# TNSDC-Generative AI

# Predicting the Music Mood of a Song with Deep Learning.

# A cool way to predict the mood of music tracks with Neural Networks models using Keras and Tensorflow Libraries on Python.

## Done by – Jayapriya.T

## IIIrd year CSE A

## Reg no- 211521104058

Music is a powerful language to express our feelings and in many cases is used as a therapy to deal with tough moments in our lives. Emotions and moods can be easily reflected in music, when we are doing sports, we tend to listen to energetic music, similarly when we are anxious or tired a nice relaxed song can help us to calm down. That’s why I try to figure out how classification models could help to determinate which is the mood of a specific track.

In this article, I will expose and explain how I could achieve this idea using a **Multi-Class Neural Network for Classification** and a cool Dataset provided from Spotify. So with no more talk, let’s start working!!.

# Required Tools:

* **Pandas** and **Numpy** for data analysis.
* **Keras and Tensorflow**to build the Deep Learning model.
* **Sklearn** to validate the model.
* **Seaborn**and **Matplotlib**to plot a nice graph.
* **Spotipy** Python Library ([click here](https://spotipy.readthedocs.io/en/2.13.0/) for more info).
* **Spotify Credentials** to access their Apis and Data acquisition ([click here](https://developer.spotify.com/)for more info).

# Spotify Audio Features:

Spotify uses a series of different features to classify tracks. I copy/paste the information from the Spotify Webpage.

* Acousticness: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
* Danceability: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
* Energy: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
* Instrumentalness: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
* Liveness: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides a strong likelihood that the track is live.
* Loudness: the overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
* Speechiness: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
* Valence: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
* Tempo: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, the tempo is the speed or pace of a given piece and derives directly from the average beat duration.

# 1. Explaining and Analysing the Data:

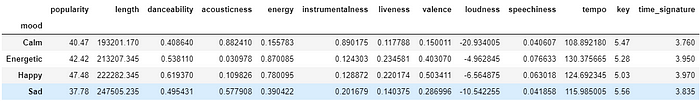
To obtain the data I had to create a series of functions using the Spotipy Library. This library helps to automate the Spotify services downloading more technical information (explaining above) about playlists, songs, artist music, etc. For the main purpose of this article, I’ll not mention how I obtained the data, but I’ll explain what the data consists of.

As you may know, Classification problems use labeled data, so I had to create these labels. I decided to create 4 categories to label the tracks, these categories are “**Energetic**”, ”**Calm**”, “**Happy**” and “**Sad**”. I choose these categories based on the following article, who explains what is the best way of classifying music by mood.

Then I searched on Spotify some playlists with different music tracks based on these 4 labels (200 tracks per label) and finally, I concatenated all these tracks into the main data frame labeled by each mood.

The main data have **800 rows** and **18 columns**, but for information reduction purposes I decided to use the features of **Length**,**Danceability**, **Acousticness**, **Energy**, I**nstrumentalness**, **Liveness**, **Valence**, **Loudness**, **Speechiness** and **Tempo** because they have more influence to classify the tracks.

I grouped the data frame by labels calculating the mean of the tracks’ features. I obtained the following result:



Doing this simple analysis I quickly noticed that the most popular songs are Happy, Sad songs tend to have a long length, Energetic songs are most fast in tempo, and Calm songs tends to be acousticness.

# 2. Building the Model:

## 2.1- Pre-Processing the Data:

To normalize the features I used **MinMaxScaler**to scale the values between a range of [0,1] and preserving the shape of the original distribution. I also encoded the 4 labels because Neural Networks uses numerical values to train and test. Finally, I split the data by **80% for training** and **20% for testing**.

#Libraries to pre-process the variables  
from sklearn.preprocessing import LabelEncoder,MinMaxScaler  
from sklearn.model\_selection import train\_test\_split

The code I used to process the data:

#Define the features and the target  
col\_features = df.columns[6:-3]  
X = df[col\_features]  
Y = df['mood']#Normalize the features  
X= MinMaxScaler().fit\_transform(X)#Encode the labels (targets)  
encoder = LabelEncoder()  
encoder.fit(Y)  
encoded\_y = encoder.transform(Y)#Split train and test data with a test size of 20%  
X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,encoded\_y,test\_size=0.2,random\_state=15)

