目录

[**介绍** 2](#_Toc501098051)

[**所需技术** 2](#_Toc501098052)

[**方法** 2](#_Toc501098053)

[**设计** 3](#_Toc501098054)

[**Methodology** 3](#_Toc501098055)

[**Creating the dataset** 3](#_Toc501098056)

[**Data preprocessing** 5](#_Toc501098057)

[**Constructing the neural network** 5](#_Toc501098058)

[**Cost function** 5](#_Toc501098059)

[**Optimization function** 6](#_Toc501098060)

[**Training** 6](#_Toc501098061)

[**Training epochs** 6](#_Toc501098062)

[**Number of neurons in the hidden layers** 6](#_Toc501098063)

[**Learning rate** 6](#_Toc501098064)

[**Dropouts** 6](#_Toc501098065)

[**Results and discussion** 7](#_Toc501098066)

[**References** 10](#_Toc501098067)

**如何创建一个分类器来确定歌曲的潜在流行度**

帮助音乐家成功的工具

# **介绍**

According to my perspective I saw many musicians, may it be a band or an individual, do not achieve their desired status in the society in terms of popularity while few dominate the music industry. Therefore, my efforts were to develop a system where each music enthusiast who is interested in displaying their unique talent via song(s) receives the ability to assess the potential popularity they are likely to receive from the audience.

根据我的观点，我看到很多音乐家，不管是乐队还是个人，在流行方面都没有在社会上达到他们所期望的地位，而在音乐界则很少。 因此，我的努力是开发一个系统，每个有兴趣通过歌曲展示他们独特才能的音乐爱好者都能够评估他们可能从观众那里获得的潜在受欢迎程度。

The popularity is linked to a rating which depends on music audience preference. There will be three rating classes which will be labelled as excellent, moderate and poor. Therefore a higher rating(excellent) suggest a popularity similar to other highly rated songs among music audience and vice versa. While the moderate hold tracks in between the above two.

受欢迎度与依赖于音乐受众偏好的评分相关联。 将有三个评级班，将被标为优秀，中等和差。 因此，较高的评价（优秀）表明在音乐受众中与其他高度评价的歌曲类似的受欢迎程度，反之亦然。 而温和的持有轨道在上述两者之间。

# **所需技术**

Tensorflow- 流行的机器学习库

Pandas - 一个用于python的数据处理和操作库

Librosa - 一个音乐和音频分析工具

scikit-learn- 另一个机器学习库(高级API接口)

