## 目标

《基于双向情感分析的实时性音乐推荐系统设计》

**索引**

情感分析；推荐系统；贝叶斯分类器

《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

## 收获

《基于双向情感分析的实时性音乐推荐系统设计》

这篇文章采取了双向情感分析的方法，一方基于梅尔频率倒谱系数对音乐情感进行分类，另一方面通过用户在社交媒体上的语言文字的情感判断感知用户的心情，然后通过贝叶斯分类器将两部分的情感分析进行整合和匹配，完成契合用户心情的音乐推荐。

这个音乐系统可分为三个模块：用户音乐情感分类、用户社交网络情感分类和推荐系统界面，其中用户音乐情感和用户社交网络情感分为四个维度：喜悦、励志、躁动和悲伤。

音乐情感分类：

先对每首歌的音乐数据进行预处理，分别使用歌曲的频域特征和梅尔倒谱特征来设计分类器，测试和评估分类效果。在基于不同音频特征构建的分类器结果进行比较后，选取梅尔频率倒谱系数这一准确率较高的特征作为最终分类的依据，完成音乐情感分类。

实现过程

1. 音乐数据预处理

该篇文章发表时，在音乐情感识别领域还未有公开的数据集，但现在已经有了一个公开的数据集：Million Song Dataset(MSD)。链接：<https://labrosa.ee.columbia.edu/millionsong/>

1. 人工标定音乐情感分类
2. 划分训练集和测试集，将预处理后的WAV文件按照7:3分成两个子集，前者作为训练集，用于构建分类器，后者作为测试集，用来评估分类器的性能
3. 音乐特征提取。根据分析的角度，可以对音乐信号进行时域分析和频域分析等，时域特征提取方式的优点是：计算量较小，提取较容易，但不足之处是鲁棒性差（是指控制系统在一定（结构，大小）的参数摄动下，维持其它某些性能的特性）。始于参数包括音乐的短时能量，短时自相关函数，短时过零率等，音乐信号的功率谱可从多方面来反映音乐的情感特性。
4. 训练分类器：采用逻辑分类器进行音乐分类
5. 测试并评估分类结果。将测试集作为输入，巡行分类器，得到最终的分类结果并进行评价

音频信号特征提取过程

从频域特征参数的分析角度，分别提取每首歌曲的频谱特征和梅尔倒谱特征。对于音频信号等数字信号的处理过程，离散傅里叶变换是常用的变换方法。使用快速傅里叶变换可分析信号的频谱、计数滤波器的频率响应和实现信号通过线性系统的卷积运算等。

梅尔频率倒谱系数（MFCC）是标度提取的倒谱参数，描述了音频信号在频域上的非线性特征。原始的频率特征由两部分组成：包络和频谱的细节，若从低频到高频按临界带宽的大小依次设置一组带通滤波器，对输入信号进行滤波，再将每个带通滤波器的输出信号的能量作为信号的基本特征，就可以提取出梅尔频率倒谱系数。由于该特征不依懒于信号的性质，因此拥有良好的识别性和抗噪能力，更符合人耳的听觉特性。

MFCC参数提取基本流程如下：

1. 将连续音乐信号s(n)经过预加重，分帧及加窗得到一帧音乐信号x(n)。
2. 将x(n)信号进行快速傅里叶变换（FFT）得到品与信号，即线性频谱X(k),X(k)=,k=0,1,2,…，N-1.其中x(n)为输入语音信号，N为傅里叶变换的点数
3. 将线性频谱X(k)输入三角滤波器组Hm(k),,M为滤波器的个数。每个滤波器的中心频率为f(m)。三角滤波器组的中心频率在梅尔频率内的间隔是相等的。由此可以计算两个相邻的三角带通滤波器的中心频率距离，每个带通滤波器组传递参数为Hm(k)=
4. Mel坐标上的能量谱s(m)经滤波器组输出为：

S(m)=ln(),0

1. 经离散余弦反变换，得到频率上的Mel频率倒谱C(n)

