Generación de comentarios a partir de código fuente utilizando Transformers

Cristian Vincenzini and Sandra Roger

Grupo de Investigación en Lenguajes e Inteligencia Artificial (GILIA), Facultad de Informática. Universidad Nacional del Comahue. Neuquén
cristian.vincenzini@est.fi.uncoma.edu.ar, roger@fi.uncoma.edu.ar

Resumen En este trabajo se utilizará un modelo de generación de lenguaje natural (GLN) para crear comentarios a partir de código fuente. Para esto, se aprovechará la técnica de transferencia de conocimiento en un modelo basado en mecanismo de atención, utilizando pequeños conjuntos de datos de entrenamiento sobre grandes modelos ya entrenados y posteriormente se analizarán los resultados obtenidos.

Keywords: Generación del Lenguaje Natural, Aprendizaje Automático, Código, Comentarios

Contexto Este trabajo se desarrolla en el marco de la tesis final de la carrera Licenciatura en Ciencias de la Computación.

1. Introducción

Los programadores utilizan los comentarios con diversos fines: para especificar los requerimientos del software que desarrollan, comunicarse con otros desarrolladores, informar tareas que faltan realizar, pero fundamentalmente se utilizan para describir la funcionalidad de algún fragmento de código. Este tipo de comentarios en particular, contienen una gran cantidad de información que puede aprovecharse para mejorar el mantenimiento, la fiabilidad del software y para facilitar la comprensión del programa [1,8]. La generación de lenguaje natural (GLN) a partir de código atañe a proveer de manera automatizada una descripción resumida de un fragmento de código. Los modelos de GLN basados en el mecanismo de atención -conocidos como Transformers- se presentan aún como un campo poco explorado. Hasta la fecha, distintas arquitecturas entrenadas con millones de datos para diferentes tareas se hicieron disponibles para uso público. Una de las características de estas arquitecturas es que permiten refinar su aprendizaje para realizar una tarea específica, mecanismo que se conoce como *transferencia de aprendizaje*. Mediante esta técnica es posible utilizar estos grandes modelos ya entrenados y especializarlos, obteniendo resultados aceptables en un tiempo considerablemente inferior al de entrenar un modelo desde cero.

Para este trabajo utilizaremos un transformer conocido como T5 (2020)[7]. Este modelo permite procesar múltiples tareas relacionadas a la GLN. La tarea a desarrollar para este trabajo será la generación de comentarios a partir de código fuente. Para ello tomaremos un modelo ya entrenado que será refinado sobre un conjunto de datos propio y finalmente analizaremos los resultados obtenidos.

2. Trabajos Relacionados

En los últimos años han surgido varias arquitecturas basadas en el mecanismo de atención, y utilizadas para diversas tareas en el campo de la ingeniería del software. Tenemos por ejemplo CodeBERT[3], un modelo de propósito general que permite, entre otras tareas, buscar código utilizando lenguaje natural, generar documentación de código fuente, etc. Otro ejemplo es TransCoder[4], un transpilador que permite la traducción de un código fuente a otro entre diversos lenguajes de programación. Estos trabajos han alcanzado el estado del arte.

3. Nuestra Propuesta

El modelo de lenguaje utilizado se basa en la arquitectura de transformadores de codificación-decodificación de CodeTrans [2]. En dicho trabajo se utilizaron tres tamaños del modelo T5 para realizar los entrenamientos: *small*, *base* y *large* (60, 220 y 770 millones de parámetros respectivamente). Estos modelos fueron entrenados para diferentes tareas en varios lenguajes de programación utilizando transferencia de aprendizaje. La transferencia de aprendizaje consiste en dos etapas. Una etapa inicial de entrenamiento auto-supervisado donde se utilizan datos sin etiquetar y una segunda etapa, conocida como *fine-tuning* donde el modelo se entrena para una tarea específica utilizando datos etiquetados. Otra técnica utilizada en CodeTrans es el entrenamiento multi-tarea. Esta estrategia consiste en entrenar un modelo en mútiples tareas, utilizando datos etiquetados y sin etiquetar. Esta metodología permite, además, realizar un *fine-tuning* posterior a través de la transferencia de aprendizaje.

En este trabajo tomamos la arquitectura T5 ya entrenada con los datos de CodeTrans y realizamos *fine-tuning* sobre cada uno de ellos, utilizando sets de datos propios para realizar diversos experimentos. En particular, tomamos los modelos realizados por transferencia de aprendizaje (TF) y multi-tarea (MT) para los tres tamaños considerados: small, base y large.

