García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Minería de repositorios para la recomendación de librerías Python con CrossRec*

Pedro Pablo García-Pozo, Aurora Ramírez y José Raúl Romero

Universidad de Córdoba, Córdoba, España
{i82gapop, aramirez, jrromero}@uco.es

Resumen La minería de repositorios software permite la extracción y análisis de datos provenientes de plataformas como GitHub con el objetivo de construir sistemas de recomendación para ingenieros software, entre otros. Uno de los campos de aplicación es la recomendación de librerías de terceros en base a las similitudes entre repositorios. En este trabajo abordamos este problema para repositorios de código Python. A pesar de la importancia actual de este lenguaje, aún no se han propuesto sistemas de recomendación específicos para sus librerías. En nuestro estudio experimental analizamos la influencia de varios factores a la hora de extraer los datos, como los criterios de filtrado de repositorios y la fuente de donde se extraen las librerías utilizadas por cada repositorio. A continuación, aplicamos el sistema de recomendación CrossRec sobre los conjuntos de datos para evaluar su rendimiento. Nuestros resultados confirman la necesidad de adaptar los sistemas de recomendación en función de la cantidad y diversidad de repositorios software que se observa en el ecosistema Python.

Keywords: Minería de repositorios software · librerías software · sistema de recomendación · Python · CrossRec

1. Introducción

El uso de librerías de terceros es una práctica muy extendida para acelerar la construcción de aplicaciones. Gracias a la disponibilidad de librerías, especialmente las de código abierto, los desarrolladores pueden encontrar funcionalidades ya probadas que se ajustan a sus necesidades o, al menos, requieren un esfuerzo de integración bajo en comparación con el coste que implicaría desarrollarlas desde cero [4]. El uso de librerías es particularmente relevante en el lenguaje de programación Python, que ha experimentado un notable incremento en su uso convirtiéndose por segundo año consecutivo en el lenguaje más importante según el índice TIOBE[1]. Ya en 2018 se observaba un crecimiento exponencial

* Proyecto PID2020-115832GB-I00 financiado por MICIN/AEI/10.13039/501100011033, Beca "Semillero de investigación" financiada por el plan propio de investigación 2022 de la Universidad de Córdoba, y contrato postdoctoral de la Junta de Andalucía (DOC_00944)

[1] https://www.tiobe.com/tiobe-index/ (Último acceso: 20/06/2023)

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

en el número de nuevos paquetes añadidos a PyPi [12], uno de los gestores de paquetes para Python y que actualmente cuenta con más de 450.000 paquetes[2].

Encontrar la librería adecuada dentro de un ecosistema como el de Python puede resultar una labor ardua para los desarrolladores, especialmente para aquellos con menos experiencia. Es por ello por lo que la recomendación automática de librerías se ha venido estudiando en los últimos años, aunque de momento solo para sistemas Java [11] o Android [3]. No sabemos cómo las técnicas de recomendación —algunas de ellas basadas en inteligencia artificial— podrían funcionar para la recomendación de librerías Python.

En general, los sistemas de recomendación en ingeniería del software se basan en dos pilares [8]: la extracción de datos para construir una base de conocimiento y el uso de técnicas de recomendación para, en base a esos datos, sugerir qué artefactos o acciones son interesantes dada una petición del usuario. En el caso de la recomendación de librerías, los datos son a menudo extraídos de una colección de repositorios de código abierto proveniente de plataformas como GitHub. En cuanto a técnicas, es habitual encontrar sistemas basados en técnicas de recuperación de la información (IR, *Information Retrieval*) o aprendizaje no supervisado (agrupamiento y minería de patrones). CrossRec es un ejemplo de sistema de recomendación de librerías Java basado en minería de repositorios software (MSR, *Mining Software Repositories*) e IR [6].