C(n)=。其中L是指MFCC系数的阶数。

音乐感情分类的评估

利用梅尔倒谱特征和频谱特征以后，就可以利用训练集来训练分类器，最后使用测试集数据评估分类结果。

混淆矩阵对分类结果进行可视化。混淆矩阵是用于比较分类结果准确性的一种可视化工具，通过比较每个实测单元与预测分类图像的相应位置，对应单元模块的颜色浓度直观显示分类结果。对于完美的分类器，从左上角到右下角都是深色的方格，其余地方都是浅色甚至是无色



横轴代表预测分类结果，纵轴代表实际类别结果。

《基于情感的音乐分类系统的研究和实现》

首先先把音乐的情感分成四种类别：平静的、激昂的、愉悦的、悲伤的四中基本情感类别。提出了双层的音乐情感分类系统结构，即首先根据音乐的节奏来划分，激昂的、喜悦的属于节奏快的类别，而悲伤的和平静的属于节奏慢的类别，之后进行第二次分类，若测试歌曲节奏快，则将激昂的和喜悦的进行分类，若测试歌曲节奏较慢，则将平静的悲伤的进行分类。

主要步骤为：特征提取、训练分类模型和测试环节

音乐情感受两部分影响1.音乐片段特征提取的准确率，其中第一层的正确率最为重要，若第一层的正确率偏低会严重影响第二层的正确率。2.音乐片段特征的选取

而音乐的情感拥有很高的主观性，层次性、运动性、模糊性、选择性、客观性等等

目前在音乐情感识别与分类领域比较常提到的情感模型分别为：文本关键字模型、基本情绪理论模型和声学参考模型。其中，文本关键字模型是采用一些描述音乐情感的形容词关键字作为情感类别来建立情感模型，代表是Hevner情感环；基本情绪理论源自三原色原理，就是指人类的大部分情绪都可以有几种基本情绪以下不同比例组合而成，Ekman的基本情绪理论；而声学参考模型就是音乐特征映射到Arousal和Valence两个维度来量化音乐的情感，（Arousal维是从“非常活跃”到“平静”，Valence维度是指人们所感受到的音乐作品中的情感的积极性与消极性不同程度的反应）代表是Thayer的二维情感模型。

音乐的特征大致可以分为两大类：一类是音色特征，一类的韵律特征，这两类的特征提取都是音乐信号在时域和频域上所提取的特征参数的整合。

2011年有徐欣、周运和邵曦发表的音乐情感提取一文中对不同情感类别的频谱进行分析得出以下结论

振奋（excited）：音调较高、振幅也比较高、过零率在两次以上。

欢快（pleased）：音调高，频谱相对平稳但比较密集，振幅很高，过零率在一次左右。

平静：音调较高但相对平稳，振幅相对高，过零率在两次左右。

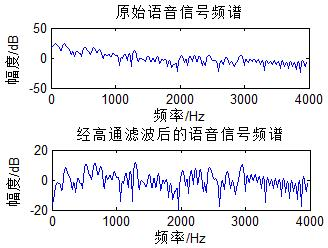
音乐信号是一种非平稳信号，但是这类物理信号比声音振动速度要慢得多，所以我们可以把它看做一个短时间内的平稳信号。音乐信号的鱼预处理包括预加重和加窗分帧等。

预加重：

原因：在高频部分，因为一些神门激励和口鼻辐射的影响，在800Hz以上的高频段以6db/倍频程跌落，为了是音乐信号在着呢哥哥频带中能够用相同的信噪比求频率，所以需要对信号的高频部分强化。

而通过预加重，可以是音乐信号的频谱变的更加平坦

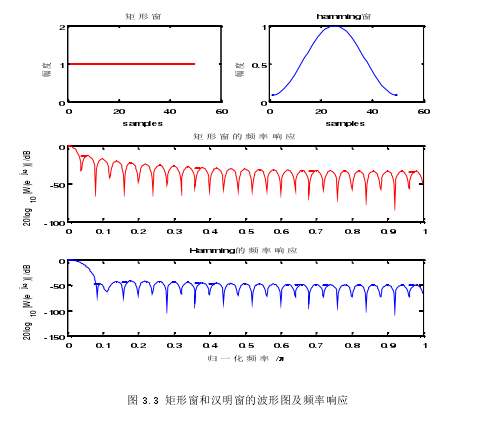
流程：首先对音乐信号进行数字化处理，再用6db/倍频程的提升高频特性的预加重数字滤波器来滤波。一般采用一阶数字滤波器：H（z）=1-μz-1，其中μ接近于1



