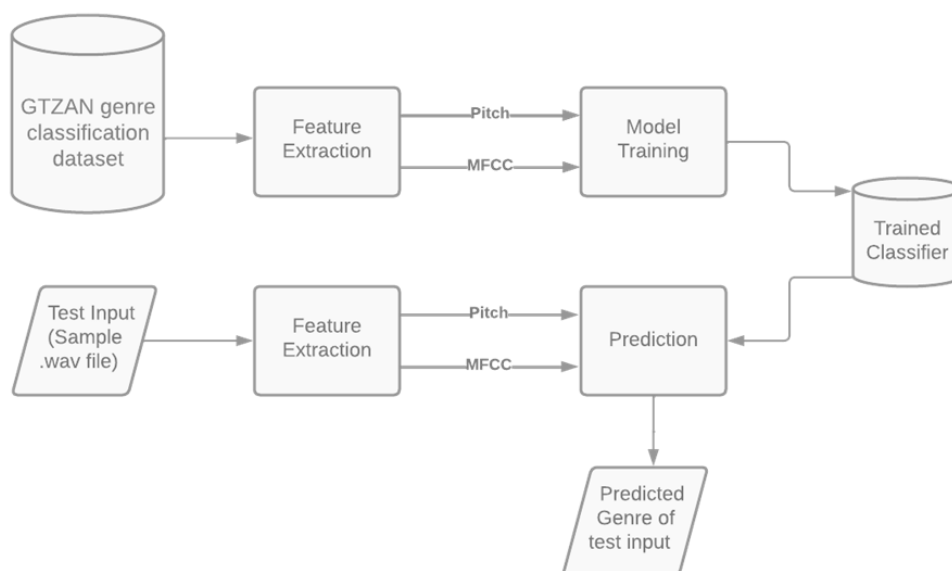


Informe: KNN: Clasificación de géneros musicales mediante Machine Learning

Este trabajo se fundamenta, mediante el uso de técnicas de **procesamiento de señales digitales y codificadores automáticos**, para extraer las características musicales de las canciones, para luego con estas características realizar una respectiva clasificación y agrupación musical dependiendo el género. **La naturaleza inherente** de la música puede tener **patrones recurrentes**, tempos establecidos, ritmo, melodía, armonía, timbre, estructura, dinámica, entre otros aspectos de la teoría musical, es por esta razón hace que los algoritmos de IA / ML sean buenos para el trabajo de comprender y clasificar música.

En el estudio, se utilizó el conjunto de datos **GTZAN**. El propósito de este estudio es comparar la extracción de características de resultados con los codificadores automáticos. y técnicas de procesamiento digital de señales.

El objetivo principal de este proyecto era **mejorar con éxito un algoritmo de ML** para poder clasificar cualquier byte de sonido de cualquier pieza musical como uno de los géneros del dataset y adicionalmente **nutrir el dataset con géneros latinos** y por supuesto mejorar a precisión del mismo.



Extracción de características

El primer paso es extraer características y componentes de los archivos de audio. Se incluye la identificación del contenido lingüístico y el descarte del ruido. Se utilizan los coeficientes cepstrales (MFCC) para este proyecto.

Cuando escuchamos música, se producen sonidos que llamamos ondas sonoras. Estas ondas sonoras están formadas por diferentes frecuencias, que se pueden imaginar como "alturas" de los sonidos. El MFCC, que significa "Coeficientes Cepstrales en Frecuencia Mel", es una técnica que nos ayuda a extraer características especiales de la música. El MFCC nos ayuda a capturar cosas como el tono, el timbre y el ritmo de la música.

Para hacer esto, la técnica MFCC divide la música en pequeños pedazos llamados "tramas" y luego analiza cada trama. Primero, convierte la onda sonora en una representación especial llamada "espectrograma", que muestra cómo se distribuyen las frecuencias en el tiempo.

Luego, el MFCC toma el espectrograma y aplica una serie de pasos matemáticos para seleccionar las características más importantes. Algunas de estas características pueden ser el volumen, la forma de las frecuencias y la energía de la música.

Una vez que hemos extraído estas características especiales, podemos usarlas para clasificar y agrupar diferentes canciones según su género musical.

Las señales de audio se dividen en cuadros más pequeños.

Se identifican las diferentes frecuencias presentes en cada cuadro.

Se separan las frecuencias lingüísticas del ruido.

Se toma la transformada discreta de coseno (DCT) de estas frecuencias para descartar el ruido, manteniendo solo una secuencia específica de frecuencias que tienen una alta probabilidad de información útil.

Entrenamiento del modelo

El algoritmo K-vecinos (KNN) se está utilizando porque diferentes investigaciones anteriores han demostrado que este algoritmo brinda los mejores resultados para el problema de clasificación de música. Se trata de uno de los algoritmos de aprendizaje automático más sencillos basados en la técnica de aprendizaje supervisado. Se asume la similitud entre el nuevo caso/datos y los casos disponibles y se clasifica el nuevo caso en la categoría más similar a las categorías disponibles.