La calidad y representatividad de los datos extraídos de repositorios es sin duda uno de los factores principales que afecta al rendimiento de estos sistemas. No siempre es sencillo localizar y extraer la información relevante de plataformas como GitHub, donde cohabitan muchos repositorios de distinta naturaleza y grado de actividad [5]. En un trabajo anterior, presentamos un estudio que analizaba el ecosistema de librerías Python utilizando herramientas MSR y minería de patrones [2]. Además de descubrir los conjuntos de librerías más populares, este estudio nos permitió detectar la ausencia o incorrecta especificación de las dependencias para un alto porcentaje de repositorios. Es más, el hecho de que la gestión del fichero de dependencias sea responsabilidad de los contribuidores al repositorio no da garantías de que su contenido esté actualizado respecto a la última versión del código alojada. Abordar este tipo de situaciones es necesario para poder construir una base de conocimiento fiable con la que entrenar a cualquier sistema de recomendación de librerías para Python.

En este trabajo realizamos un estudio inicial sobre el uso de un sistema de recomendación de librerías Python a partir de repositorios GitHub. En primer lugar, presentamos un análisis actualizado del uso de librerías en repositorios, extendiendo notablemente el número de repositorios respecto a nuestro trabajo anterior. En dicho análisis, nos centramos en estudiar la coherencia entre la información presente en el repositorio y la que se infiere del código con el fin de corroborar cuál es más fiable para un sistema de recomendación. A continuación, hemos realizado un estudio experimental del rendimiento de CrossRec con más de 40.000 repositorios para analizar si el rendimiento de este sistema de recomendación, originalmente propuesto para el lenguaje Java, se ve afectado

[2] https://pypi.org/ (Último acceso: 20/06/2023)

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

por la naturaleza de los datos (cambio de lenguaje y tamaño de la muestra). Los resultados indican que la información sobre las dependencias de los repositorios no está actualizada respecto al código fuente en la gran mayoría de los casos. En cuanto al rendimiento de CrossRec, nuestros conjuntos de datos repercuten en la cantidad de librerías que son necesarias recomendar para incrementar las cotas de acierto y reducir la tasa de falsos positivos.

El resto del artículo se organiza como sigue. La Sección 2 presenta un ejemplo motivador. La Sección 3 resume el trabajo relacionado. La Sección 4 detalla la metodología para la extracción de datos y la experimentación con CrossRec. Los resultados experimentales se muestran en la Sección 5, seguida de una discusión en la Sección 6. La Sección 7 recoge las conclusiones y el trabajo futuro.

2. Ejemplo motivador

Para motivar la problemática de la extracción de datos, hemos realizado una prueba para comprobar cómo difiere el listado de librerías de un proyecto Python en función de la herramienta que se utilice para extraer sus dependencias[3]. El comando *pip freeze* permite listar los paquetes instalados en una localización, ya sea la general del usuario o un entorno virtual. Por otro lado, la herramienta *pipreqs*[4] analiza los comandos *import* del código para generar el listado de librerías en uso. La Figura 1 muestra el resultado de utilizar ambas opciones para un mismo proyecto Python cuyas dependencias se instalaron en un entorno virtual dedicado. Las librerías inicialmente instaladas fueron: *numpy*, *pandas*, *scikit-learn*, *lightgbm* y *dice-ml*. Cabe mencionar que alguna de estas librerías depende a su vez de otras previamente instaladas. Si no estuviesen ya instaladas, *pip* las instala automáticamente. Por ejemplo, *dice-ml* depende de *jsonschema*, *numpy*, *pandas*, *scikit-learn*, *tqdm* y *raiutils*[5].

Como puede verse en la Figura 1a, el comando *pip freeze* ha generado un listado con todas las librerías instaladas en el entorno virtual, incluyendo aquellas de "segundo nivel" instaladas por ser requeridas por las librerías explícitamente instaladas por el desarrollador. Por el contrario, la herramienta *pipreqs* ha generado un listado donde se incluyen las librerías realmente utilizadas en el código (Figura 1b). Así, vemos que el código está utilizando las cinco librerías originalmente instaladas y *tqdm*, una librería que fue instalada automáticamente pero que finalmente el desarrollador también importó. Por tanto, el fichero generado por *pipreqs* es más fiable para un sistema de recomendación pues se va a evitar la sugerencia de librerías de propósito general. Dichas librerías podrían aparecer en un alto número de repositorios, pero no necesariamente estar siendo utilizadas por todos ellos. La alta presencia de algunas librerías puede alterar

[3] La idea de esta prueba surge de la discusión sobre el funcionamiento del comando *pip freeze* tras la presentación de nuestro trabajo en las Jornadas SISTEDES 2022.

