A2: Python a Docstring

Ana Valentina López Chacón

Traducción Automática
MIARFID, UPV

Enero, 2025

1. Descripción de la Práctica

Se buscó realizar diferentes experimentos sobre el *dataset* de código en python a docstring [1], para esta tarea se exploraron modelos pre-entrenados, técnicas de *prompting* y métricas de evaluación que permiten medir la calidad de la traducción, algunas especializadas en tareas de traducción de código.

1.1. Objetivos

Se plantearon los siguientes objetivos:

- Obtener métricas de evaluación aptas para el problema de traducción empleando como *baseline* diferentes modelos pre-entrenados.
- Explorar técnicas de *finetuning* e ingeniería de *prompts* aplicadas al *dataset* escogido.
- Ajustar los resultados del objetivo anterior y realizar una exploración de parámetros.

2. Análisis de los Datos

El *dataset* completo viene configurado para diversos lenguajes de programación: go, java, javascript, php, python, ruby. En este caso se consideró como lenguaje de origen únicamente python. De igual forma, se encuentra dividido en tres particiones: train (251820 muestras), validation (13914 muestras) y test (14918 muestras), cada una de estas contiene su correspondiente listado de atributos (ver Cuadro 1). Se observó que los datos cuentan con una buena variabilidad tanto de dificultad de código como longitud de las secuencias. Se consideraron principalmente los atributos de `code` y `docstring`.

Nombre del Campo	Tipo	Descripción
`id`	`int32`	Índice de la muestra
`repo`	`string`	Repositorio: el propietario/repo
`path`	`string`	Ruta completa al archivo original
`func_name`	`string`	Nombre de la función o método
`original_string`	`string`	Cadena original antes de tokenizar o analizar
`language`	`string`	Nombre del lenguaje de programación
`code`	`string`	Código o función, parte de `original_string`
`code_tokens`	`Sequence[string]`	Versión tokenizada del código
`docstring`	`string`	Comentario o docstring de nivel superior, si existe
`docstring_tokens`	`Sequence[string]`	Versión tokenizada del docstring
`sha`	`string`	SHA del archivo
`url`	`string`	URL del archivo

Cuadro 1: Características del conjunto de datos `code_x_glue_ct_code_to_text`.

3. Experimentación

Todos los experimentos se realizaron usando *google collab*, debido a los recursos limitados se optó por tomar subconjuntos de las muestras de train, validación y test, de 200, 1000 y 1000 respectivamente. Posteriormente y dependiendo de las limitaciones del modelo pre-entrenado se realizó un filtro para el máximo número de tokens. Adicional a las métricas de BLEU y COMET se reportaron valores de ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*), tipicamente usado en tareas de generación de texto y traducción de código a docstring, que mide el *overlap* entre la predicción y la referencia a nivel de n-grama. El ROUGE-1 mide el *overlap* de unigramas, el ROUGE-2 a nivel de bigrama y el ROUGE-L mide la subsecuencia común de mayor longitud.

3.1. Baselines de Modelos Pre-entrenados

En esta parte se cargaron cuatro modelos pre-entrenados y se usaron para realizar las predicciones que posteriormente fueron ajustadas. NLLB (No Language Left Behind) es un modelo de traducción desarrollado por Meta, se centra en idiomas de bajos recursos, ofreciendo alta calidad para más de 200 idiomas y dialectos. LLAMA-2, también desarrollado por Meta, es un modelo grande de lenguaje diseñado para tareas generales de entendimiento y generación de lenguaje natural, la estrategia 1-shot consiste en proveer un ejemplo simple de la tarea a realizar.

Ahora bien, con respecto a modelos enfocados a tareas relacionadas a códigos se consideró CodeT5-large, desarrollado por Salesforce es un modelo basado en transformers optimizado para tareas *sequence-to-sequence* como resumen o generación de código. Por último, se cargó el modelo desarrollado por UCLANLP y basado en BART, PLBART-large, entrenado con muestras tanto de código como de lenguaje natural. Para cada uno de los modelos pre-entrenados se realizó una predicción de un subconjunto de las muestras de test y se reportaron sus métricas de evaluación (ver Cuadro 2).

