MATEMÁTICAS Y CIENCIAS DE DATOS

IDENTIFYING THE GENRE OF A SONG WITH NEURAL NETWORKS

PARA LA TOMA DE DECISIONES

María Cecilia Chávez del Castillo A00573538

Elizabeth Guadalupe Martínez Arellano A00573510

Maria Fernanda Paz Aguilar A00573593

Diana Sofia Ponce López A00573537

Paola Valle Coronado A00573711

BITÁCORA SEMANA 1

DÍA 1 SE LEE EL PROYECTO PARA SABER QUÉ ES LO QUE SE VA A ESTAR DESARROLLANDO Y FAMILIARIZARNOS CON EL MISMO.

DÌA 2 CONCLUIMOS EN EQUIPO SOBRE LA IMPORTANCIA DEL PROGRAMA Y EL IMPACTO QUE TIENE PARA AQUELLAS EMPRESAS O PERSONAS DEDICADAS AL RUBRO MUSICAL.

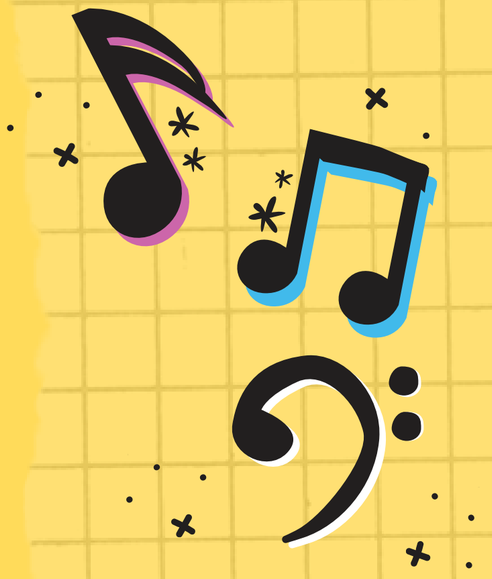
```
In [1]: import librosa
        import librosa.feature
        import librosa.display
        import glob
        import numpy as np
        import matplotlib.pyplot as plt
        from keras.models import Sequential
        from keras.layers import Dense, Activation
        from keras.utils.np_utils import to_categorical

        Using TensorFlow backend.
```