4. Configuración de la Experimentación

Utilizamos dos conjuntos de datos para desarrollar los experimentos. Todos contienen 120 líneas de entre las cuales se seleccionaron de forma aleatoria 20 líneas para formar un conjunto de datos de test, el resto se utilizó para el entrenamiento del nuevo modelo. Los datos consisten en tuplas que referencian pares de funciones -en algún lenguaje de programación- y el comentario en inglés que describe la funcionalidad de la misma.

Seleccionamos a GO como primer *set* de datos. GO es un lenguaje de programación imperativo[1], desarrollado por Google en el año 2009. Las funciones y sus respectivos comentarios fueron extraídos en su mayor parte utilizando GitHub

[1] `https://go.dev/`

Tabla 1: Resultados de la evaluación de nuestro *dataset test* en todas las tareas para el lenguaje GO. Se utilizó BLEU-4 tanto para evaluar los modelos de CodeTrans[2] como los propios.

	Nuestra salida	Code-Trans
go-tf-s	**3,53**	2,75
go-tf-base	**13,96**	11,34
go-tf-large	**20**	12,55
go-mt-tf-s	12,77	16,44
go-mt-tf-base	11,19	11,69
go-mt-tf-large	**17,56**	9,95

Copilot [2]. El siguiente *set* de datos seleccionado es PROLOG. PROLOG es un lenguaje lógico que utiliza predicados como elementos de ejecución. Los datos fueron tomados de diversas fuentes. Durante la selección, se evitó utilizar predicados provenientes de módulos o paquetes de terceros, también se evitó usar sintaxis de gramáticas de cláusulas definidas.

5. Resultados y discusión

Se llevaron a cabo experimentos para evaluar las tareas de *Transfer Learning* (TF) y de *Multi-Task Learning* con *Fine-Tune* (MT-TF), debido a que en ambas tareas se emplea la utilización de un *Fine-Tune*. En cada tarea, se realizaron experimentos para los modelos de tres tamaños distintos: pequeño, base y grande (*S*, *B* y *L*, respectivamente). Para evaluar estas tareas se utilizó la medida BLEU-4 [6]. Los experimentos relacionados al lenguaje GO se realizaron para comparar cómo se comportaba nuestro *finetune*, construido con el corpus descripto en 4 con respecto a los modelos de CodeTrans de [2] utilizando nuestro *dataset-test*. La Tabla 1 muestra los resultados. El desarrollo del *Fine-tune* sobre nuestro corpus brindó buenos resultados en la tarea de TF y solo en el mt-tf-large se logró una mejora.

Por otro lado, se hicieron evaluaciones de estas mismas tareas en los modelos pequeños y bases, como así también los modelos grandes en el lenguaje PROLOG. Cabe destacar que CodeTrans no cuenta con este lenguaje. Al igual que con el lenguaje GO, se construyó un corpus tanto para realizar el *fine-tune* como el *dataset* utilizado en el testeo. El objetivo fue probar cómo se comportaban los modelos para un lenguaje no contemplado previamente. Además, elegimos PROLOG por ser un lenguaje de programación declarativo, a diferencia de los otros lenguajes utilizados. En esta oportunidad se utilizaron dos métricas: el BLEU y el ROUGE [6]. Dentro de las variantes de la medida ROUGE nos concentramos en la ROUGE-L, con la ventaja de que puede capturar la estructura del nivel de oración de una manera natural.[5]

La Tabla 2 muestra la evaluación realizada para el lenguaje de programación PROLOG en las dos tareas y modelos (*S, B, L*) en los cuales se hicieron, también, las evaluaciones de GO. Como se puede apreciar los valores del BLEU-4 son de un dígito en su totalidad. Siendo el mejor resultado para el *Transfer Learning* con un modelo *Large*. Por el contrario, las medidas ROUGE-L arrojan valores

[2] `https://github.com/features/copilot`

Tabla 2: Resultados de la evaluación de todas las tareas para el lenguaje de programación PROLOG.

MEDIDA	PRO-TF-S	PRO-TF-B	PRO-TF-L
ROUGE-L	24.4537	28.2623	**31.4393**
BLEU-1	18.79	20.56	20.56
BLEU-2	10.71	12.21	13.43
BLEU-3	5.74	6.27	9.07
BLEU-4	0	0	**6.05**
	PRO-MT-TF-S	PRO-MT-TF-B	PRO-MT-TF-L
ROUGE-L	27.5264	29.0601	28.5726
BLEU-1	20.56	19.85	17.02
BLEU-2	13.43	11.6	9.53
BLEU-3	9.07	7.33	5.72
BLEU-4	5.1	4.34	0

Tabla 3: Ejemplo de una pregunta del test. La referencia humana fue tomada del recurso donde se extrajo el programa.