numpy- 一个用于高效执行矩阵（2d阵列）操作的便捷工具

os-处理文件系统相关的操作

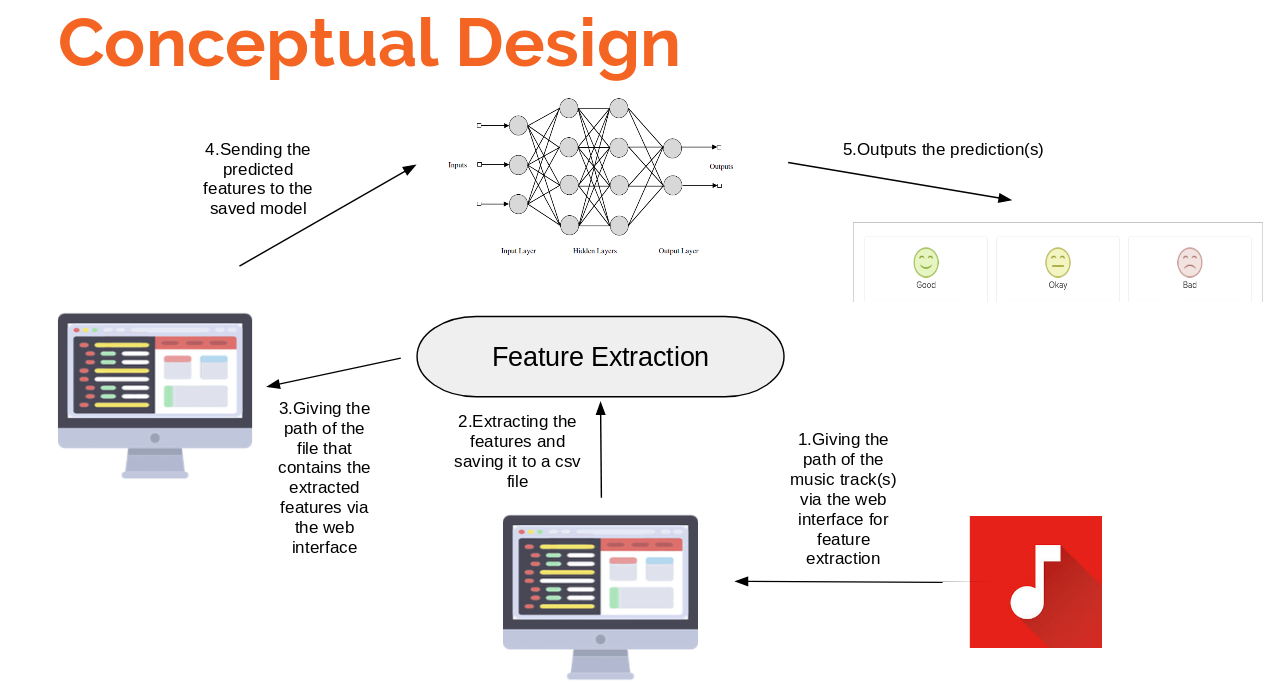
CatBoost- 一个有利于使用很少过拟合而完成提升树的实现

XGBoost - 梯度提升算法

# **方法**

因为僧伽罗语（Sinhala）是我的母语，且关于它的研究比较少，所以我选择僧伽罗歌曲作为研究对象。现在，还没有僧伽罗语歌曲相关的数据集。因而，有需要创建一个数据集。我们的任务就是采用监督学习的方法创建一个分类器。我们主要使用神经网络作为我们分类器算法。具体的说，使用由tensorflow实现的多层神经网络。我们也和其它技术获取的结果进行比较。

# **设计**



A complete process execution of the system

# **方法论**

When developing our solution we considered various approaches to choose the best one.The comparison of the performance of each approach can be found under the results section.The following implementations were considered .

1.Vanilla neural networks(multi layer neural network)

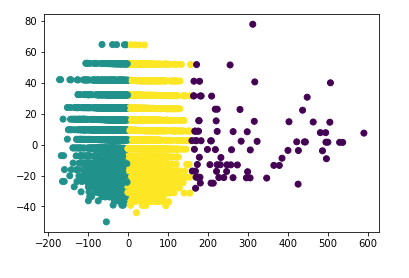
2.Ensemble technique(Random forest)

3.Boosting (XGboost,CatBoost)

4.Stacking(2 base learners ,1 meta learner)

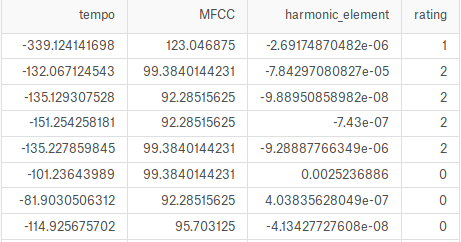
## **创建数据集**

We will extract three meaningful features for each song,from a 8000+ music repository using librosa.The three features will be tempo(beats per minute),Mel Frequency Cepstral Co-efficients(mimics some parts of the human [speech production](http://recognize-speech.com/speech/speech-production) and [speech perception](http://recognize-speech.com/speech/sense-of-hearing) ) and the harmonic element(the harmonic component within an audio signal). The reason of using these three features are because they are considered to be high level features of music and high level features have proven to be more determinant factors of preference by audience since they capture the characteristics the audience value most.Next we need to label this dataset.For this we use K-means clustering to cluster the data points in to three clusters which equal to the number of rating classes.Here we assume the songs with similar characteristics generates produce feature values close to each other, therefore when calculating the distance measure to determine the cluster that the data points belong to, the distances for data points that have similar rating will have minor differences.Thus falling to the same cluster. After the labels were determined the the features and labels were merged to create the dataset.