**2.2 Creating the model:**

To build the model I used the library Keras, this library is designed to enable fast experimentation with **Deep Neural Networks**, focused on being user-friendly. My main goal is to classify tracks in the 4 categories of moods (**Calm, Energetic, Happy and Sad**) so my model consists of a **Multi-Class Neural Network**with an input of **10 Features**, **1 Layer**with **8 nodes**, and **4 outputs**with the output Layer. I also need to use a Classifier as an Estimator, in this case, the Classifier is **KerasClassifier**, which takes as an argument a function that I created previously with the Neural Network model defined. The activation Function corresponds to a **Rectified Linear Unit**(**Relu)**, the Loss function is a L**ogistic Function** and **Adam Gradient Descent Algorithm** is the optimizer.

*#Libraries to create the Multi-class Neural Network*  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.wrappers.scikit\_learn import KerasClassifier  
from keras.utils import np\_utils*#Import tensorflow and disable the v2 behavior and eager mode*  
import tensorflow as tf  
tf.compat.v1.disable\_eager\_execution()  
tf.compat.v1.disable\_v2\_behavior()

* **Important**: I disabled the eager execution and v2 behavior of TensorFlow because I keep trying to understand and to learn how the library works in those modes (Sorry I’m a newbie in Tensorflow hehe).

The code I used to build the Neural Network:

*#Function that creates the structure of the Neural Network*  
def base\_model():  
 *#Create the model*  
 model = Sequential()*#Add 1 layer with 8 nodes,input of 4 dim with relu function*  
 model.add(Dense(8,input\_dim=10,activation='relu'))*#Add 1 layer with output 3 and softmax function*  
 model.add(Dense(4,activation='softmax'))*#Compile the model using logistic loss function and adam*  optimizer, accuracy correspond to the metric displayed  
 model.compile(loss='categorical\_crossentropy',optimizer='adam',  
 metrics=['accuracy'])  
 return model*#Configure the estimator with 300 epochs and 200 batchs. the build\_fn takes the function defined above*.  
estimator = KerasClassifier(build\_fn=base\_model,epochs=300,  
 batch\_size=200)

**3.3 Evaluating the model:**

Using **K-Fold Cross Validation** I evaluated the estimator using the train data. The number of splits is K=**10** shuffling all the values.

*#Library to evaluate the model*  
from sklearn.model\_selection import cross\_val\_score, KFold

The code I used to evaluate the model:

*#Evaluate the model using KFold cross validation*kfold = KFold(n\_splits=10,shuffle=True)results = cross\_val\_score(estimator,X,encoded\_y,cv=kfold)print("%.2f%% (%.2f%%)" % (results.mean()\*100,results.std()\*100))

The Accuracy of the model is the average of the accuracy of each fold, in this case, the Accuracy was **72.75**%.

**3.4 Training the Model:**

It’s time to train the model! So let’s go code:

*#Train the model with the train data*  
estimator.fit(X\_train,Y\_train)*#Predict the model with the test data*  
y\_preds = estimator.predict(X\_test)

It’s important to mention that the model was trained with **640 samples** (80% of the main data).

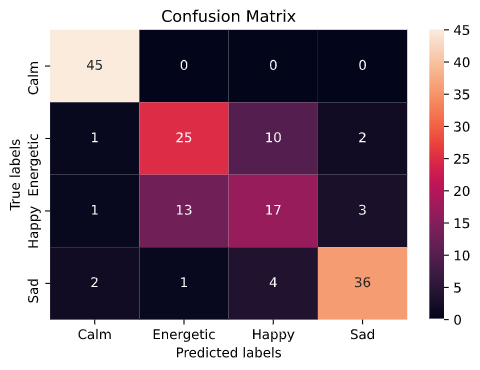
Some of the output of the last epochs when the model was training.



# 3. Accuracy of the Multi-Class Neural Network:

Finally to evaluate the accuracy of the model I plotted a Confusion Matrix using Seaborn Library and Matplotlib. I also calculated the accuracy score provided by Sklearn Library.

