。

2010年，Jijun Wang和Kuo Zhang提出采用BP神经网络算法对MIDI音乐文件进行情感分类。Lie Lu，Hong-Jiang Zang采用了GMM机器学习算法对音乐情绪检测并跟踪。

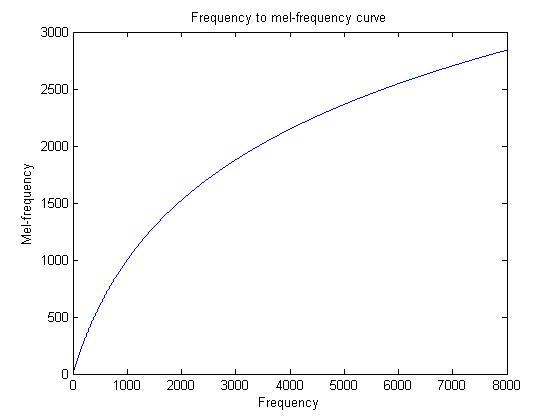
2013年，Yu-Hao Chin，Change-Hong Lin等人提出了双层的SVM模型用于音乐的情感分类。首先将音乐进行特征提取，再将特征输入已经训练好的四种类别的情感模型中，得到四个决策值作为第二层SVM模型的特征，输出最后的分类结果。

对于音乐信号的音色特征提取：

倒谱的音色特征分析

求取音频倒谱特征参数即为对音频信号的倒谱进行分析，这便能够采用同态处理来实现，事先先求音频信号的卷积，即将卷积关系变换为求和关系。通过对音频信号解卷，可以将音频信号的声门激励信息和声道相应信息分离，求得声道共振特征和基音周期，而音频信号解卷的方法主要有两种：一种是线性预测分析，另一种是同态分析处理

1. 梅尔倒谱系数：是由同态分析求出音频信号倒谱的实例。MFCC与普通实际频率分析不同之处在于MFCC分析着眼于人耳的听觉特性。由于人耳听到的声音的高低并不是与声音的频率的高低呈现简单的线性关系，而用梅尔频率尺度则可以反映人耳所听到的音高与音低，更符合人耳的听觉特性。

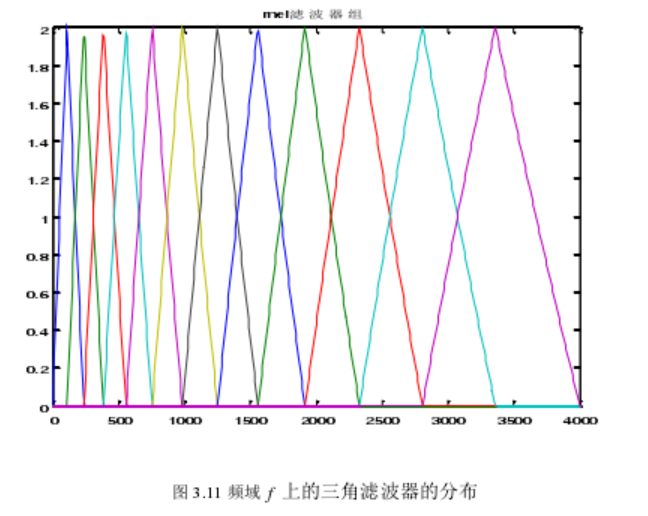


2.梅尔倒谱频率和实际频率f的转换关系

在1000Hz以下呈现线性分布，1000Hz以上呈现对数增长

MFCC计算的总体流程

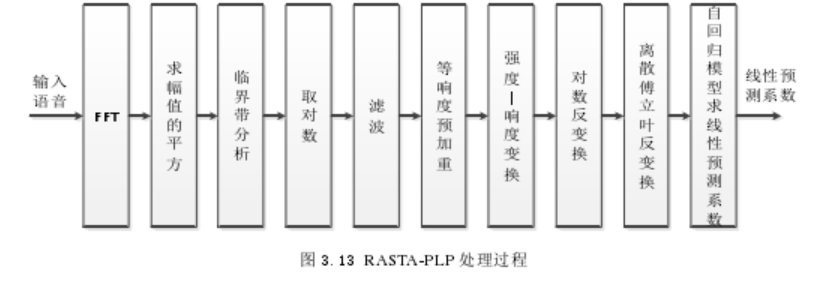




MFCC特征参数的识别与抗噪能力都比较好，不足之处在于其对计算量和计算精度的要求也高。

1. PASTA-PLP

PLP（感知线性预测）等效于LPC特征，不同之处在于PLP是将时域音频信号经过听觉模型处理后得到的信号代替传统的LPC分析所采用的时域信号，而RASTA-PLP在每个PLP频带对数频谱上，使用一个低端截止频率非常低的带通滤波器进行滤波处理来代替通常的短时频谱。这样可以抑制每个频带上任何常量或变化缓慢的部分。