Predicciones:

Este paso implica el uso de datos de prueba para probar y predecir el género del archivo de música de muestra en comparación con el modelo entrenado y obtener parámetros de precisión.

Implementación

La implementación implica los siguientes pasos:

1. Imports:

```
from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np
from tempfile import TemporaryFile
import os
import pickle
import random
import operator
```

```
import math
import numpy as np
```

2. Definición de una función para calcular la distancia entre vectores de características y encontrar vecinos:

```
def getNeighbors(trainingSet, instance, k):
    distances = []
    for x in range (len(trainingSet)):
        dist = distance(trainingSet[x], instance, k )+
distance(instance, trainingSet[x], k)
        distances.append((trainingSet[x][2], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

3. Identificación de los vecinos más cercanos.

```
def nearestClass(neighbors):
    classVote = {}

    for x in range(len(neighbors)):
        response = neighbors[x]
        if response in classVote:
            classVote[response]+=1
        else:
            classVote[response]=1

    sorter = sorted(classVote.items(), key = operator.itemgetter(1),
reverse=True)
    return sorter[0][0]
```

4. Definición de una función para la evaluación del modelo y las métricas de rendimiento.

```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range (len(testSet)):
        if testSet[x][-1]==predictions[x]:
            correct+=1
    return 1.0*correct/len(testSet)
```

6. Implementación de dependencias, es decir, el conjunto de datos de locales.

```

directory = 'genres_original/'
f= open("my.dat" , 'wb')
i=0

```

7. Extracción de características del conjunto de datos y exportación de las características extraídas en un archivo binario .dat ("my.dat").

```

for folder in os.listdir(directory):
    i+=1
    if i==10 :
        break
    for file in os.listdir(directory+folder):
        (rate,sig) = wav.read(directory+folder+"/"+file)
        mfcc_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy =
False) #(firt miy.dat)
        # mfcc_feat = mfcc(sig,rate ,winlen=0.020, nfft=1024,
nfilt=25, appendEnergy = False)
        covariance = np.cov(np.matrix.transpose(mfcc_feat))
        mean_matrix = mfcc_feat.mean(0)
        feature = (mean_matrix , covariance , i)
        pickle.dump(feature , f)

f.close()

```

8. División de entrenamiento y prueba en el conjunto de datos.

```

dataset = []
def loadDataset(filename, split, trset, tset):
    with open('my.dat','rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break
        for x in range(len(dataset)):
            if random.random() < split:
                trset.append(dataset[x])
            else:
                tset.append(dataset[x])
trainingSet = []
testSet = []
loadDataset('my.dat', 0.68, trainingSet, testSet)

```

9. Realización de una predicción utilizando KNN y obtención de la precisión en los datos de prueba.

```

leng = len(testSet)
predictions = []
for x in range (leng):
    predictions.append(nearestClass(getNeighbors(trainingSet
,testSet[x] , 7)))

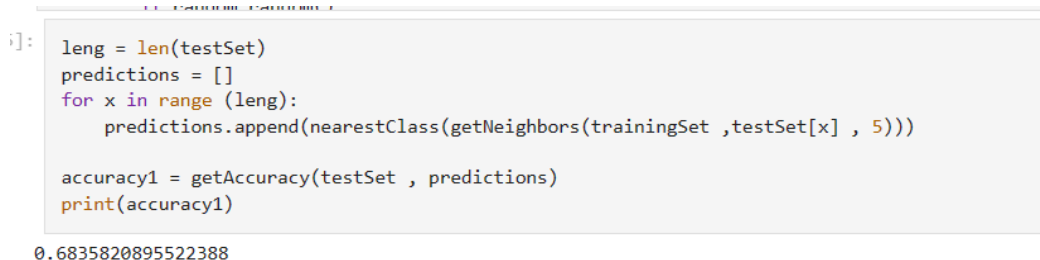
accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)

```

Resultados y discusión

Dado que todos los pasos se completaron con éxito, se logró implementar un algoritmo de aprendizaje supervisado para clasificar un archivo de música de prueba en su género designado adicionalmente se mejoró la precisión del mismo pasando de 68,35% a 75,8% ampliando la base de datos, actualizando algunas canciones y cambiando el valor de $k=5$ a $k=7$, y también solo analizando los géneros añadidos sin considerar el anterior dataset se llegó a obtener un precisión hasta de 83.7%, lo cual es un buen resultado para un modelo KNN con un conjunto de datos de esta escala.

Precisión de anterior algoritmo, $k=5$:



```


i]: leng = len(testSet)
   predictions = []
   for x in range (leng):
       predictions.append(nearestClass(getNeighbors(trainingSet ,testSet[x] , 5)))

   accuracy1 = getAccuracy(testSet , predictions)
   print(accuracy1)

```

0.6835820895522388

Precisión del algoritmo solo con la actualización de los nuevos géneros musicales, $k=7$:



```

PS C:\Users\Cont2\Desktop\ProySIC> & C:/Users/Cont2/Desktop/Python/Python310/python.exe c:/Users/Cont2/Desktop/ProySIC/ProySIC.py
0.8378378378378378
PS C:\Users\Cont2\Desktop\ProySIC>

```