#Create the confusion matrix using test data and predictions  
cm = confusion\_matrix(Y\_test,y\_preds)#plot the confusion matrix  
ax = plt.subplot()  
sns.heatmap(cm,annot=True,ax=ax)  
labels = target['mood'].tolist()  
ax.set\_xlabel('Predicted labels')  
ax.set\_ylabel('True labels')  
ax.set\_title('Confusion Matrix')  
ax.xaxis.set\_ticklabels(labels)  
ax.yaxis.set\_ticklabels(labels)  
plt.show()#Show the accuracy score   
print("Accuracy Score",accuracy\_score(Y\_test,y\_preds))

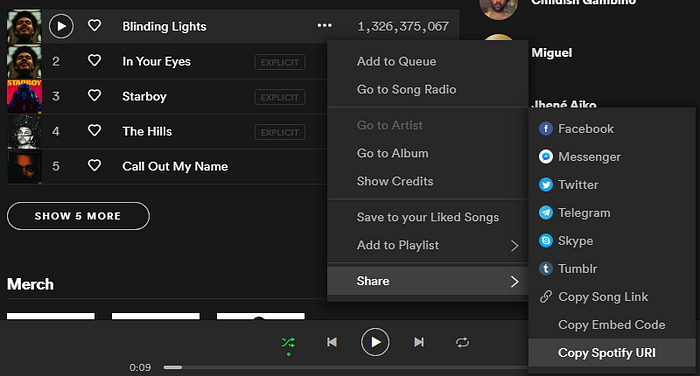


With a Final Accuracy score of **76%** and taking a look at the Confusion Matrix, I noticed my model is good classifying Calm and Sad songs, but it’s having some issues dealing with Energetic and Happy songs. I could modify some parameters like the batch size, epochs, or maybe aggregate or delete some track features to train my model and thus help to improve the accuracy of the model.

# 4. Having Fun Classifying Music by Mood:

I want to show how to predict the mood of a song that may you are lazy to listen completely but you want to know if the song will make you dance or cry.First, we need to obtain the Spotify URI of any song provided by the Spotify App. For instance, I will like to predict the mood of “**Blinding Lights** by T**he Weekend**”.

Spotify URI of Blinding Lights : spotify:track:0VjIjW4GlUZAMYd2vXMi3b

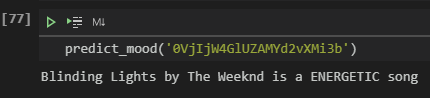


Then I will pass the Spotify URI of the song into a function I defined called predict\_mood. This function takes the Id of the song as an argument and includes inside the Neural Network model created

The code of the predict\_mood function is:

*#Import the Script helpers.py*  
from helpers import \*def predict\_mood(id\_song):  
 *#Join the model and the MinMaxScaler in a Pipeline*  
 pip = Pipeline([('minmaxscaler',MinMaxScaler()),('keras',  
 KerasClassifier(build\_fn=base\_model,epochs=300,  
 batch\_size=200,verbose=0))])*#Fit the Pipeline*  
 pip.fit(X2,encoded\_y)  
   
 *#Obtain the features of the song (Function created on helpers.py)*  
 preds = get\_songs\_features(id\_song)*#Pre-processing the input features for the Model*  
 preds\_features = np.array(preds[0][6:-2]).reshape(-1,1).T  
   
 *#Predict the features of the song*  
 results = pip.predict(preds\_features)  
 mood = np.array(target['mood'][target['encode']==int(results)])  
   
 #Obtain the name of the song and the artist  
 name\_song = preds[0][0]  
 artist = preds[0][2]*#Store the name,artist and mood of the song to print.* result\_pred=print("{0} by {1} is a {2} song".format(name\_song,  
 artist,mood))return result\_pred

Spotify URI of Blinding Lights is “spotify:track:0VjIjW4GlUZAMYd2vXMi3b” but thepredict\_mood function takes the Id,so I just need to take the code after “spotify:track”: in this case, the Id is **0VjIjW4GlUZAMYd2vXMi3b.**



And the result display that Blinding Lights by the Weeknd have an Energetic Mood. Pretty accurate don’t you think so?.

# 5. Conclusion

Deep Learning Algorithms are a lot of fun to implement ideas or projects related to things you like. In my case, I like music a lot, so I could use this knowledge to create cool ways helping me to automate a task that can take a long time to perform it. I also could learn more about this amazing world of Data Science and my tendencies to music tastes.

**References:**

* <https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85>
* <https://towardsdatascience.com/clustering-music-to-create-your-personal-playlists-on-spotify-using-python-and-k-means-a39c4158589a>
* https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/