Adaboot与支持向量机分类算法理论基础

**支持向量机**

Boser，Guyon和Vapnik在1992年提出了最优便捷分类器的概念，这便是支持向量机的原型。

Cortes与Vapnik在1993年在最优便捷分类的基础上对不可分情况下的最优便捷分类问提，

1995年，在以上研究的基础之上，Vapnik提出了支持向量机方法。

支持向量机最开始是对两种类别的识别问题，它主要是对线性可分的情况进行分析，假设训练样本数据线性不可分的话，则通过非线性映射将低维特征空间映射到高维，已达到线性可分。

**Adaboost算法原理及概述**

Boosting算法通过组合若干个弱分类器来提高最终的分类性能。Adaboost算法在1995年由Freund和Schapirc提出并完善，Adaboost是一种自适应算法，它在若干轮循环中生成若干个弱分类器，并将它们以不同的权重组合起来形成强分类器。在训练样本每一轮的训练当中，Adaboost会增大在上一轮中训练错误样本的权重，减小训练正确样本的权重。这样就是整个训练过程都侧重于对分类错误样本的训练。理论上来说Adaboost算法可以随着迭代次数的增加无限缩小训练样本的错误率。

1. 弱分类器：对于一个二分类问题，采用随机分类的方法准确率大概在50%左右，若一个分类器。其分类准确率比随机分类器略好，我们就可以成这样的分类器为弱分类器
2. 强分类器：如果一个分类器对整个测试样本进行分类，其准确率能达到很高，我们就称这样的分类器为强分类器
3. 样本权值：Adaboost算法中会在每次迭代过程中更新每个训练样本的权值，增大被错误分类的样本权值，降低正确分类的样本权值，以达到整个训练过程中更关注分类错误样本的目的
4. 弱分类器权值：Adaboost算法根据每个弱分类器的分类错误率来分配权值。最后将权值与分类器求内积，得到最终的强分类器

弱分类器的构造

弱分类器并不是指某种特定类型的分类器，它可以是任何分类器，包括决策树、SVM、高斯混合模型。在Adaboost算法中的每一轮循环中需构造一组弱分类器，并从这组弱分类器中选择性能最好的作为本轮的分类器。但是尽管Adaboost中队弱分类器的要求只要其分类准确率稍微高于随机分类准确率就可以，但是大部分的研究人员都选择决策树作为弱分类器，但Friedman等文献中给出了建议：采用基于树桩的弱分类器时，boosting效果更好。树桩是指深度为一，仅有一个分裂点，两个叶子节点的决策树。

《基于循环神经网络的音素识别研究》

CTC（连续时间分类）使得RNN可以直接进行序列数据分类的任务，移除了语音数据预对齐的工作，而且CTC直接输出所有预测序列的概率，不需要后补的额外处理。

使用RNN进行语音识别的任务，首先得进行语音预处理，在分帧的过程中，为了使每一帧能够平滑过渡，相邻的帧之间有重合部分，而由于RNN的特殊结构，每个节点的输出和相邻节点的输入相关，所以通过去除帧之间的重合部分，使得在神经网络的通过输入序列的长度减少，大幅降低训练时间。

在计算第一个节点的隐层输出的时候，使用元素全为0的向量来初始化，如果是多分类分体，则在计算出Ot后增加一部softmadx的计算：yt=sofmax(Ot+by);

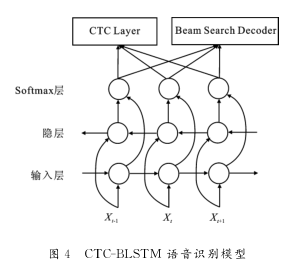
而标准的RNN（LSTM）以时序来处理序列数据从而忽略了上下文，BRNN的概念在1997年被提出，BRNN的基本思想是让训练序列分别通过一层正向的隐层和一层逆向的隐层，这两个分开的隐层都连接到输出层，通过正向计算得到的st你和逆向计算得到的st计算输出。

RNN 擅长处理序列数据，但是对于语音识别的任务来说，使用 RNN 进行此类任务需要预先对语音数据进行分段，而且需要对语音的输出标签进行预处理，前者好解决，但是后者工作量则巨大．语音识别声学模型的训练属于监督学习，需要知道每一帧对应的标签才能进行有效的训练，使用连接时序分类（ CTC ）则不用考虑上面的问题，只要一个输入序列和输出序列就能进行有效的训练， CTC直接输出预测序列的概率，不需要外部的后处理．CTC的其他不同之处在于引进了一个BLANK（一）标签，表示空白，去掉输出中连续重复的元素和空白，就能得到标签，输出和标签满足以下的等价关系：