加窗处理：通常采用移动的有现场的窗口进行加权的方法来试下先音乐信号分帧，每秒钟大概可以分为33-100帧，分帧时可采取连续分段或者是交叠分段，而交叠分段优点是前后两帧之间平滑过渡，保持连续性，前后帧的重叠部分成为帧移，帧移和帧长的比值一般为0~1/2

用的最多的两种窗函数：

1. 矩形窗w(n)=
2. 汉明窗 w(n)=

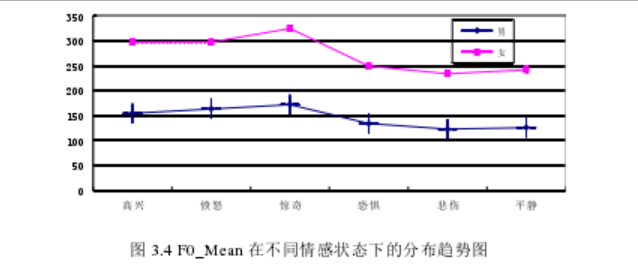


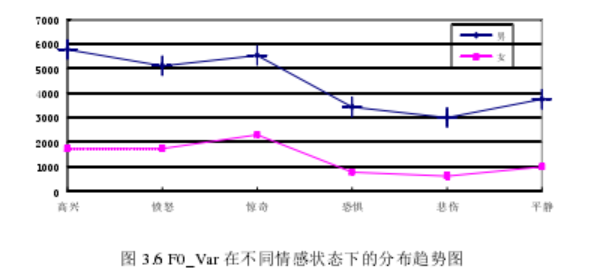
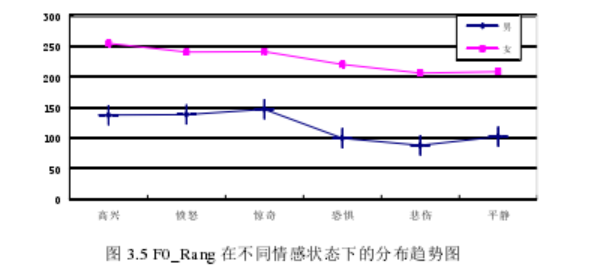
区别：矩形窗主瓣宽度小，但旁瓣宽度高；汉明窗主瓣宽度宽，而旁瓣高度低

矩形窗的旁瓣过高，则会发生很严重的泄漏现象，汉明窗的旁瓣过低，能够有效的客服GIBBS，具有更平滑的低通特性

基频特性分析

人在发浊音是，气流通过声门使声带产生张弛振荡式振动，产生一股准周期脉冲气流，这一气流激励声道就产生浊音，又称为有声语言，它携带者语音中大部分的能量，这种声带振动的频率称为基频



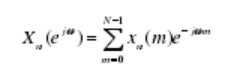


基轮轮廓的凹凸特性也与音乐信号的严肃和柔和程度紧密相关

音乐信号的音色特征提取和分析：

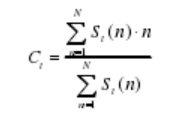
经过很多研究这表明：音乐信号的短时频谱和倒谱可以有效的反映出音乐信号的音色。

计算公式



子带特征：可先将每一帧的短时频谱划分为7个子带，每个子带包含多个频谱分量，激昂每个子带的最小频谱分量幅度值、最大频谱分量幅度值和频谱分量均值作为一个特征值，这样每个子带有3个特征值，7个子带共有21维特征值，此特征值能够有效反映出频谱幅度值在每个子带的分布情况。