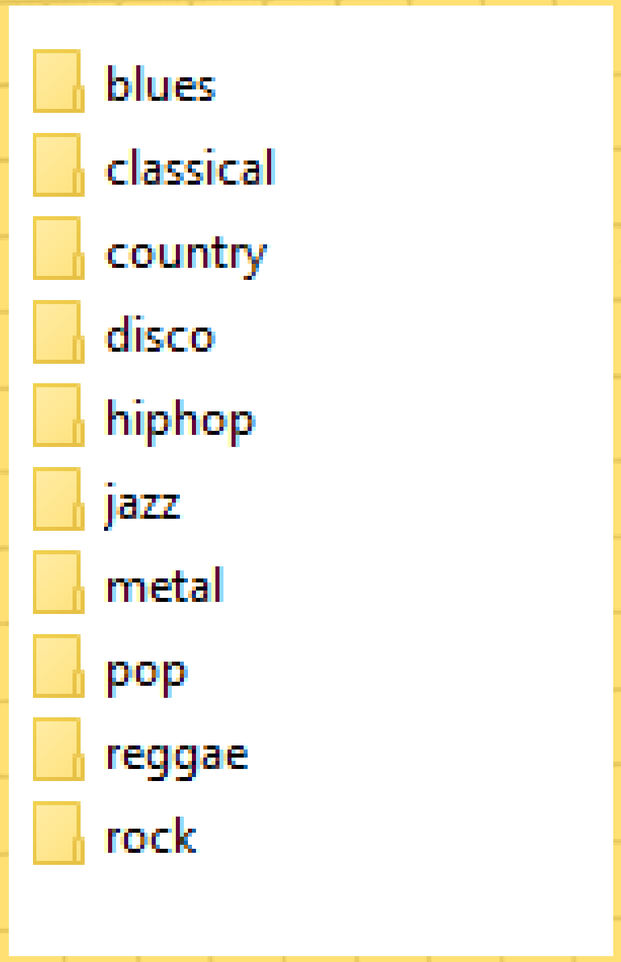
BITÁCORA SEMANA 2

DÍA 3	SE INVESTIGO LA BASE DE DATOS DE LAS FRECUENCIAS MUSICALES NOS DEDICAMOS A BUSCAR LA BASE DE DATOS CON LA QUE SE ESTARÁ TRABAJANDO
DÌA 4	SE INSTALARÁN LAS LIBRERÍAS QUE SE NECESITEN Y SE CORRERÁN PRUEBAS PARA ENTENDER SU FUNCIONAMIENTO.

BITÁCORA SEMANA 3

DÍA 5	NOS ENCARGAMOS DE COMENZAR EL DESARROLLO DEL ALGORITMO Y A OBTENER ASESORÍA PARA GARANTIZAR QUE SEA CORRECTO.
DÌA 6	CONTINUAMOS EL DESARROLLO DEL ALGORITMO PARA PASAR A LA FASE DE PRUEBAS, CORREGIR Y COMPROBAR LA FUNCIONALIDAD DEL CÓDIGO.

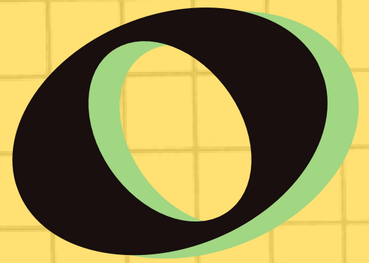




BASE DE DATOS

- *TODAS LAS CANCIONES QUE NECESITAREMOS PARA ANALIZAR.*

```
import librosa
import librosa.feature
import librosa.display
import glob
import numpy as np
import matplotlib.pyplot as plt
```



LIBRERIAS

- LIBROSA
- GLOB
- NUMPY
- MATPLOTLIB
- KERAS
- *TENSORFLOW*

```
In [6]: from keras.models import Sequential
Traceback (most recent call last):

  File "C:\Users\MaCec\AppData\Local\Temp/ipykernel_14624/412018735.py", line 1, in <module>
    from keras.models import Sequential

  File "C:\Users\MaCec\anaconda3\lib\site-packages\keras\__init__.py", line 21, in <module>
    from tensorflow.python import tf2

ModuleNotFoundError: No module named 'tensorflow'
```

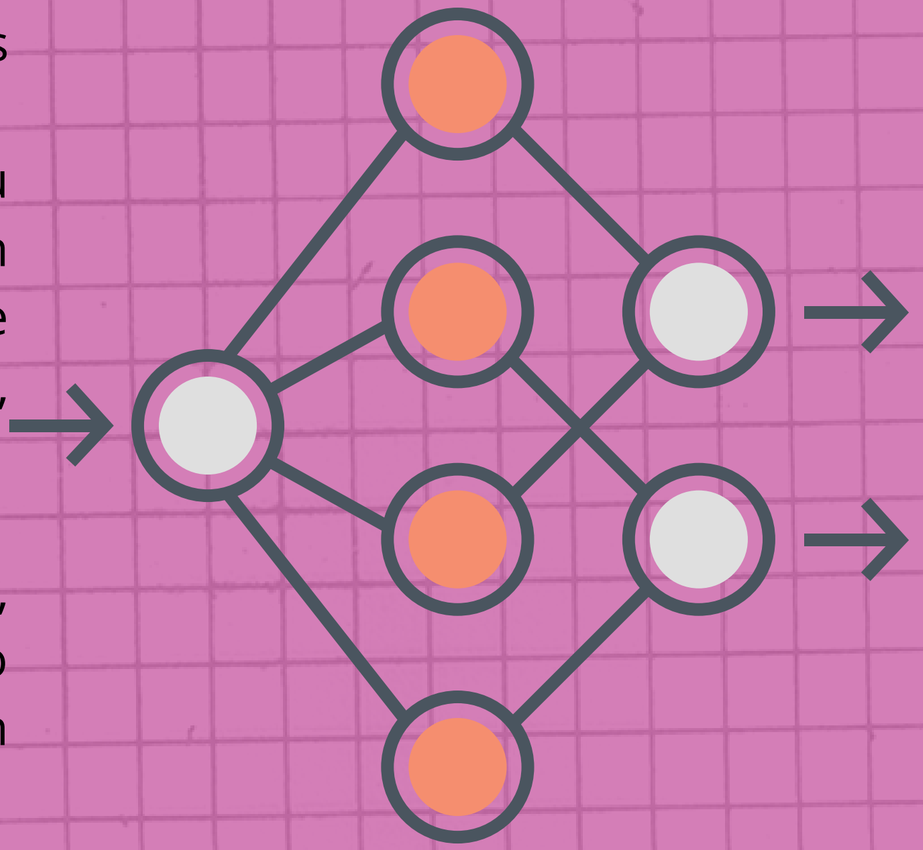
```
C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Users\MaCec> conda install-forge tensorflow

CommandNotFoundError: No command 'conda install-forge'.
Did you mean 'conda install'?

(base) PS C:\Users\MaCec> conda install -c conda-forge tensorflow
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: \
```

