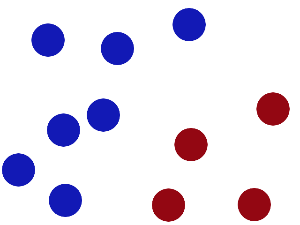
重新分组后分到了音乐分类小组，所以这周看了两篇文章，看的一篇文章是《特征组合的中文音乐情感识别研究》，其中需要的一种分类的算法是支持向量机，所以先去知乎了解了关于支持向量机的算法：<https://www.zhihu.com/question/21094489>

1. 关于支持向量机算法

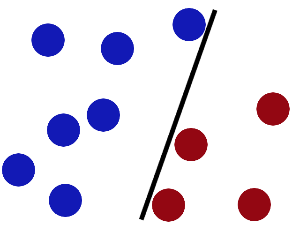
在看到的的内容中有一个很形象的解释，让我们了解到了什么叫支持向量机，不是一种机器，而是分类中的其中一种算法，知乎上是一个很有趣的解释：

在很久以前的情人节，大侠要去救他的爱人，但魔鬼和他玩了一个游戏。

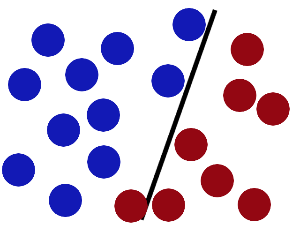
魔鬼在桌子上似乎有规律放了两种颜色的球，说：“你用一根棍分开它们？要求：尽量在放更多球之后，仍然适用。”



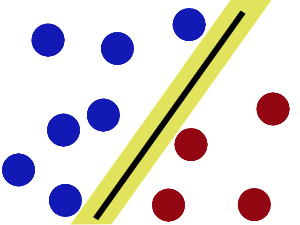
于是大侠这样放，干的不错？



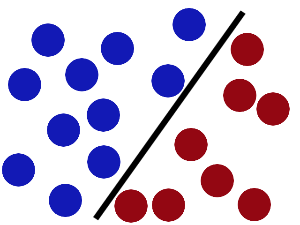
然后魔鬼，又在桌上放了更多的球，似乎有一个球站错了阵营。



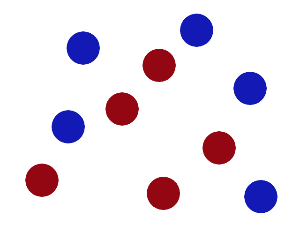
**SVM就是试图把棍放在最佳位置，好让在棍的两边有尽可能大的间隙。**



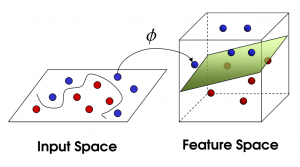
现在即使魔鬼放了更多的球，棍仍然是一个好的分界线。



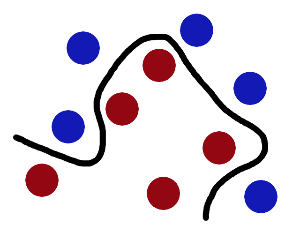
然后，在SVM 工具箱中有另一个更加重要的 trick（方法）。 魔鬼看到大侠已经学会了一个trick（方法），于是魔鬼给了大侠一个新的挑战。



现在，大侠没有棍可以很好帮他分开两种球了，现在怎么办呢？当然像所有武侠片中一样大侠桌子一拍，球飞到空中。然后，凭借大侠的轻功，大侠抓起一张纸，插到了两种球的中间。



现在，从魔鬼的角度看这些球，这些球看起来像是被一条曲线分开了。



再之后，无聊的大人们，把这些球叫做 **data（数据，类）**，把棍子 叫做 **classifier（分类）**, 最大间隙trick （**在两个类的中间，距离两个类的点都一样远**）叫做**optimization（优化）**， 拍桌子叫做**kernelling（核函数，即根据我们的模型选用适用的核函数将多维空间数据进行分类）**, 那张纸叫做**hyperplane（超平面）**。

从上边有趣的故事我们局能基本的了解到了支持向量机（SVM），训练数据时，支持向量机接受这些数据，将不同的数据在平面或超平面中进行分类，当数据是线性可分时，我们就能通过一条简单的直线来划分这些数据，并且寻找一条最优的分界线使得离该直线最近的元素距离最远，而距离该直线最近的元素就是支持向量。