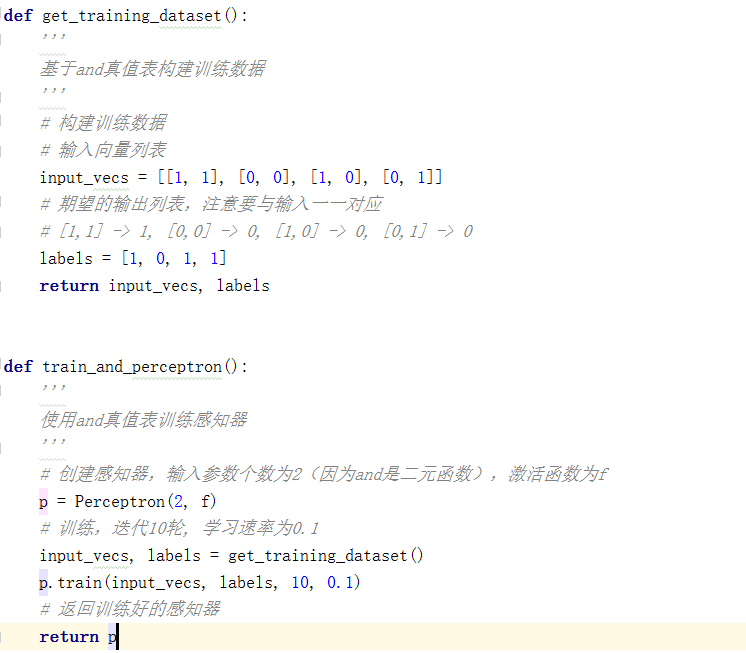
谱质心：它表示个频谱幅度值得加权平均值，相当于频谱分布的“质心”，计算公式如下

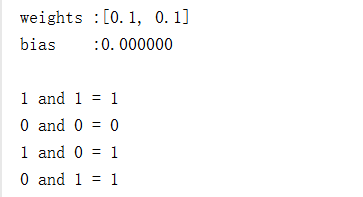
 St(n)表示第t帧音频的短时傅里叶变换的幅度值。一般愉快的，激昂的歌曲的高频分量会多一些，所以谱质心也偏高一些，而悲伤的平静的歌曲中低频分量多，高频分量偏少，所以谱质心也偏低。

倒谱的音色特征分析

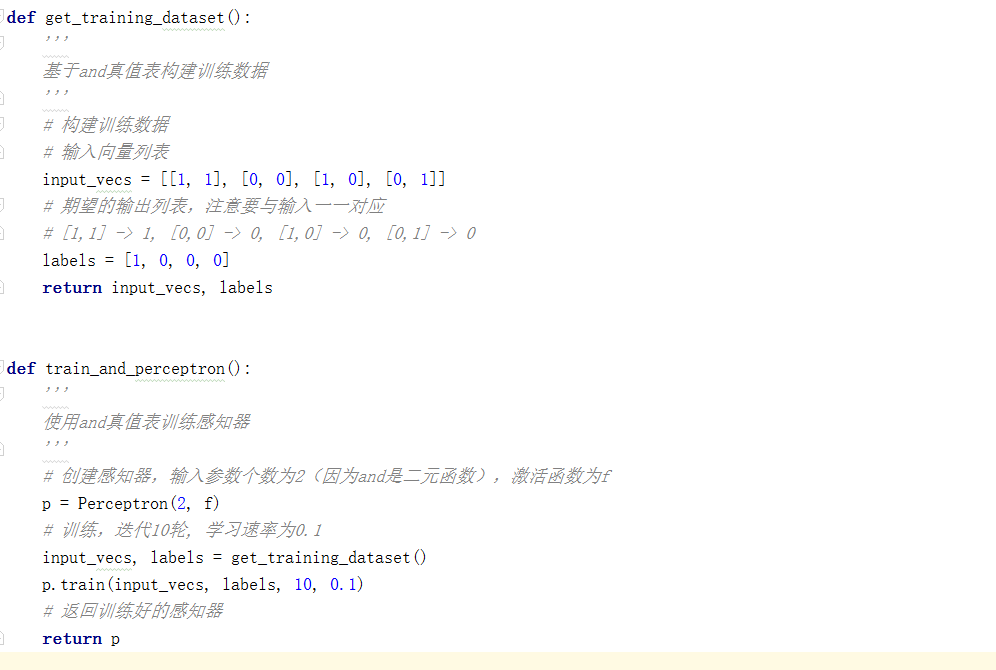
通过对音频信号解卷，可以将音频信号的声门激励信号和声道相应信息分离，求得升共振特征和基音周期。

