WHISPER (Hugging Face)

Adrián Yared Armas de la Nuez

Contenido

1. Objetivo

El propósito de esta tarea es que el alumnado se familiarice con el modelo Whisper de OpenAI, disponible en Hugging Face, y lo utilice para transcribir el audio de una canción descargada. Se espera que el alumnado realice un análisis crítico del proceso, documente los pasos seguidos y proponga mejoras o dificultades encontradas.

2. Resolución de la actividad

2.1 Instalación

2.1.1 Torch

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install torch
```

2.1.2 Transformers

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install transformers
```

2.1.3 Ffmpeg

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install ffmpeg
```

2.1.4 Whisper

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install whisper
```

2.1.5 Accelerate

Requiere la dependencia accelerate para otras librerías anteriores.

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install accelerate
```

2.1.6 Librosa

Requiere la dependencia librosa para otras librerías anteriores.

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install librosa soundfile
```

2.2 Error y solución

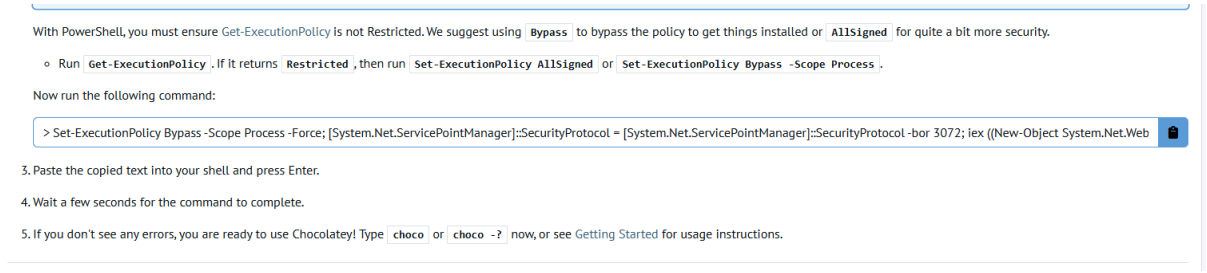
A la hora de la ejecución del código me decía que ffmpeg seguía sin estar instalado correctamente, por lo que Adolfo me recomendó la instalación de Chocolatey: https://chocolatey.org/install

2.2.1 ¿Qué es Chocolatey?

Chocolatey es un administrador de paquetes para Windows que permite instalar, actualizar y gestionar software desde la línea de comandos de forma rápida y automatizada.

2.2.2 Descarga de Chocolatey

Comando en la web:



Ejecución en la consola:

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProt
ocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadStri
ng('https://community.chocolatey.org/install.ps1'))
```

2.2.3 Descarga Ffmpeg a través de Chocolatey

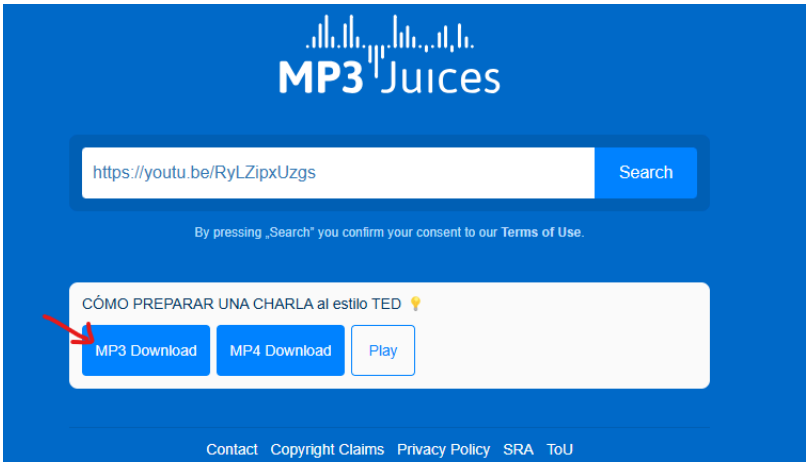
```
PS C:\WINDOWS\system32> choco install ffmpeg-full
```

2.3 Selección y descarga del audio

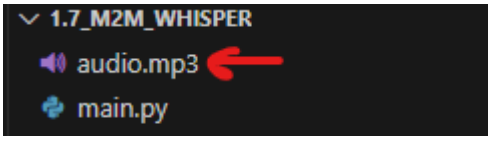
Video de la descarga:
https://youtu.be/RyLZipxUzgs

2.3.1 Descarga

A través de la web https://emp3juice.la/ descargué el audio del video:



2.3.2 Añado el audio al entorno de trabajo



2.4 Uso del modelo whisper

2.4.1 Instalación

Previamente hice la instalación del modelo:

```
PS C:\Users\Adrian\Desktop\1.7_M2M_whisper> pip install whisper
```

2.4.2 Código base

A través de la página del modelo, https://huggingface.co/openai/whisper-large-v3 , he obtenido el código base de funcionamiento del modelo:

```
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor,
pipeline
from datasets import load_dataset


device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else
torch.float32

model_id = "openai/whisper-large-v3"
```

```
model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True,
use_safetensors=True
)
model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    torch_dtype=torch_dtype,
    device=device,
)

dataset = load_dataset("distil-whisper/librispeech_long", "clean",
split="validation")
sample = dataset[0]["audio"]

result = pipe(sample)
print(result["text"])
```

2.5 Código final y explicación

2.5.1 Imports

```
import torch
import unicodedata  # Manejo de Unicode
import re  # Manipulación de texto
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor,
pipeline  # Herramientas de Hugging Face
from difflib import SequenceMatcher  # Comparar secuencias y calcular
la similitud
```

2.5.2 Configuración de dispositivo

Selecciona la GPU:

```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else
torch.float32
```

2.5.3 Carga del modelo

Carga el modelo preentrenado whisper-large-v3 de OpenAI con el tipo de datos adecuado, optimiza el uso de memoria en CPU y lo envía al dispositivo seleccionado (En mi caso la CPU):

```
# Cargar modelo
model_id = "openai/whisper-large-v3"
model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True,
use_safetensors=True
)
model.to(device)
```

2.5.4 Procesador

Carga los datos de entrada y salida:

```
processor = AutoProcessor.from_pretrained(model_id)
```

2.5.5 Pipeline

Crea un pipeline usando el modelo cargado y las características:

```
# Pipeline de reconocimiento de voz
pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    torch_dtype=torch_dtype,
    device=device,
)
```

2.5.6 Transcripción

Usa el Pipe para convertirlo a texto:

```
# Transcripción del audio
```

```
result = pipe("audio.mp3", return_timestamps=True)
texto = result["text"]
```

2.5.7 Formateo

Formateo los saltos de línea para no mostrar todo en una línea:

```
# Formatear el texto con saltos de línea
texto_con_saltos = texto.replace(", ", ".\n")
print("Texto transcrito con saltos de línea:\n", texto_con_saltos)
```

2.5.8 Muestra resultado

Muestra del resultado formateado:

```
Texto transcrito con saltos de línea:
  ¿Te gustaría saber cómo preparar tus charlas y presentaciones al estilo de TED? Pues entonces quédate en este vide
ia. Quizás has escuchado estas famosas charlas que se han vuelto tan virales impactando a millones de personas por t
os gracias a que son de mucho interés y a que sus ideas logran ser innovadoras. Pues tú también lo puedes hacer y a
Chris Anderson.
dice que no hay una fórmula de éxito.
sí hay dos pasos que debes seguir. El primero tiene que ver con un mensaje principal. Busca esa idea innovadora.
eso que te apasione.
pero que también logre sorprender al público. El segundo paso tiene que ver con que sigas una clara estructura. así
debes seguir un inicio.
un desarrollo y un final. En el tercer paso vas a incluir estos elementos que pueden enriquecer tu presentación.
como datos.
estadísticas o una buena historia. El paso número cuatro tiene que ver con que practiques la forma de cómo vas a abr
cuál ha sido tu mayor fracaso.
```

2.6 Código de comparación

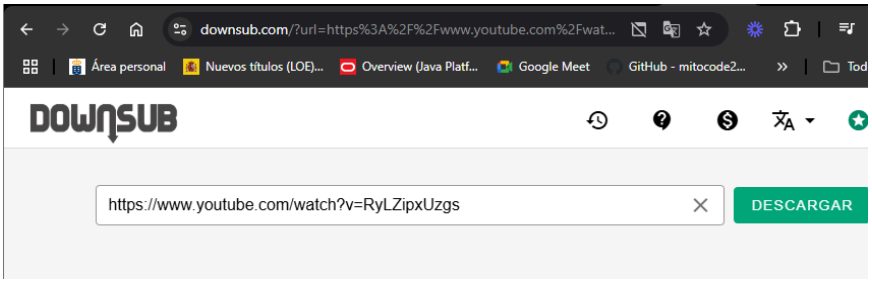
2.6.1 Normalización

Para comparar los textos de manera más exacta, creé el código que normaliza el texto (quité espacios, párrafos, convertí todo a minúscula y quite los signos de puntuación):

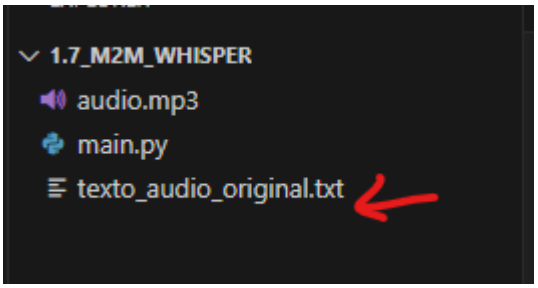
```
# Normalización del texto
def normalizar(texto):
    texto = unicodedata.normalize("NFD", texto)  # Descomponer
caracteres
    texto = texto.encode("ascii", "ignore").decode("utf-8")  # Eliminar
tildes
    texto = re.sub(r"[^\w\s]", "", texto)  # Quitar signos de
puntuación
    texto = texto.lower()  # Convertir a minúsculas
    texto = re.sub(r"\s+", "", texto)  # Eliminar todos los espacios y
saltos de línea
    return texto
```

2.6.2 Texto a comparar

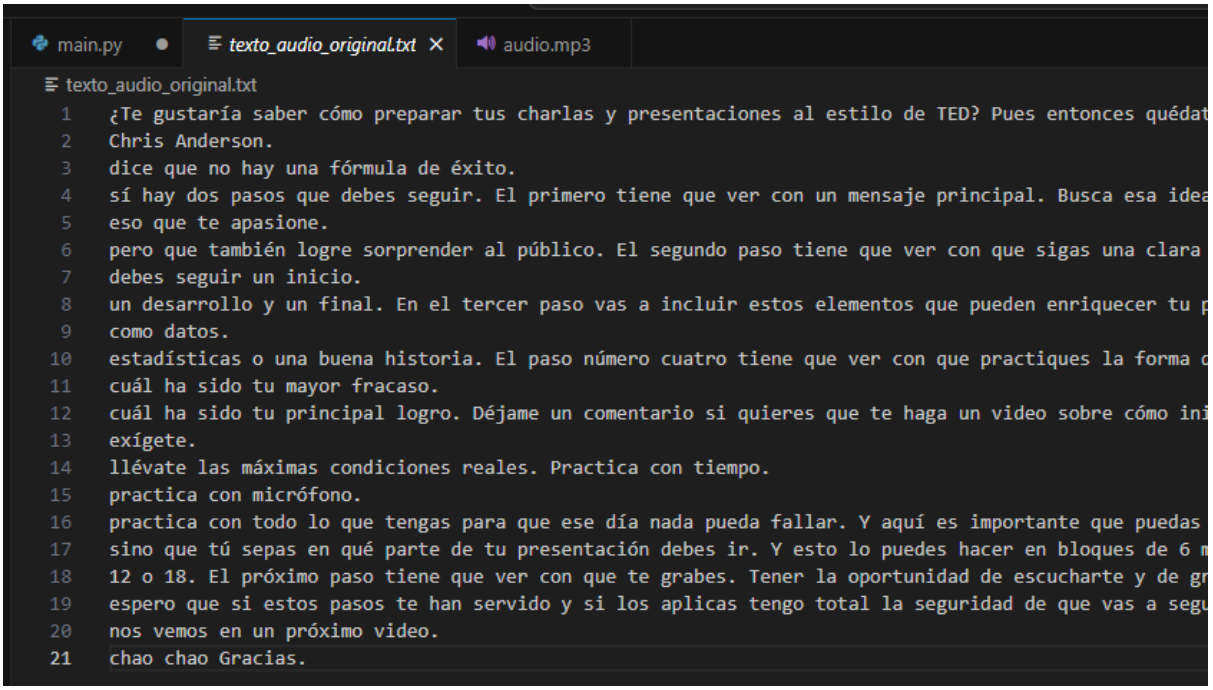
Puse el video en una página para descargar los subtítulos y tener una base:



Pero ya que esta web usará un modelo parecido y no se trata de comparar modelos, comparé con los subtítulos de youtube. Y corregí eso manualmente, y lo añadí al proyecto.



Resultado:



2.6.3 Lectura del archivo txt

```
# Leer y normalizar el texto original desde un archivo
with open("texto_audio_original.txt", "r", encoding="utf-8") as file:
```

```
    texto_original = file.read()
```

2.6.4 Normalización de los textos

Uso la función de normalizado que había creado anteriormente y le paso ambos textos, el original y el generado:

```
texto_normalizado = normalizar(texto)
texto_normalizado_original = normalizar(texto_original)
```

2.6.5 Comparar textos

Paso ambas cadenas de texto, y uso la librería SequenceMatcher (la cual compara dos cadenas de texto) para obtener el porcentaje de similitud

2.6.5.1 Código

```
# Comparar los textos
def calcular_similitud(texto1, texto2):
    matcher = SequenceMatcher(None, texto1, texto2)
    return matcher.ratio() * 100  # Convertir a porcentaje
```

2.6.5.2 Resultado

2.6.5.3 Calculo de error

Calcula el error restando al 100% el porcentaje de acierto:

```
similitud = calcular_similitud(texto_normalizado,
texto_normalizado_original)
error = 100 - similitud
```

2.6.5.4 Muestra similitud y error

Print del resultado en modo porcentual:

```
# Mostrar resultados
print(f"Porcentaje de similitud: {similitud:.2f}%")
print(f"Porcentaje de error: {error:.2f}%")
```

El porcentaje de acierto es extremadamente alto debido a la claridad y pausas entre palabras, así como la ausencia de ruido de entorno, como podría ser en el caso de la música de fondo en una canción.

```
Porcentaje de similitud: 100.00%
Porcentaje de error: 0.00%
```