Trabajo Práctico II

Predicción de puntuación y capitalización en texto normalizado

Objetivo general

El objetivo de este trabajo práctico es que los estudiantes diseñen, implementen, entrenen y comparen al menos tres modelos de secuencia distintos para resolver la tarea conjunta de:

- Predicción de **puntuación**
- Predicción de **capitalización**

Por ejemplo:
Entrada: "cuándo salimos para la uba ya se hace tarde por dónde andarán"
Salida: "¿Cuándo salimos para la UBA? Ya se hace tarde. ¿Por dónde andarán?"

Motivación

Este tipo de sistema tiene aplicaciones reales en procesamiento de lenguaje natural. Un caso típico es el postprocesamiento de la salida de sistemas de reconocimiento automático del habla (ASR, por sus siglas en inglés). Estos sistemas suelen producir texto plano, en minúsculas y sin puntuación, lo cual dificulta su lectura y comprensión.

El objetivo de este TP es construir modelos que permitan "reconstruir" ese texto enriquecido con puntuación y mayúsculas, mejorando su utilidad en tareas posteriores como: Resumen automático, Análisis de sentimiento o Traducción automática.

Tareas a resolver

Dado un string texto en minúscula (de tamaño no determinado) y sin puntuación y devolver las siguientes predicciones para cada palabra:

- **Puntuación**:
  - **¿** → signo de pregunta de apertura
  - **,** → coma
  - **.** → punto (de división de oraciones)
  - **?** → signo de pregunta de cierre
  - → sin puntuación
- **Capitalización**:
  - **0:** todo en minúsculas (ej: ***"hola"***)
  - **1**: primera letra en mayúscula (ej: ***"Hola"***) (incluye palabras de 1 letra)
  - **2**: algunas (pero no todas) letras en mayúscula (ej: ***"McDonald's"*** o ***"iPhone"***)
  - **3**: todo en mayúsculas (ej.: ***"ONU", "NASA", "UBA"***) (más de una letra)

Entrada del modelo

Una instancia para nuestro modelo será un texto normalizado que puede contener parte de una o varias oraciones. Estas estarán conformadas por una secuencia de tokens generada por el tokenizador ***bert-base-multilingual-cased***, disponible en HuggingFace.

Explicación del tokenizador BERT

El tokenizador BERT divide el texto en unidades más pequeñas llamadas "tokens". A diferencia de la división simple por espacios, BERT utiliza un enfoque más sofisticado:

1. **Tokens completos**: Palabras frecuentes que se mantienen intactas (ej.: "el", "a").
2. **Sub-tokens**: Fragmentos de palabras marcados con `##` que indican que son continuación de un token previo.

Por ejemplo, la palabra "cuándo" se divide en los tokens `cu`, `##án`, `##do` porque el tokenizador no la reconoce como una unidad completa, pero sí identifica estos fragmentos en su vocabulario.

⚠️ **Importante**:

- Los estudiantes NO necesitan implementar este tokenizador.
- Para la evaluación, proporcionaremos los textos ya tokenizados en el formato esperado.
- Las predicciones deben realizarse a nivel de token (no de palabra completa).

Instalación:

```
pip install transformers
```

Ejemplo de tokenización en español:

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
texto = "a qué hora pasa el ciento siete"
tokens = tokenizer.tokenize(texto)

print(tokens)
# ['a', 'qué', 'hora', 'pasa', 'el', 'cien', '##to', 'siete']

tokenizer.convert_tokens_to_ids(tokens)
# [169, 38188, 24301, 26088, 10125, 99485, 10340, 28394]
```

Embeddings:
Deberán utilizar esta manera de extraer embeddings a partir de tokens.
(ver notebook visto en clase práctica 7 para un ejemplo más completo)

```
from transformers import BertTokenizer, BertModel
import torch

model_name = "bert-base-multilingual-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertModel.from_pretrained(model_name)
model.eval()

def get_multilingual_token_embedding(token: str):
    """
    Devuelve el embedding (estático) para el token.
    """
    token_id = tokenizer.convert_tokens_to_ids(token)

    if token_id is None or token_id == tokenizer.unk_token_id:
        print(f"❌ El token '{token}' no pertenece al vocabulario de multilingual
BERT.")
        return None

    embedding_vector = model.embeddings.word_embeddings.weight[token_id]