Programa	`circle(X, Y, R):- number(X), number(Y), number(R), R >0.`
Referencia Humana	succeeds if the item represents a valid circle x,y represents the centrepoint of a circle on an x,y plane r represents the radius of the circle.
PRO-TF-BASE	computes a circle from the coordinates.
BLEU-4 y BLEU-3	0.0
BLEU-2	6.79
BLEU-1	11.54

Tabla 4: Ejemplo de una pregunta del test. La referencia humana fue tomada del recurso donde se extrajo el programa.

Programa	`reverse([], Z, Z).` `reverse([H|T], Z, Acc) :- reverse(T, Z, [H|Acc]).`				
Referencia Humana	reverses a list of any length.	BLEU-1	BLEU-2	BLEU-3	BLEU-4
pro-tf-s	reverses the array of elements in the array z.	20.22	0.0	0.0	0.0
pro-tf-b	reverses the order of the elements in the list.	38.46	17.9	0.0	0.0
pro-tf-l	reverses an array.	16.67	0.0	0.0	0.0
pro-mt-tf-s	reverses the reverse of the given array.	28.22	0.0	0.0	0.0
pro-mt-tf-b	reverses a list of numbers.	66.67	63.25	58.48	50.81
pro-mt-tf-l	reverses the order of the elements in the array.	20.22	0.0	0.0	0.0

más interesante. Igualmente el mejor resultado sigue siendo para el *TF* de tamaño *Large*. Estos valores podrían deberse a las características propias de los comentarios para este tipo de lenguajes. Como lenguaje declarativo, podría pensarse que los comentarios tienen cierta semi-estructuración donde se describen, muchas veces, las características de los parámetros que intervienen haciendo uso de los nombres de las variables y de los predicados (Tabla 3). Además, la longitud de los comentarios son relativamente más cortos. De igual manera, como se puede apreciar en dicho ejemplo, las métricas podrían no ser del todo adecuadas. La Tabla 4 muestra la salida de las dos tareas y de los modelos utilizando

distintos tamaños. También se observa su puntuación BLEU-n en relación a su referencia humana y su programa asociado.

6. Conclusiones y trabajos futuros

Se estudiaron diversas arquitecturas en la tarea de generación de comentarios a partir de código fuente. Estudiamos más a fondo la arquitectura de codificación-decodificación de CodeTrans. Se construyeron dos corpus, uno para el lenguaje GO y otro para el lenguaje PROLOG. La elección del primero radica en la selección de un lenguaje ya presente en la arquitectura antes mencionada. En cuanto al segundo, se consideró utilizar un lenguaje basado en otro paradigma de programación y que no estuviera presente. La experimentación relacionada al lenguaje GO fue satisfactoria para los modelos estudiados, obteniendo resultados significativos. En cuanto al lenguaje PROLOG, los resultados considerando la medida BLEU no fueron del todo satisfactorios. Como mencionamos en la Sección 5 hay muchos factores a considerar. Por ejemplo, las características de los comentarios, o bien la elección adecuada de la medida de evaluación para este tipo de lenguaje, entre otros. Sobre estos temas se seguirá trabajando.

Referencias

1. K. K. Aggarwal, Y. Singh, and J. K. Chhabra. An integrated measure of software maintainability. In *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No. 02CH37318)*, pages 235–241. IEEE, 2002.
2. A. Elnaggar, W. Ding, L. Jones, T. Gibbs, T. Feher, C. Angerer, S. Severini, F. Matthes, and B. Rost. Codetrans: Towards cracking the language of silicon's code through self-supervised deep learning and high performance computing. *arXiv preprint arXiv:2104.02443*, 2021.
3. Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.
4. M.-A. Lachaux, B. Roziere, L. Chanussot, and G. Lample. Unsupervised translation of programming languages. *arXiv preprint arXiv:2006.03511*, 2020.
5. C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.
6. C.-Y. Lin and F. J. Och. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 501–507, 2004.
7. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.
8. L. Tan, D. Yuan, G. Krishna, and Y. Zhou. icomment: Bugs or bad comments? In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 145–158, 2007.