CHECAR EL ALMACENAMIENTO, SI NO HAY SUFICIENTE LA INSTALACIÓN SE CICLA.

REDES NEURONALES

Las redes neuronales son neuronas conectadas entre ellas de manera secuencial. A cada una de esas secuencias las llamamos capas.
Las redes neuronales están compuestas de neuronas, que a su vez se agrupan en capas: cada neurona está conectada con todas las neuronas de la capa anterior. En cada neurona, se realizarán una serie de operaciones las cuales, al optimizar, conseguiremos que nuestra red aprenda.

La red aprende examinando los registros individuales, generando una predicción para cada registro y realizando ajustes a las ponderaciones cuando realiza una predicción incorrecta.



PRIMER PASO

```
def extract_features_song(f):
    y, _ = librosa.load(f)

    # get Mel-frequency cepstral coefficients
    mfcc = librosa.feature.mfcc(y)
    # normalize values between -1,1 (divide by max)
    mfcc /= np.amax(np.absolute(mfcc))

    return np.ndarray.flatten(mfcc)[:25000]
```

SEGUNDO PASO

SE TENDRÁ QUE REALIZAR UNA FUNCIÓN QUE EN EL CASO DEL LIBRO SE LLAMA "GENERATE_FEATURES_AND_LABELS" , QUE RECORRERÁ TODOS LOS DIFERENTES GÉNEROS Y PASARÁ POR TODAS LAS CANCIONES DEL CONJUNTO DE DATOS Y PRODUCIRÁ ESOS VALORES MFCC Y LOS NOMBRES DE LAS CLASES:

```
In [12]: def generate_features_and_labels():
             all_features = []
             all_labels = []

             genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
             for genre in genres:
                 sound_files = glob.glob('genres/'+genre+'/*.au')
                 print('Processing %d songs in %s genre...' % (len(sound_files), genre))
                 for f in sound_files:
                     features = extract_features_song(f)
                     all_features.append(features)
                     all_labels.append(genre)

             # convert labels to one-hot encoding
             label_uniq_ids, label_row_ids = np.unique(all_labels, return_inverse=True)
             label_row_ids = label_row_ids.astype(np.int32, copy=False)
             onehot_labels = to_categorical(label_row_ids, len(label_uniq_ids))
             return np.stack(all_features), onehot_labels
```