    print(f"✅ Token: '{token}' | ID: {token_id}")
    print(f"Embedding shape: {embedding_vector.shape}")
    return embedding_vector
```

Detalles de formato

- **Puntuación inicial**: Se asigna al token que inicia inmediatamente después de dicha puntuación.
    - El signo de apertura `¿` se asigna como puntuación inicial al primer token de la pregunta.

- **Puntuación final**: Se asigna al token que aparece inmediatamente antes de dicha puntuación.
    - Los signos de cierre `.`, `?` se asignan como puntuación final al último token de la frase/pregunta.
    - La coma `,` se asigna como puntuación final al token que la precede.

- **Un mismo token puede tener puntuación inicial y final**:
    - Para casos como `"¿Qué?"` suponiendo `tokens("qué") == [qué]` :
        - `¿` es puntuación inicial del token `qué`
        - `?` es puntuación final del token `qué`

- **Capitalización**:
    - Se asigna a todos los tokens de la palabra. Por ejemplo, para `['m', '##c', '##don', '##ald']` esperaríamos ver la asignación `[2, 2, 2, 2]`, respectivamente. Otro ejemplo sería `['me', 'g', '##ust', '##ar', '##ía', 'i', '##r', 'a', 'la', 'na', '##sa']` en donde la predicción esperada sería `[1, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3]`

Salida esperada

La salida será un archivo .csv con una línea por token, incluyendo las predicciones de puntuación inicial, puntuación final y capitalización.

📄 **Ejemplo**

Entrada original:
"cuándo vamos a mcdonald's ellos no vienen hoy dónde están ahora"

Tokens generados (usando BERT bert-base-multilingual-cased):
['cu', '##ánd', '##o', 'va', '##mos', 'a', 'm', '##c', '##dona', '##ld', "'", 's', 'ellos', 'no', 'viene', '##n', 'hoy', 'dó', '##nde', 'están', 'ahora']

📋 **Predicciones esperadas (CSV)**
(para el caso de la puntuación “,” esta deberá estar encerrada entre comillas)

```
instancia_id,token_id, token,punt_inicial,punt_final,capitalización
1,10854,cu,¿,,1
1,101439,##ánd,,,1
1,10133,##o,,,1
1,10321,va,,,0
1,13386,##mos,,,0
1,169,a,,,0
1,181,m,,,2
1,10350,##c,,,2
1,64674,##dona,,,2
1,12620,##ld,,,2
1,112,',,,2
1,187,s,,?,2
1,16169,ellos,,,1
1,10192,no,,,0
1,12266,viene,,,0
1,10115,##n,,,0
1,25235,hoy,,.,0
1,82639,dó,¿,,1
1,11382,##nde,,,0
1,16037,están,,,0
1,29291,ahora,,?,0
```

Nota: En la evaluación, proporcionaremos múltiples instancias en este mismo formato (sin las columnas a predecir), cada uno con su identificador único (`instancia_id`). Los estudiantes deberán procesar cada párrafo y agregar la información de puntuación y capitalización para cada instancia, manteniendo la referencia al original mediante este ID.

Texto esperado reconstruido:
"¿Cuándo vamos a McDonald's? Ellos no vienen hoy. ¿Dónde están ahora?"

(Notar que McDonald's debe ser predicho como capitalización compuesta, pero no es trivial saber qué letras capitalizar. Sugerencia: utilizar diccionarios de palabras vistas en entrenamiento, y si no fue vista, resolver como les parezca)

Evaluación

En la fecha de entrega se liberará un conjunto de textos preprocesados, y cada grupo deberá generar las predicciones correspondientes con sus modelos.

Cada tarea será evaluada por separado con métricas estándar:

Tarea	Métrica de evaluación
Puntuación inicial	F1 macro sobre 2 clases: `¿`, `" "`
Puntuación final	F1 macro sobre 4 clases: `,`, `.`, `?`, `" "`
Capitalización	F1 macro sobre 4 clases: 0, 1, 2, 3

Pese a que se entregan los resultados a nivel token, la evaluación será llevada a cabo a nivel palabra. Es decir, si una palabra tiene múltiples tokens, no afectará más al resultado que una que tenga uno o pocos.

Modelos a implementar

Cada grupo debe desarrollar, entrenar y comparar **al menos** los siguientes enfoques (pueden agregar otros):2

a) Algún algoritmo clásico (no para procesamiento de secuencias, ej: RF)
b) RNN estándar (si quieren con celdas LSTM o GRU).
c) RNN bidireccionales.

Si ya completaron satisfactoriamente el trabajo con las actividades anteriores y desean profundizar explorando nuevas alternativas, pueden consultar con los docentes si está permitido continuar con alguna de las siguientes opciones:

d) Opción 1: Realizar fine-tuning de BERT.
e) Opción 2: Utilizar la API de Google Gemini u otro modelo de lenguaje (LLM).

Requisitos del dataset

- **No hay** restricciones sobre la fuente de los datos de entrenamiento.
- Tengan en cuenta que deberán **preprocesar y tokenizar** (con bert-base-cased).
- Tengan en cuenta que tienen que extraer embeddings con el código provisto.

Entregables

- **Archivo de predicciones en formato .csv** (como el ejemplo)
- **Código fuente y scripts de: Entrenamiento, Inferencia y Evaluación**
- **Póster técnico (en PDF)** que resuma el trabajo y los resultados incluyendo:
    - Descripción del dataset utilizado
    - Mini descripción de la Arquitectura de los modelos implementados
    - Resultados. Sugerimos intentar contestar preguntas tales como:
      ¿Conviene compartir pesos entre tareas? ¿Qué clase le costó más a qué modelo? ¿Cómo se relaciona con su arquitectura?
    - Discusión y Conclusiones

Nota: Pueden utilizar herramientas de inteligencia artificial, pero deberán poder defender tanto los hallazgos como el código proporcionado (sin mucha profundidad en el código)