Trabajo práctico

En este trabajo se realizan una serie de ejercicios de implementación de un reconocedor automático del habla basado en modelos estilo whisper como los vistos en los ejemplos de clase.

Los datos para la realización del trabajo están en:

https://dihana.unizar.es/~cadrete/valencia2526/fechas2.zip

Para dudas podéis contactarme en:

amiguel@unizar.es

Tarea 1. Reconocedor basado en whisper (6 puntos)

- Tarea 1.1: (1 punto)

Aprovechando lo visto en los cuadernos del taller se propone crear un tokenizador adaptado al vocabulario de la tarea. La idea es que sea orientado a palabra ya que hay pocas y simplifica el reconocimiento. Se puede reaprovechar el tokenizador DigitSumTokenizer que aparece en el cuaderno: parte2/run11. Implemente el tokenizador para la tarea:

fechas2/fechas2_train.es.csv

fechas2/fechas2_test.es.csv

- Tarea 1.2: (1 punto)

Reutilice el contenido del cuaderno part1/run5 para implementar un Trainset y Testset adaptados a la tarea fechas2 que estamos implementando. Compruebe de forma similar a como se hace en el cuaderno varios ficheros de train con aumento de ruido aditivo y reverberación con figuras de sus espectrogramas y la posibilidad de escucharlos.

- Tarea 1.3 (2 puntos)

Reutilice el tokenizador de la tarea 1.1, el trainset/testset de la tarea 1.2 y el contenido del cuaderno part3/run21 para implementar el entrenamiento y test de la tarea de fechas con los ficheros: fechas2_train.es.csv y fechas2_test.es.csv. Compruebe como se hace en el cuaderno la salida generada para algunos ejemplos de test y la visualización del modelo de atención y mida la tasa de error total WER en test. Si el modelo se entrena muy lento por falta de GPU no hace falta que se hagan muchas iteraciones completas de los datos, lo que se valora es el código. Puede usar también el frontend preentrenado que se muestra en el cuaderno parte3/run12 para mejorar las prestaciones.

- Tarea 1.4 (2 puntos)

Prepare ahora los pasos necesarios para implementar la versión en inglés del reconocedor: fechas2_train.en.csv y fechas2_test.en.csv y calcule el WER obtenido en test.

Tarea 2. Transformer con instrucciones (3 puntos)

- Tarea 2.1: (1 punto)

A continuación prepare un tokenizador que codifique mediante tokens especiales antes del mensaje de texto la acción que queremos ejecutar con el transformer de las 4 posibles:

- transcribe_es para transcribir directamente un audio en español a español
- transcribe_en para transcribir directamente un audio en inglés a inglés
- translate_en_es para traducir el mensaje con audio en inglés a español
- translate_es_en para traducir el mensaje con audio en español a inglés

- Tarea 2.2: (1 punto)

Prepare un dataset/dataloader nuevo que utilizando los ficheros y transcripciones de la tarea anterior prepare una combinación de las cuatro acciones posibles con ejemplos suficientes para entrenar el transformer, es decir, todas las combinaciones de audio en español/inglés y texto español/inglés con la instrucción correspondiente antes del texto.

- Tarea 2.3: (1 punto)

Entrene el modelo con el trainset y tokenizador anteriores y evalúe la tasa de error usando como test el conjunto de ficheros de audio, acciones y transcripciones de la tabla: fechas2_test_instruct.csv. Mida el resultado como WER en el texto reconocido.

A continuación, se muestra un fragmento del fichero de test donde la columna action indica el token de acción que hay que insertar antes de generar.

```
wav,action,txt
fechas2/test/fechas2_010000.es.wav,translate_es_en,the day after tomorrow thanks
fechas2/test/fechas2_010001.es.wav,transcribe_es,por favor en un par de días
fechas2/test/fechas2_010002.en.wav,translate_en_es,por favor en un par de días
fechas2/test/fechas2_010003.en.wav,translate_en_es,este jueves gracias
fechas2/test/fechas2_010004.en.wav,translate_en_es,por favor el miércoles siguiente
fechas2/test/fechas2_010005.en.wav,transcribe_en,next tuesday
```

Tarea 3. Transformer con llamadas a funciones (1 punto)

Una evolución interesante de los primeros LLMs fue la incorporación de llamadas a funciones para superar las limitaciones en cálculos concretos de los modelos de lenguajes.