当数据是非线性可分，无法找出一个线性决策边界时，就需要引入多个维度，即核函数，讲一个低维度映射到高纬度，这样决策边界就有可能变成了平面或者其他，因此，支持向量机就是分类中的一个决策（分类边界）。

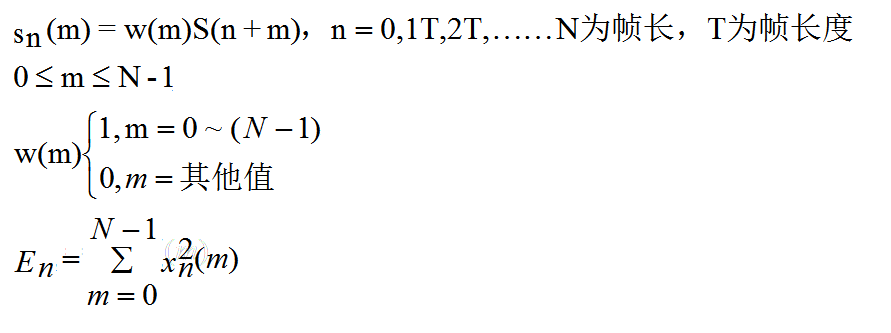
二、《特征组合的中文音乐情感识别研究》文章收获

目的：研究不同音频特征组合在音乐情感分类上的效果。

方法：在网上寻找不同的中文歌曲，并将中文歌曲分为**开心**和**悲伤**两类。通过分析音乐的音频声学特征,尝试提取代表音乐声学特性的时域、频域、倒谱域的各种特征,使用提取到的该音乐特征数据进行组合来构建该音乐情感识别的训练和测试样本，并利用支持向量机(support vector machine,简称 SVM)算法对重新用特征值进行组合的音乐进行情感分类,然后观察不同的特征组合的结果谁更加接近原来的音乐情感。

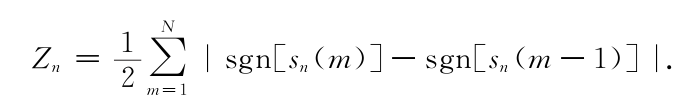
结论：通过对比各种不同特征组合的音乐情感识别效果,发现**由 4 个时域特征、频谱、幅度谱和相位谱组成的音乐特征对中文音乐情感分类的效果良好**。

音乐声学特性的各个特征介绍：

1. 1时域特征
2. 短时能量：音频信号的能量是时间的函数，用来检测浊音，因为浊音的能量比清音大，第 n 帧的音频信号的短时能量定义为：

（图片中的公式最后一行X2n(m)应为S2n（m）,敲错）

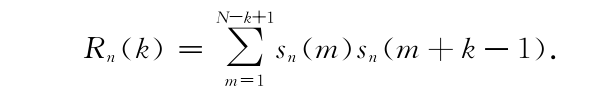
(2)短时过零率:一帧音频的时域信号波形穿过零电平的次数,高频段过零率较高,低频段过零率较低.短时过零率可以用来区分清音和浊音，第 n 帧的短时过零率定义：



1. 短时平均幅度:度量音频信号幅度值变化的函数,与短时能量的区别是没有对音频的时域信号取平方,故不会对高频敏感,第 n 帧音频信号的短时平均幅度定义为：



1. 短时自相关函数：提供了一种获取信号周期的方法,它是在信号的第m 个样本点附近用短时窗截取一段信号,做自相关计算所得的结果.第 n 帧短时自相关函数的定义为：



1.2频域特征：将时域信号经过傅里叶变换至频域以得到其幅度谱、相位谱、功率谱和对数谱

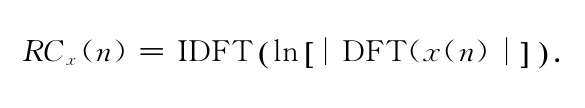


1.3倒频域特征

1. 复倒谱变换定义：时域信号的傅里叶变换的**对数**的傅里叶反变换



1. 倒谱变换的定义：时域信号的傅里叶变换的**幅值求自然对数**,然后再做傅里叶逆变换



2.1 SVM理论

论文中的总结与自己的相比比较正式化，即SVM 的原理是用分类超平面将空间中两类样本点正确分离,并取得最大边缘(正样本与负样本到超平面的最小距离)这样,原问题为一个有约束非线性规划问题

3.1音乐特征截取过程

音乐情感通过其基本特征得以表现，特征提取的结果进一步影响对wav格式的音频信号分析及其后续研究工作，因此特征是实现音乐情感识别的重要环节

论文中音乐情感分析的音频来源：由于歌曲的高潮部分情感特性较为明显，因此文中先过获取歌曲总时长 t，截取歌曲从歌曲 t/4地方开始到歌曲 t/2 地方的音频段获取歌曲的高潮部分，此为情感分析来源

步骤：

步骤 1 将歌曲格式全部转换成 wav 格式.