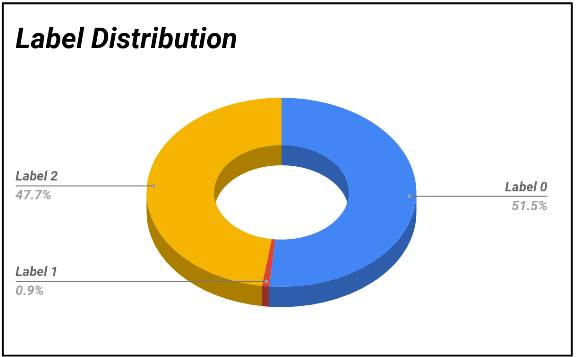


Clustering performed on the dataset using three clusters

The cluster labels will be assigned random whole numbers between 0 and number of clusters -1 to the three clusters.These labels 0,1 and 2 are just notations that separate the three clusters and therefore have no numeric representation.Therefore if someone needs to identify which label is excellent ,moderate and poor to assess where they stand .They will have to define success in terms of their perspective.The reason for this is the subjective nature of music from person to person.so for instance if I want to assess my song relative to what I see as a popular song.I have to first choose three song which I perceive as excellent,moderate and poor and extract the features of those songs and give them to the system and obtain the rating/label for those songs.Now since I know what the labels mean from my perspective I can give my creation to the system and obtain the label/rating for it and compare where I stand.



First few records of the dataset



Label Distribution

## **Data preprocessing**

The LabelBinerizer() in sklearn has been used to create the equivalent one-hot encoding on labels.

StandardScaler() is used to standardize data to a common Gaussian distribution with a mean of zero and a standard deviation of one.

## **Constructing the neural network**

The neural network has an input layer ,two hidden layers and an output layer.First we create three place holders to feed the features, labels and the probability of each neuron being present.which is required in the dropout layer as a parameter.The values for these place holders will be provided during runtime.Then we create each layer of the network by declaring weights and biases for each layer.dropouts will be added on the output from the activation function after each layer except after the output layer.The basic concept of feed forward neural nets is that the inputs that are fed to a layer are multiplied by a weight matrix and added to biases in that layer.These weights and biases are the variables that can be changed in order to make the the learning function generalize.The result from a layer is given to an activation function which maps the inputs to outputs in a certain range depending on the activation function.The activation function for input layer and hidden layers would be *tanh* while the output layer will have the*softmax*activation function.The best activation for intermediary layers were proven to be*tanh* for which the comparison of popular activation functions against performance will be available under the results section.

## **Cost function**

The cost function which we used was the cross entropy function.which takes the log values of the predicted labels and multiply it by the actual labels. then it takes the summation and creates the new matrix values. To obtain the cost for each batch we compute the mean along the rows of the matrix.Now we have a column matrix which specifies the cost from each batch or one epoch.

## **Optimization function**

As the optimization function we use the stochastic gradient descent which adjusts the learning curve by the learning rate in the direction of cost reduction.

## **Training**

Training is done in batches to reduce over-fitting.Further the number of training epochs are set to 200.This should be done within a session in Tensorflow as the computational graph in Tensorflow is evaluated only within a session.Values to be fed to placeholders are fed during training using the feed\_dict dictionary parameter.The run method in the session class can be used to run the operations of the computational graph.

Setting Hyper-parameters

### **Training epochs**

We say that an epoch is completed when we have used all our training data for the training process. Training data consist of our training features and it’s corresponding training labels.Here we have set training epochs to 200 which mean we train on our entire training data on 200 iterations. There is no ideal number of training epochs we could use.This depends on the complexity of your data.Therefore you should do parameter tuning or basically try few parameter configurations to find the ideal/suitable value for this parameters.

Hyper parameter 1:training\_epochs = 200

Since we are implementing a multi-layer neural network.It will consist of an input layer, two hidden layers and an output layer.

### **Number of neurons in the hidden layers**

Hidden layers are the layers which perform transformations on the input data to identify patterns and generalize our model.Here I have used 120 neurons each in my first and second hidden layers which was sufficient in achieving a decent accuracy. But as I explained earlier all hyper-parameters should be tuned in such a way that it improves your model.

Hyper parameter 2:n\_neurons\_in\_h1 = 120

Hyper parameter 3:n\_neurons\_in\_h2 = 120

### **Learning rate**

This is the phase at which the algorithm learns.Machine Learning guru’s say that we should start with a high learning rate and gradually reduce it to achieve best results. Further the learning rate is advised to be kept within the range of 0 & 1.

Hyper parameter 4:learning\_rate = 0.001

### **Dropouts**

Used to reduce over-fitting during training.

keep\_prob=0.5 for training and 1.0 for testing. Dropouts are only used during training and not testing.The above probability variable specifies the probability of each neuron to remain in a layer.