TERCER PASO

PARA TODAS LAS ETIQUETAS EN ESTE MOMENTO, HAY UNA LISTA DE 1000 ENTRADAS, Y DENTRO HAY PALABRAS COMO: BLUES, CLASSICAL, COUNTRY, DISCO, ETC. AHORA, ESTO VA A SER UN PROBLEMA PORQUE UNA RED NEURONAL NO VA A PREDECIR UNA PALABRA O INCLUSO LETRAS. NECESITAMOS DARLE UNA CODIFICACIÓN NUMERICA , LO QUE SIGNIFICA QUE CADA PALABRA AQUÍ SE REPRESENTARÁ COMO DIEZ NÚMEROS BINARIOS. EN EL CASO DE LOS BLUES, SERÁ UNO Y LUEGO NUEVE CEROS.

```
In [12]: def generate_features_and_labels():
             all_features = []
             all_labels = []

             genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
             for genre in genres:
                 sound_files = glob.glob('genres/'+genre+'/*.au')
                 print('Processing %d songs in %s genre...' % (len(sound_files), genre))
                 for f in sound_files:
                     features = extract_features_song(f)
                     all_features.append(features)
                     all_labels.append(genre)

             # convert labels to one-hot encoding
             label_uniq_ids, label_row_ids = np.unique(all_labels, return_inverse=True)
             label_row_ids = label_row_ids.astype(np.int32, copy=False)
             onehot_labels = to_categorical(label_row_ids, len(label_uniq_ids))
             return np.stack(all_features), onehot_labels
```

CUARTO PASO

NOS FIJAMOS EN EL CÓDIGO Y LO QUE NOS REGRESA SON 100 (CANCIONES) X 10 (CADA UNO DE ELLOS TIENE DIEZ NÚMEROS BINARIOS PARA REPRESENTAR LA CODIFICACIÓN ONE-HOT) DIMENSIONES.

LLAMAREMOS A LA VARIABLE **FEATURES,LABELS.** LA CUAL REGRESA TODAS LAS FUNCIONES APILADAS JUNTAS POR EL COMANDO DEVUELVEN **NP.STACK, ONEHOT_LABELS** EN UNA SOLA MATRIZ, ASÍ COMO LA MATRIZ **ONE-HOT.**

```
In [13]: features, labels = generate_features_and_labels()

         Processing 100 songs in blues genre...
         Processing 100 songs in classical genre...
         Processing 100 songs in country genre...
         Processing 100 songs in disco genre...
         Processing 100 songs in hiphop genre...
         Processing 100 songs in jazz genre...
         Processing 100 songs in metal genre...
         Processing 100 songs in pop genre...
         Processing 100 songs in reggae genre...
         Processing 100 songs in rock genre...
```

QUINTO PASO

```
In [14]: print(np.shape(features))
         print(np.shape(labels))

         training_split = 0.8

         # last column has genre, turn it into unique ids
         alldata = np.column_stack((features, labels))

         np.random.shuffle(alldata)
         splitidx = int(len(alldata) * training_split)
         train, test = alldata[:splitidx,:], alldata[splitidx:,:]

         print(np.shape(train))
         print(np.shape(test))

         train_input = train[:,:-10]
         train_labels = train[:,-10:]

         test_input = test[:,:-10]
         test_labels = test[:,-10:]

         print(np.shape(train_input))
         print(np.shape(train_labels))

         (1000, 25000)
         (1000, 10)
         (800, 25010)
         (200, 25010)
         (800, 25000)
         (800, 10)
```

Dividiremos el conjunto de datos en una división de entrenamiento y otra de prueba.

Decidamos la marca del 80% definida como training_split=0.8 para realizar una división de entrenamiento.

Por otra parte la división de de prueba es de 0.2.

Es decir el código analiza y graba el 0.20 y en base a este analiza el 0.8

SEXTO PASO

HAREMOS SHUFFLE, Y ANTES DE BARAJAR, DEBEMOS PONER LAS ETIQUETAS CON LAS CARACTERÍSTICAS PARA QUE NO SE MEZCLEN EN DIFERENTES ÓRDENES. COMO RESULTADO TENDREMOS CONJUNTOS DE PRUEBA .

In [15]:
```
model = Sequential([
    Dense(100, input_dim=np.shape(train_input)[1]),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
    ])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model.summary())

model.fit(train_input, train_labels, epochs=10, batch_size=32,
          validation_split=0.2)

loss, acc = model.evaluate(test_input, test_labels, batch_size=32)

print("Done!")
print("Loss: %.4f, accuracy: %.4f" % (loss, acc))
```

Link para encontrar la base de datos

https://www.kaggle.com/carlthome/gtzan-genre-collection