步骤 2 截取歌曲的高潮部分,即截取歌曲 t/4 到 t/2 的音频段(t 为歌曲总时长).

步骤 3 对截取的高潮音频段提取各种音频特征.提取时域特征前需对音频段进行分帧处理,其中

帧长为 fix(FS/2 20),FS=44 1 00 Hz,帧移与帧长长度相同.

步骤 3.1 提取短时能量特征;

步骤 3.2 提取短时过零率特征;

步骤 3.3 提取短时平均幅度特征;

步骤 3.4 提取短时自相关函数特征,求基音周期 T,并求出去除野点后的基音周期 T 1 =medfilt1(T,5);

步骤 3.5 求频域特征:

频谱：X = fft(x,4 0 9 6),

幅度谱：magX = abs(X),

相位谱：angX = angle(X),

功率谱：power = magX2 ,

对数谱：ln = log(magX);

求复倒谱特征：z = cceps(x(4 000:4 9 9 9)),

倒谱：rz = rceps(x(4 000:4 9 9 9)).

步骤 4 对所得的特征进行求和、求均值、求方差、求中值处理

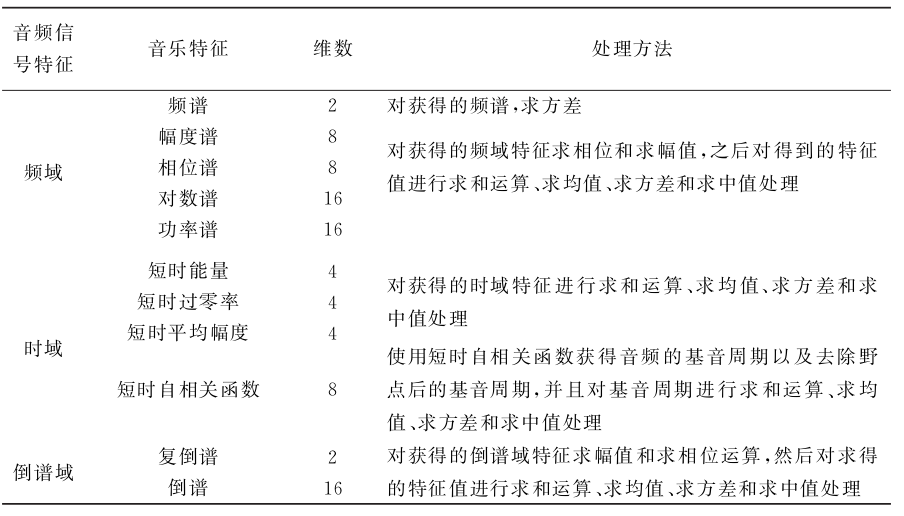
（以上方法应该可以直接在matlab中实现）

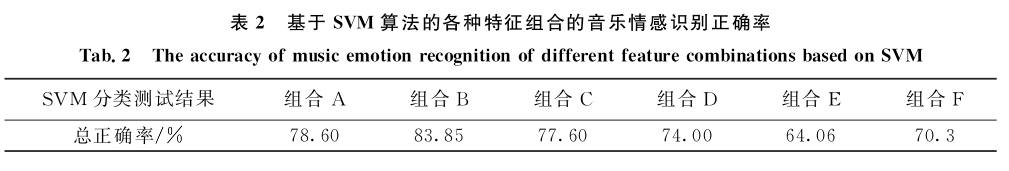
步骤 5 尝试用已提取出的所有音频特征值进行特征组合来构建训练和测试样本.