Finally the model can be saved after training using the save() method in the Saver() class.

Measures taken to reduce over-fitting

1.Shuffling the dataset

2.Standardizing the dataset

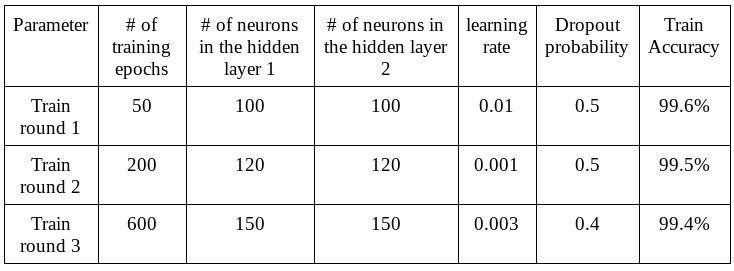
3.Adding dropout layers

4.Training dataset in batches of samples.

# **Results and discussion**

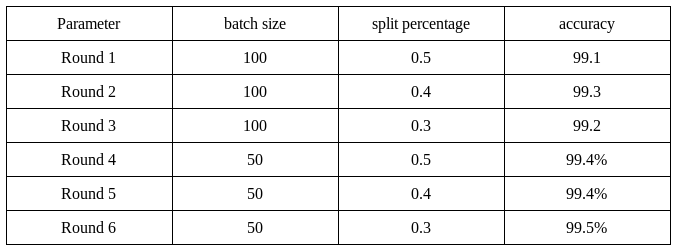
In this section we will be assessing the performance of each approach we used in solving the problem and the inferences we could gain.

Parameter tuning results are as follows:

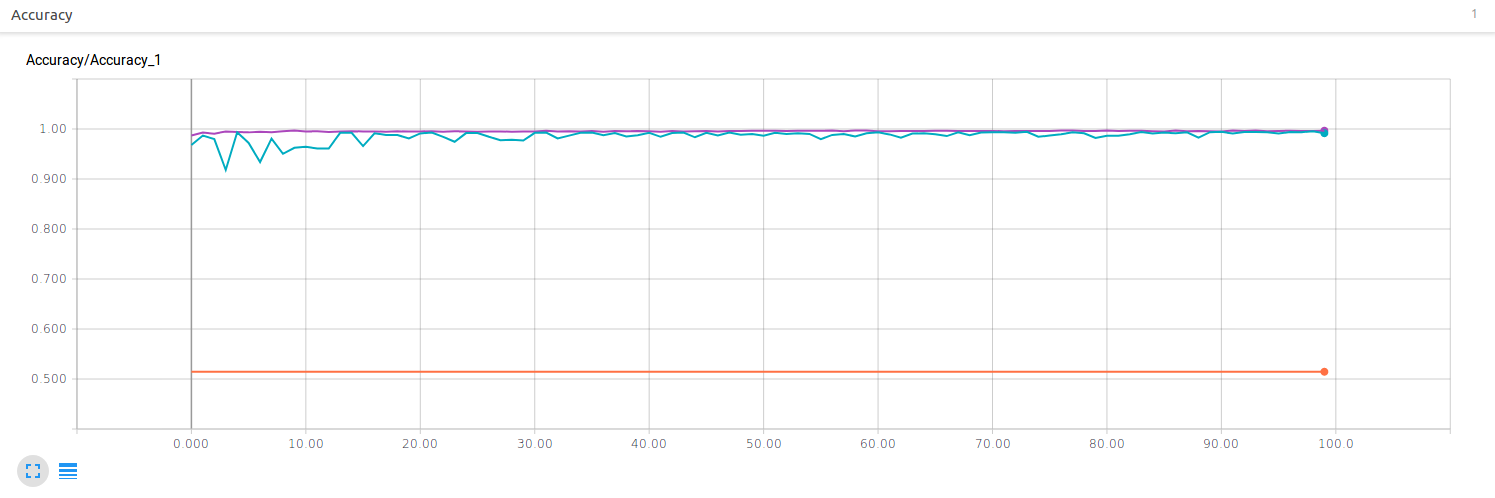


Hyper parameter configurations with changes in accuracy

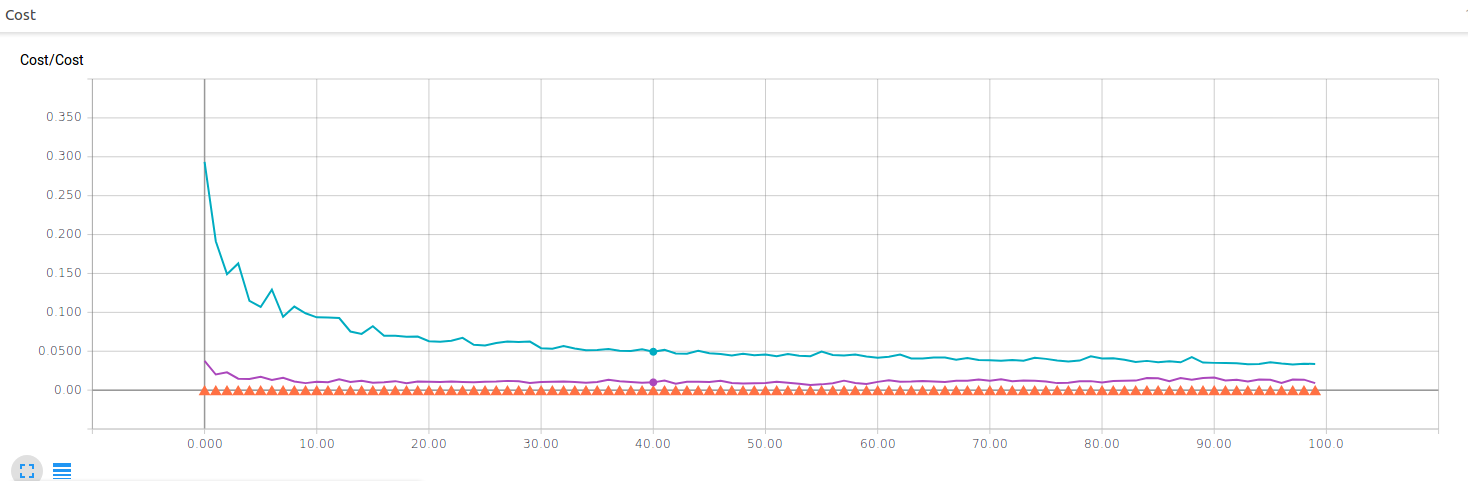
Other parameter(batch size & split percentage) changes:



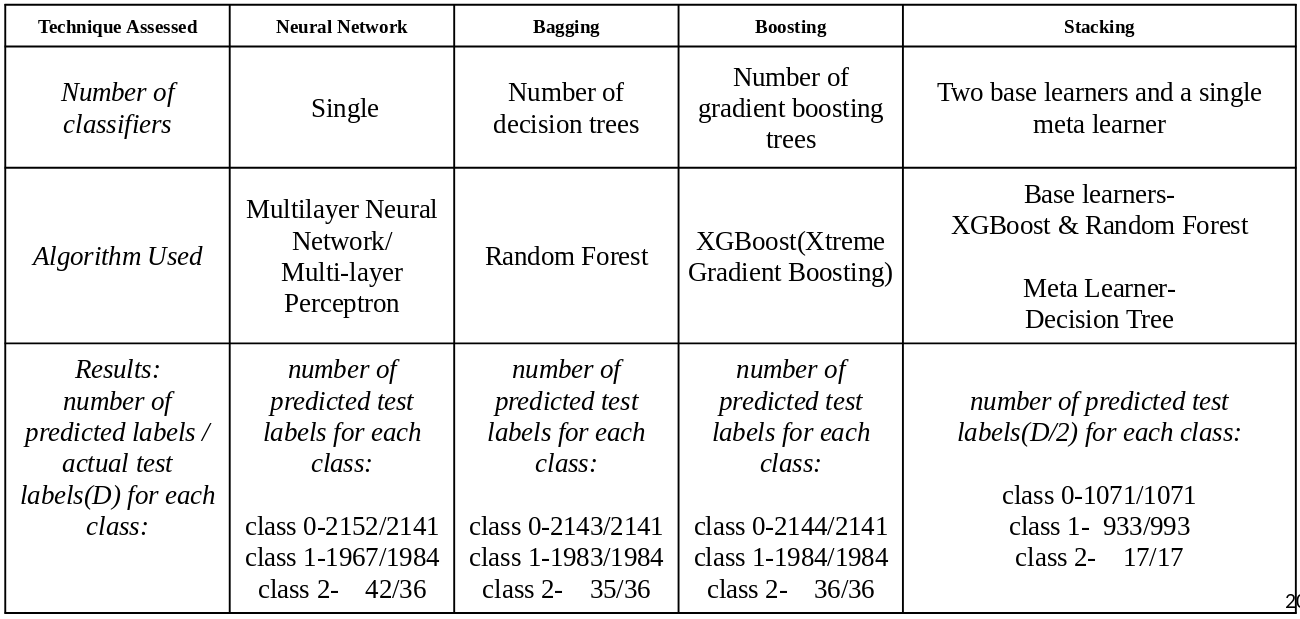
Changes in accuracy with other parameter configurations