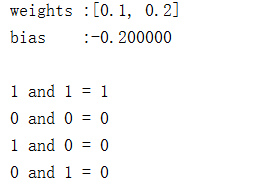
And功能



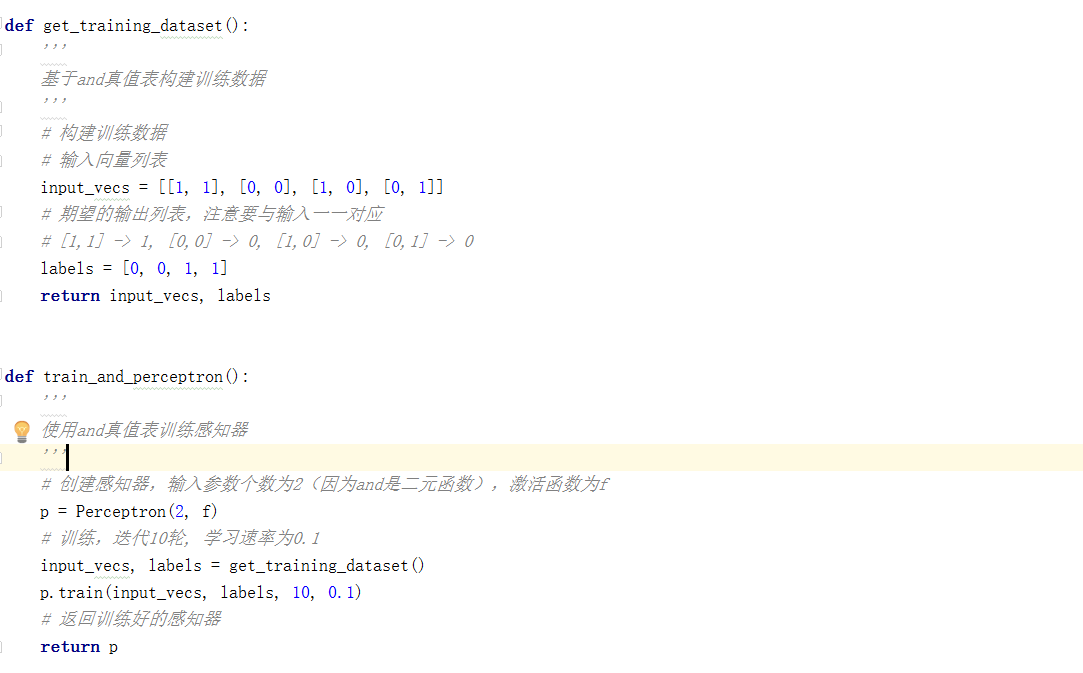


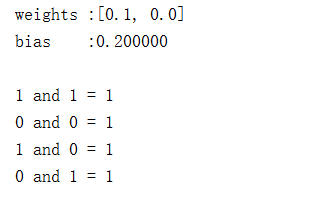
Or功能

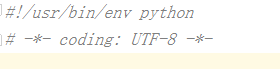




实现亦或





在需要输入注释中文的程序中，在开头加入

与门代码

*#!/usr/bin/env python  
# -\*- coding: UTF-8 -\*-***class** Perceptron(object):  
 **def** \_\_init\_\_(self, input\_num, activator):  
 *'''  
 初始化感知器，设置输入参数的个数，以及激活函数。  
 激活函数的类型为double -> double  
 '''* self.activator = activator  
 *# 权重向量初始化为0* self.weights = [0.0 **for** \_ **in** range(input\_num)]  
 *# 偏置项初始化为0* self.bias = 0.0  
  