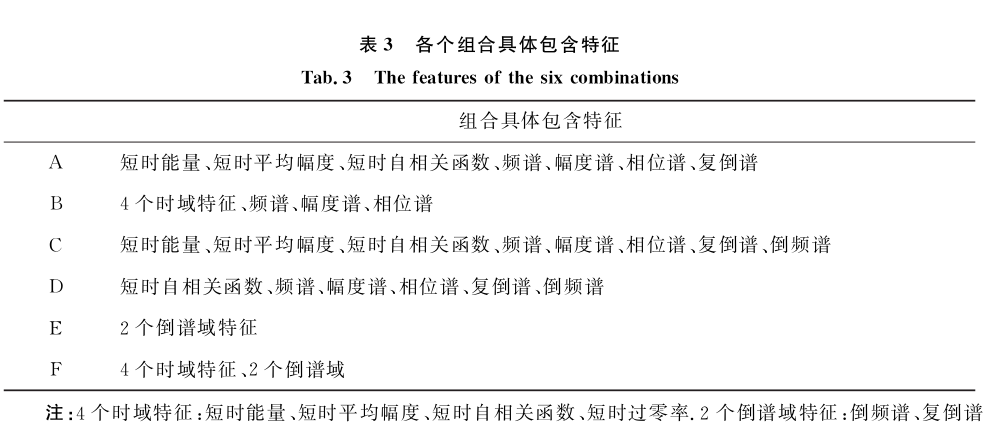
步骤 6 训练基于支持向量机算法的音乐情感分类器.

步骤 7 使用已训练好的音乐情感分类器来对测试样本进行音乐情感分类.

步骤 8 重复步骤 1~7,对比不同特征组合在音乐情感分类上的效果.







以上图片为该论文的测试结果。

仿真

发现了一个有关机器学习的网络，其中有很多关于神经网络的源码

<https://ask.hellobi.com/blog/guodongwei1991/9009>

找到了网上的一个感知机实现的例子，感知机是一种最简单，最基本的线性分类算法（二分类）。其前提是数据本身是线性可分的。

*#coding = utf-8***import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
*#确定一个画图函数，规定其中的标题，坐标尺寸，数据***class showPicture:  
 def** \_\_init\_\_(self,*data*,*w*,*b*)**:** self.b **=** *b* self.w **=** *w* plt.figure(1)  
 plt.title(**'Plot 1'**, size**=**14)  
 plt.xlabel(**'x-axis'**, size**=**14)  
 plt.ylabel(**'y-axis'**, size**=**14)  
 xData **=** np.linspace(0, 5, 100)  
 yData **=** self.expression(xData)  
 plt.plot(xData, yData, color**='r'**, label**='y1 data'**)  
 plt.scatter(*data*[0][0],*data*[0][1],s**=**50)  
 plt.scatter(*data*[1][0],*data*[1][1],s**=**50)  
 plt.scatter(*data*[2][0],*data*[2][1],marker**='x'**,s**=**50,)  
 plt.savefig(**'2d.png'**,dpi**=**75)  
 *#初始化w0，b0，确定初始化超平面，并确定各样例点是否正确分类（利用yi和wx+b的正负性关系）* **def expression**(self,*x*)**:** y **=** (**-**self.b **-** self.w[0]**\****x*)**/**self.w[1]  
 **return** y  
 **def show**(self)**:** plt.show()  
**class perceptron:  
 def** \_\_init\_\_(self,*x*,*y*,*a***=**1)**:** self.x **=** *x* self.y **=** *y* self.w **=** np.zeros((*x*.shape[1],1))  
 self.b **=** 0  
 self.a **=** 1  
 **def sign**(self,*w*,*b*,*x*)**:** result **=** 0  
 y **=** np.dot(*x*,*w*)**+***b* **return** int(y)  
 *#开始训练数据* **def train**(self)**:** flag **= True** length **=** len(self.x)  
 **while** flag**:** count **=** 0  
 **for** i **in** range(length)**:** tmpY **=** self.sign(self.w,self.b,self.x[i,**:**])  
 **if** tmpY**\***self.y[i]**<=**0**:** tmp **=** self.y[i]**\***self.a**\***self.x[i,**:**]  
 tmp **=** tmp.reshape(self.w.shape)  
 self.w **=** tmp **+**self.w  
 self.b **=** self.b **+** self.y[i]  
 count **+=**1  
 **if** count **==** 0**:** flag **= False  
 return** self.w,self.b  
*#原始数据*data **=** [[3,3],[4,3],[1,1]]  
xArray **=** np.array([3,3,4,3,1,1])  
xArray **=** xArray.reshape((3,2))  
yArray **=** np.array([1,1,**-**1])  
*#感知机计算权值*myPerceptron **=** perceptron(x**=**xArray,y**=**yArray)  
weight,bias **=** myPerceptron.train()  
*#画图*picture **=** showPicture(data,w**=**weight,b**=**bias)  
picture.show()