F（a—ab—）=F(—aa—abb)=aab

CTC接在RNN的softmax层之后，RNN每个节点的softmax层zhihou ,RNN每个节点的softmax层都能得到一个向量，表示各个标签在此刻的概率，根据每个节点的各个标签的条件概率，可以计算出每一条可能路径的条件概率。

CTC-BLSTM的语音识别模型：

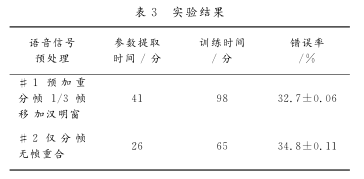


在训练阶段将softmax层的输出作为CTC层的输入，通过CTC算法计算出损失值，然后通过BPTT（反向传播）算法，调整网络的参数，在测试阶段则通过集束搜索方法，対softmax层的输出进行解码，得到对应的输出序列。

实验结果

神经网络使用BLSTM，隐层数量为1层，隐层单元数量为140，神经网络的输入单元数量为39，输出单元也为39，学习率为0.008





RNN可以通过去除分帧时的相邻帧重合部分，而使得训练时间变短，并且错误率并无大幅的提升。

结论：RNN可以直接用来进行序列数据的建模，搭配用于序列建模的连接时序分类，不需要对语音预先对齐，而且效果也要好于HMM和LSTM的混合模型，并且在语音与处理方面，可以适当去减少或者直接省去帧与帧之间的重叠部分，从而减少整个系统的训练时间

基于Python2.7，但是因为电脑上有些库安装不上去，所以无法自己尝试运行

Music-classification

|  |
| --- |
| #coding:utf-8 |
|  |

|  |
| --- |
| import os |

|  |
| --- |
| import sys |

|  |  |
| --- | --- |
| import numpy as np | |
| import sklearn |
| from sklearn.linear\_model import LogisticRegression | | |

|  |
| --- |
| from sklearn.cross\_validation import cross\_val\_score |

|  |
| --- |
| from sklearn.cross\_validation import train\_test\_split |

|  |
| --- |
| from sklearn.ensemble import RandomForestClassifier |

|  |
| --- |
| from sklearn.metrics import classification\_report, roc\_auc\_score, precision\_recall\_curve, auc, roc\_curve |

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_': |

|  |
| --- |
| data = np.load('data.npy') |

|  |
| --- |
| labels=np.load('label.npy') |
|  |

|  |
| --- |
| #交叉验证 |
| accuracys=[] | |

|  |  |  |
| --- | --- | --- |
| for i in range(10): | | |
| X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data, labels, test\_size=0.1, random\_state=i) | | | |
| lr = LogisticRegression() | |
| lr.fit(X\_train, Y\_train) |

|  |
| --- |
| print "====%d====" % (i) |

|  |
| --- |
| print "Accuracy:", lr.score(X\_test, Y\_test) |

|  |
| --- |
| predict=lr.predict(X\_test) |

print classification\_report(Y\_test, predict, target\_names=["classical", "country", "jazz", "metal", "pop", "rock"])

music\_fearure\_extrat

|  |
| --- |
| #coding:utf-8 |

|  |
| --- |
| import os |

|  |
| --- |
| import sys |

|  |
| --- |
| import numpy as np |

|  |
| --- |
| import librosa |

|  |
| --- |
| import sklearn |
| from matplotlib.pyplot import specgram | |

|  |
| --- |
| from sklearn.multiclass import OneVsRestClassifier |

|  |
| --- |
| from sklearn.svm import SVC |

|  |
| --- |
| from sklearn.cross\_validation import train\_test\_split |

|  |
| --- |
| from sklearn.metrics import accuracy\_score |

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_': |

|  |
| --- |
| path='./genres/' |

|  |
| --- |
| data=[] |

|  |
| --- |
| labels=[] |

|  |
| --- |
| for i, folder in enumerate(os.listdir(path)): |

|  |
| --- |
| for file in os.listdir(path+folder): |

|  |
| --- |
| music, fs = librosa.audio.load(path+folder+'/'+file) |
|  |

|  |
| --- |
| mfcc\_feature = librosa.feature.mfcc(music,n\_mfcc=13) |

|  |
| --- |
| #使用前10％和最后10％的框架 |

|  |
| --- |
| mfcc\_feature = np.array([x[int(len(x)\*1/10):int(len(x)\*9/10)] for x in mfcc\_feature]) |

|  |
| --- |
| data.append(mfcc\_feature.mean(axis=1)) |

|  |
| --- |
| labels.append(folder) |

|  |
| --- |
| np.save('data.npy', data) |

np.save('label.npy', labels)