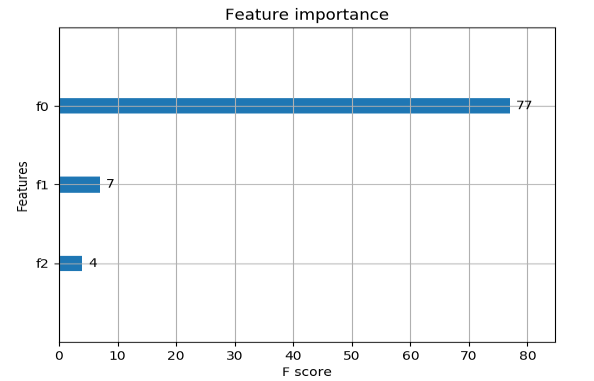
Accuracy variation against training epochs



Cost variation against training epochs

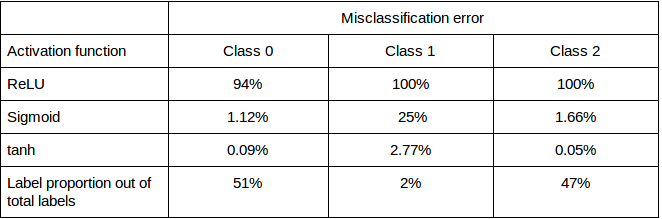


Performance comparison of each approach



Feature importance plot

The above diagram shows the predictive power of each feature as measured by the F-score(2TP/2TP+FP+FN) .Where TP is the number of true positives,FP is the number of false positives and FN is the number of false negatives.



Variation in misclassification error with various activation functions

Its clear that in terms of prediction accuracy stacking provides impressive results by classifying every element in each class correctly.While Boosting and bagging are also perform well.The neural network implementation is not very impressive but gives acceptable results.The most important fact is given the bias nature of the dataset still the neural network has managed to identify the small proportion of class 2.But this was after many techniques mentioned above were incorporated to minimize over-fitting. Further*tanh* is considered to be the best choice in terms of choosing an activation function for layers other than the output layer.

# **References**

1. TensorFlow

An open-source software library for Machine Intelligence

<https://www.tensorflow.org/>

2. LibROSA – librosa 0.5.1 documentation

libROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music…

<https://librosa.github.io/librosa/>

3. scikit-learn: machine learning in Python – sci-learn 0.19.1 documentation

Edit description

<http://scikit-learn.org/stable/>

4. catboost/catboost

Catboost – CatBoost is an open-source gradient boosting on decision trees library with categorical features support out…

<https://github.com/catboost/catboost>

5. XGBoost Documentation – xgboost 0.4 documentation

This is document of xgboost library. XGBoost is short for eXtreme gradient boosting. This is a library that is designed…

<http://xgboost-clone.readthedocs.io/en/latest/>

6. Urban Sound Classification, Part 1

Feature extraction from sound and classification using Neural Networks.