 **def** \_\_str\_\_(self):  
 *'''  
 打印学习到的权重、偏置项  
 '''* **return 'weights\t:%s\nbias\t:%f\n'** % (self.weights, self.bias)  
  
 **def** predict(self, input\_vec):  
 *'''  
 输入向量，输出感知器的计算结果  
 '''  
 # 把input\_vec[x1,x2,x3...]和weights[w1,w2,w3,...]打包在一起  
 # 变成[(x1,w1),(x2,w2),(x3,w3),...]  
 # 然后利用map函数计算[x1\*w1, x2\*w2, x3\*w3]  
 # 最后利用reduce求和* **return** self.activator(  
 reduce(**lambda** a, b: a + b,  
 map(**lambda** (x, w): x \* w,  
 zip(input\_vec, self.weights))  
 , 0.0) + self.bias)  
  
 **def** train(self, input\_vecs, labels, iteration, rate):  
 *'''  
 输入训练数据：一组向量、与每个向量对应的label；以及训练轮数、学习率  
 '''* **for** i **in** range(iteration):  
 self.\_one\_iteration(input\_vecs, labels, rate)  
  
 **def** \_one\_iteration(self, input\_vecs, labels, rate):  
 *'''  
 一次迭代，把所有的训练数据过一遍  
 '''  
 # 把输入和输出打包在一起，成为样本的列表[(input\_vec, label), ...]  
 # 而每个训练样本是(input\_vec, label)* samples = zip(input\_vecs, labels)  
 *# 对每个样本，按照感知器规则更新权重* **for** (input\_vec, label) **in** samples:  
 *# 计算感知器在当前权重下的输出* output = self.predict(input\_vec)  
 *# 更新权重* self.\_update\_weights(input\_vec, output, label, rate)  
  
 **def** \_update\_weights(self, input\_vec, output, label, rate):  
 *'''  
 按照感知器规则更新权重  
 '''  
 # 把input\_vec[x1,x2,x3,...]和weights[w1,w2,w3,...]打包在一起  
 # 变成[(x1,w1),(x2,w2),(x3,w3),...]  
 # 然后利用感知器规则更新权重* delta = label - output  
 self.weights = map(  
 **lambda** (x, w): w + rate \* delta \* x,  
 zip(input\_vec, self.weights))  
 *# 更新bias* self.bias += rate \* delta  
  
  
**def** f(x):  
 *'''  
 定义激活函数f  
 '''* **return** 1 **if** x > 0 **else** 0  
  
  
**def** get\_training\_dataset():  
 *'''  
 基于and真值表构建训练数据  
 '''  
 # 构建训练数据  
 # 输入向量列表* input\_vecs = [[1, 1], [0, 0], [1, 0], [0, 1]]  
 *# 期望的输出列表，注意要与输入一一对应  
 # [1,1] -> 1, [0,0] -> 0, [1,0] -> 0, [0,1] -> 0* labels = [1, 0, 0, 0]  
 **return** input\_vecs, labels  
  
  
**def** train\_and\_perceptron():  
 *'''  
 使用and真值表训练感知器  
 '''  
 # 创建感知器，输入参数个数为2（因为and是二元函数），激活函数为f* p = Perceptron(2, f)  
 *# 训练，迭代10轮, 学习速率为0.1* input\_vecs, labels = get\_training\_dataset()  
 p.train(input\_vecs, labels, 10, 0.1)  
 *# 返回训练好的感知器* **return** p  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# 训练and感知器* and\_perception = train\_and\_perceptron()  
 *# 打印训练获得的权重* **print** and\_perception  
 *# 测试* **print '1 and 1 = %d'** % and\_perception.predict([1, 1])  
 **print '0 and 0 = %d'** % and\_perception.predict([0, 0])  
 **print '1 and 0 = %d'** % and\_perception.predict([1, 0])  
 **print '0 and 1 = %d'** % and\_perception.predict([0, 1])