- Tarea 3.1: (1 punto)

Para esta tarea final utilizamos los ficheros: fechas2_train_function.en.csv y fechas2_train_function.es.csv en la que se añade al final del texto a reconocer un separador '|' y un código de Python para llamar al sistema y pedir que ejecute esa función

```
wav,txt
fechas2/train/fechas2_000000.es.wav,por favor el siguiente jueves | next_day('thursday')
fechas2/train/fechas2_000001.es.wav,el viernes que viene | next_day('friday')
fechas2/train/fechas2_000002.es.wav,el viernes gracias | next_day('friday')
fechas2/train/fechas2_000003.es.wav,el martes por favor | next_day('tuesday')
fechas2/train/fechas2_000004.es.wav,mañana | relative_day(+1)
fechas2/train/fechas2_000005.es.wav,este viernes que viene gracias | next_day('friday')
fechas2/train/fechas2_000006.es.wav,pasado mañana | relative_day(+2)
```

Las dos funciones en Python se dan en el apéndice de este documento.

Prepare un sistema como el de la tarea 1 que entrene combinando las listas de los dos idiomas para que sea bilingüe dentro del contexto simplificado de esta tarea.

En test esta vez en lugar de medir la tasa de error WER lo que haremos es ejecutar el código dentro de la sección separada y comparar la fecha que genera con la de referencia de la tabla: fechas2_test_function.csv y evaluaremos la tasa de error.

A continuación, se muestran esas fechas que son nuestras etiquetas a predecir:

```
wav,txt
fechas2/test/fechas2_010000.en.wav,23/11/2025
fechas2/test/fechas2_010001.en.wav,23/11/2025
fechas2/test/fechas2_010002.es.wav,23/11/2025
fechas2/test/fechas2_010003.en.wav,27/11/2025
fechas2/test/fechas2_010004.en.wav,26/11/2025
fechas2/test/fechas2_010005.en.wav,25/11/2025
fechas2/test/fechas2_010006.es.wav,28/11/2025
```

Tarea opcional

Opcionalmente se puede implementar alguna de estas tareas opcionales con un máximo de 0.5 puntos que se sumarán a los anteriores sin pasar de 10.

- Tarea O.1 (0.25 puntos) Modifique el código de la generación de la frase de estilo greedy (utilizando la función argmax) por una función que muestree una distribución discreta que utilice la probabilidad de cada símbolo que devuelve el modelo en cada paso del bucle de generación

- Tarea O.2 (0.25 puntos) Modifique el código de la generación de la frase de estilo greedy (utilizando la función argmax) por una función que muestree una distribución discreta dentro de un subconjunto de las k con mejor probabilidad (configurable el valor de k).

- Tarea O.3 (0.5 puntos) Modifique el código de la generación de la frase para que se utilice un algoritmo de búsqueda de tipo beam search con varias opciones en paralelo.

- Tarea O.4 (0.5 puntos) La implementación que hemos usado en los ejemplos está muy simplificada a la hora de generar ya que vuelve a evaluar la secuencia completa desde el principio cada vez que se genera un nuevo símbolo. Este sobrecoste no sería viable en un sistema real y en el que el contexto es más largo. Implemente un sistema de caché para keys y values para que estos tensores aprovechen los cálculos anteriores

Entrega

La entrega consistirá en un archivo zip por pareja o individual que contenga:

- Una breve memoria (max. 2 páginas) donde se expliquen las soluciones presentadas y los resultados obtenidos. Es importante que se indiquen los nombre completos de los autores de la entrega para calificar el trabajo.
- Los cuadernos que solucionan cada tarea, lo recomendado es un cuaderno por tarea.

Apéndice

Funciones para ejecutar a partir del texto reconocido en la tarea 3:

```python
from datetime import datetime, timedelta

def relative_day(days_to_add, current_date='21/11/2025'):
    date_format = "%d/%m/%Y"
    current = datetime.strptime(current_date, date_format)
    new_date = current + timedelta(days=int(days_to_add))
    return new_date.strftime(date_format)


def next_day(day_name, current_date='21/11/2025'):
    date_format = "%d/%m/%Y"
    current = datetime.strptime(current_date, date_format)

    # Map day names → ISO weekday codes
    day_name = day_name.strip().lower()
    day_map = {
        "monday": 1,
        "tuesday": 2,
        "wednesday": 3,
        "thursday": 4,
        "friday": 5,
        "saturday": 6,
        "sunday": 7,
    }

    if day_name not in day_map:
        raise ValueError(f"Invalid day name: {day_name}")

    target_code = day_map[day_name]
    days_ahead = (target_code - current.isoweekday()) % 7
    if days_ahead == 0:
        days_ahead = 7  # always next occurrence

    new_date = current + timedelta(days=days_ahead)
    return new_date.strftime(date_format)
```