<https://aqibsaeed.github.io/2016-09-03-urban-sound-classification-part-1/>

7. The Speech Recognition Wiki

Introduction The most commonly used feature extraction method in automatic speech recognition (ASR) is Mel-Frequency…

<http://recognize-speech.com/feature-extraction/mfcc>

附录：

import numpy as np

import librosa

import os

import pandas as pd

def getSourcePath(path):

#feature extraction method

def extract\_feature(file\_name):

X, sample\_rate = librosa.load(file\_name)# Audio buffers are called X, Sampling rate is called sr

onset\_env = librosa.onset.onset\_strength(X, sr=sample\_rate)#sr deafult 22050Hz

#beats per minute(tempo)

tempo = np.mean(librosa.beat.tempo(onset\_envelope=onset\_env, sr=sample\_rate).T,axis=0)#axis=0 along columns

#captures the specific characteristics of sinhala music

mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample\_rate, n\_mfcc=1).T,axis=0)

#extracts the harmonic element

y\_harmonic = np.mean(librosa.effects.harmonic(X, margin=3.0).T,axis=0)#Use a margin > 1.0 for greater harmonic separation

return tempo,mfccs,y\_harmonic

#parsing the file system to extract features & labels

def parse\_audio\_files(dirname):

labels = []

labels = np.array([labels])

ext\_features = []

ext\_features = np.array([ext\_features])

for dirs, subdir, files in os.walk(dirname):

for file in files:

tempo, mfccs, y\_harmonic = extract\_feature(os.path.join(dirs, file))

features = np.hstack((mfccs, tempo, y\_harmonic))

ext\_features = np.append(ext\_features, np.array(features))

labels = np.append(labels, np.array(file.split("@")[0]))

return np.array(ext\_features), np.array(labels, dtype=str)

dirname=path

#recording the features total data

total\_features,labels= parse\_audio\_files(dirname)

total\_features=total\_features.reshape(total\_features.shape[0]/3,3)

x=pd.concat([pd.DataFrame(labels),pd.DataFrame(total\_features)],axis=1)

x=pd.DataFrame(x)

x.to\_csv('FeatureHolder.csv',index=False,header=None)

return "Return Success"

[FeatureExtractor.py](https://gist.githubusercontent.com/TharindraParanagama/bbec97123cf383ea381abe02da0eae0a/raw/04113cef73b7963d5918a146ee707f2d88214944/FeatureExtractor.py) hosted with by GitHub

特征提取的源代码

#declaring inputs

import tensorflow as tf

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer, OneHotEncoder, StandardScaler

#loading data from text file

dataset=np.loadtxt('/PathToDataset',delimiter=',',skiprows=1)

#specifying feature and label columns

features=dataset[:,0:3]

labels=dataset[:,3]

#binarizing labels/one-hot encoding

le=LabelBinarizer()

labels=le.fit\_transform(labels)

#standardizing data

scale=StandardScaler()

norm\_features=scale.fit\_transform(features)

#splitting data to train and test split

tr\_features,ts\_features,tr\_labels,ts\_labels=train\_test\_split(norm\_features,labels,test\_size=0.5,random\_state=42)

#printing results for testing purposes

print ("tr\_features", tr\_features)

print ("tr\_encode", tr\_labels)

print ("ts\_features", ts\_features)

print ("ts\_encode", ts\_labels)

# setting hyper parameters & other variables

training\_epochs = 200

n\_features = tr\_features.shape[1]

n\_classes = tr\_labels.shape[1]

n\_neurons\_in\_h1 = 120

n\_neurons\_in\_h2 = 120

learning\_rate = 0.001

#printing results for testing purposes

print (n\_classes)

# placeholdr tensors built to store features(in X) , labels(in Y) and dropout probability(in keep\_prob)

X = tf.placeholder(tf.float32, [None, n\_features], name='features')

Y = tf.placeholder(tf.float32, [None, n\_classes], name='labels')

keep\_prob=tf.placeholder(tf.float32,name='drop\_prob')

#The point for using truncated normal is to overcome saturation of functions like sigmoid (where if the value is too big/small, the neuron stops learning).

#Weights allow you to change the steepness(value ranges) of the activation function in such a way that you will yield better results.while the biases allow you to shift your activation function left or right.