Modelo	Max Token	BLEU	COMET	ROUGE-1	ROUGE-2	ROUGE-L
NLLB	1024	23.8	59.06 %	48.97 %	44.82 %	48.3 %
LLAMA-2	100	6.9	53.01 %	14.41 %	13.08 %	14.46 %
Code T5	512	3	47.53 %	20.83 %	12.53 %	19.24 %
PLBART	512	20.3	51.69 %	35.91 %	30.68 %	34.91 %

Cuadro 2: Resultados de Baselines de Modelos Pre-entrenados.

Dado que LLAMA-2 es un modelo pesado se determinó que el número máximo de tokens como el tamaño del batch se debían reducir lo más posible para evitar agotar todos los recursos de GPU disponibles, probablemente no esta recibiendo muestras lo suficientemente significativas para generar un resultado destacable.

3.2. Ajuste de Modelos Pre-entrenados

Se realizaron cuatro experimentos de ajuste o *finetuning* con el fin de mejorar los resultados obtenidos en la etapa anterior, logrando adaptar los modelos pre-entrenados a la tarea de traducción escogida. Para todos los experimentos se dejaron los mismos parámetros a excepción del número de tokens, sin embargo se parten de las mismas 200 muestras de train y 1000 de validación. Por temas de recursos computacionales se fijaron solo 3 épocas y los resultados se consignaron en la siguiente tabla 3:

Modelo	Max Token	BLEU	COMET	ROUGE-1	ROUGE-2	ROUGE-L
NLLB	1024	67.4	85.08 %	86.88 %	84.75 %	86.64 %
LLAMA-2	100	12.6	54.93 %	25.11 %	23.05 %	25.14 %
Code T5	512	71.3	82.48 %	83.18 %	81.69 %	82.84 %
PLBART	512	85.9	88.35 %	92.33 %	90.9 %	92.38 %

Cuadro 3: Resultados de Ajuste de Modelos Pre-entrenados.

3.3. Exploración de Parámetros

Adicional a los resultados de *finetuning* se optó por realizar una exploración de parámetros de entrenamiento, inferencia y método peft empleando la librería de `optuna`. Los resultados se encuentran en la siguiente tabla 4:

Peft	BLEU	COMET	ROUGE-1	ROUGE-2	ROUGE-L
LoRA	90.3	89.77 %	94.47 %	93.44 %	94.49 %
PrefixTuning	4.4	40.16 %	12.63 %	8.03 %	11.19 %

Cuadro 4: Resultados de la Exploración de Parámetros.

Para el experimento con LoRA, se obtuvieron los siguientes parámetros: {`'learning_rate': 0.0005691874384094053, 'batch_size': 8, 'lora_r': 32, 'lora_alpha': 32, 'lora_dropout': 0.1900416423493052`}. Para PrefixTuning: {`'learning_rate': 0.00022685046590385107, 'batch_size': 4, 'prefix_length': 30`}.

4. Conclusiones

Para finalizar este informe podemos destacar las siguientes conclusiones:

- Los modelos pre-entrenados especializados en tareas relacionadas con el código, como PLBART-large y CodeT5-large, mostraron un rendimiento superior en las métricas reportadas después de realizar el ajuste en comparación a modelos más generales o generativos.
- PLBART alcanzó los mejores resultados en la mayoría de las métricas tras el ajuste fino, lo que demuestra que su entrenamiento con código y lenguaje natural lo hace especialmente efectivo para esta tarea.
- El modelo NLLB, aunque no está especializado en código, mostró un notable rendimiento tras el ajuste gracias a su capacidad para manejar secuencias largas y diversos dominios.
- La exploración con LoRA resultó ser más efectiva que PrefixTuning, esto resalta que LoRA es más adecuado para ajustar modelos como PLBART para tareas específicas de código con un menor consumo de recursos, ya que solo se entrenan unas pocas capas del modelo.
- Las restricciones de GPU y memoria limitaron el tamaño del conjunto de datos y el número máximo de tokens para algunos modelos, como LLAMA-2, lo que posiblemente afectó su rendimiento. A pesar de estas limitaciones, las técnicas de *finetuning* demostraron ser eficaces para obtener resultados competitivos incluso con conjuntos de datos pequeños.

Referencias

[1] Husain, H., Wu, H. H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2019). *Codesearchnet challenge: Evaluating the state of semantic code search.* arXiv preprint arXiv:1909.09436. `https://huggingface.co/datasets/google/code_x_glue_ct_code_to_text`