# network parameters(weights and biases) are set and initialized(Layer1)

W1 = tf.Variable(tf.truncated\_normal([n\_features, n\_neurons\_in\_h1], mean=0, stddev=1 / np.sqrt(n\_features)), name='weights1')

b1 = tf.Variable(tf.truncated\_normal([n\_neurons\_in\_h1],mean=0, stddev=1 / np.sqrt(n\_features)), name='biases1')

# activation function(tanh)

y1 = tf.nn.tanh((tf.matmul(X, W1)+b1), name='activationLayer1')

#dropout layer 1

drop\_out\_layer1 = tf.nn.dropout(y1, keep\_prob)

# network parameters(weights and biases) are set and initialized(Layer2)

W2 = tf.Variable(tf.truncated\_normal([n\_neurons\_in\_h1, n\_neurons\_in\_h2],mean=0, stddev=1 / np.sqrt(n\_features)), name='weights2')

b2 = tf.Variable(tf.truncated\_normal([n\_neurons\_in\_h2],mean=0, stddev=1 / np.sqrt(n\_features)), name='biases2')

# activation function(tanh)

y2 = tf.nn.tanh((tf.matmul(drop\_out\_layer1, W2)+b2), name='activationLayer2')

#dropout layer 2

drop\_out\_layer2 = tf.nn.dropout(y2, keep\_prob)

# network parameters(weights and biases) are set and initialized(output layer)

Wo = tf.Variable(tf.truncated\_normal([n\_neurons\_in\_h2, n\_classes],mean=0, stddev=1 / np.sqrt(n\_features)), name='weightsOut')

bo = tf.Variable(tf.truncated\_normal([n\_classes],mean=0, stddev=1 / np.sqrt(n\_features)), name='biasesOut')

# activation function(softmax)

a = tf.nn.softmax((tf.matmul(drop\_out\_layer2, Wo) + bo), name='activationOutputLayer')

# tensorboard histograms on summary operations

tf.summary.histogram("weights1", W1)

tf.summary.histogram("biases1", b1)

tf.summary.histogram("weights2", W2)

tf.summary.histogram("biases2", b2)

tf.summary.histogram("weightsOut", Wo)

tf.summary.histogram("biasesOut", bo)

# name scope for the cost function for more clarity on tensorboard

with tf.name\_scope('Cost'):

# cost function(cross entropy)

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(Y \* tf.log(a),reduction\_indices=[1]))#reduction indices=1 means row wise mean

#optimization function

train\_step = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cross\_entropy)

# scalar summary for plotting cost variation againt epoches

tf.summary.scalar('Cost', cross\_entropy)

# name scope for the accuracy for more clarity on tensorboard

with tf.name\_scope('Accuracy'):

# compare predicted value from network with the expected value/target

correct\_prediction = tf.equal(tf.argmax(a, 1), tf.argmax(Y, 1))

# accuracy determination

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32), name="Accuracy")

# scalar summary for plotting accuracy variation against epoches

tf.summary.scalar('Accuracy', accuracy)

# initialization of all variables

initial = tf.global\_variables\_initializer()

#creating an instance of a session object to execute the computational graph

with tf.Session() as sess:

sess.run(initial)

#writing summery values to file

writer = tf.summary.FileWriter("/writingSummaryOps")

#saving the computational graph

writer.add\_graph(sess.graph)

#merging all summary operations

merged\_summary = tf.summary.merge\_all()

# training in batches of samples

batchsize=50

for epoch in range(training\_epochs):

for i in range(len(tr\_features)):

start=i

end=i+batchsize

x\_batch=tr\_features[start:end]

y\_batch=tr\_labels[start:end]

# feeding training data/examples

sess.run(train\_step, feed\_dict={X:x\_batch , Y:y\_batch,keep\_prob:0.5})

i+=batchsize

# feeding testing data to determine model accuracy

y\_pred = sess.run(tf.argmax(a, 1), feed\_dict={X: ts\_features,keep\_prob:1.0})

y\_true = sess.run(tf.argmax(ts\_labels, 1))

#accuracy for each epoch

summary, acc = sess.run([merged\_summary, accuracy], feed\_dict={X: ts\_features, Y: ts\_labels,keep\_prob:1.0})

# write results to summary file

writer.add\_summary(summary, epoch)

# print accuracy for each epoch

print('epoch',epoch, acc)

print ('---------------')

print(y\_pred, y\_true)

#saving model

saver = tf.train.Saver()

saver.save(sess,"/model/saveGeetha.ckpt")

print("Model saved")

[**CreatingAMulti-LayerNeuralNetwork.py**](https://gist.github.com/TharindraParanagama/b640b1af58c85b7a36b5c573243a1245#file-creatingamulti-layerneuralnetwork-py) hosted with by [**GitHub**](https://github.com/)

Source code fro neural network implementation,training and evaluation

关键词：

Machine Learning， Classification， Music， TensorFlow，Ensembling