[4] https://pypi.org/project/pipreqs/ (Último acceso: 20/06/2023)

[5] https://github.com/interpretml/DiCE/blob/main/requirements.txt (Último acceso: 20/06/2023)

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

```
attrs==23.1.0
colorama==0.4.6
dice-ml==0.9
h5py==3.8.0
joblib==1.2.0
jsonschema==4.17.3
lightgbm==3.3.5
numpy==1.24.3
pandas==2.0.1
pyrsistent==0.19.3
python-dateutil==2.8.2
pytz==2023.3
scikit-learn==1.2.2
scipy==1.10.1
six==1.16.0
threadpoolctl==3.1.0
tqdm==4.65.0
tzdata==2023.3
```

(a) Dependencias según *pip freeze*

```
dice-ml==0.9
lightgbm==3.3.5
numpy==1.24.3
pandas==2.0.1
scikit-learn==1.2.2
tqdm==4.65.0
```

(b) Dependencias según *pipreqs*

Figura 1: Ficheros de dependencias generados por *pip* y *pipreqs*.

el funcionamiento de un sistema de recomendación basado en la similitud entre repositorios, incurriendo en el denominado "sesgo de popularidad" [7].

3. Trabajo relacionado

Esta sección resume el trabajo relacionado, dividido en base a las dos temáticas que confluyen en nuestro estudio: la minería de repositorios software (Sección 3.1) y los sistemas de recomendación de librerías (Sección 3.2).

3.1. Extracción de datos con minería de repositorios software

La minería de repositorios software permite extraer conocimiento a partir de artefactos software y metadatos alojados en plataformas colaborativas vinculadas al desarrollo de software [13]. A pesar del gran volumen de datos disponible y del creciente potencial de las herramientas y procedimientos MSR, es necesario considerar también ciertas limitaciones [5]. Poniendo como ejemplo GitHub, deben tenerse en cuenta aspectos como que no todos los repositorios alojan código, que algunos pueden estar inactivos o que sean copias (*forks*) de otros. Estos factores pueden afectar a la representatividad de los datos y la obtención de conclusiones. Actualmente, muchos de estos aspectos pueden controlarse mediante los criterios de filtrado que incorporan herramientas como *SEART GitHub Search* [1], y el seguimiento de recomendaciones a nivel metodológico [13].

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

Siguiendo esas recomendaciones, nuestro trabajo anterior nos permitió realizar un estudio sobre el uso de librerías Python en repositorios GitHub [2]. Para ello se extrajo un listado de más de 3.000 repositorios en base a varios criterios de filtrado como el número mínimo de *commits*, de estrellas y la fecha del último commit, entre otros. Con el fin de estudiar la relación entre librerías mediante minería de patrones, nuestro estudio partía de la premisa de que los repositorios contendrían un fichero de dependencias *requirements.txt* con las librerías utilizadas por cada repositorio seleccionado. Sin embargo, nuestro análisis reveló que la especificación de librerías era una práctica poco frecuente y, que incluso estando presente, el fichero no siempre tenía un formato válido o aparecía duplicado. Por otro lado, también hubo que descartar repositorios con una única librería en su fichero de dependencias, por no ser relevante para extraer tendencias con las que construir sistemas de recomendación. Este hecho redujo la muestra de repositorios al 24 % de los inicialmente encontrados, si bien permitió detectar conjuntos de librerías utilizadas frecuentemente de forma conjunta.

3.2. Sistemas de recomendación de librerías

Analizar la aparición conjunta de librerías entre repositorios similares es la base de los sistemas de recomendación de librerías que se han propuesto en la literatura. LibRec, el primero en aparecer, aplica minería de patrones y filtrado colaborativo para analizar la similitud entre repositorios [11]. Para su evaluación se utilizaron datos extraídos de 500 repositorios Java. Otro sistema es LibCUP [10], que combina el filtrado colaborativo con una técnica propia de agrupamiento inspirada en el algoritmo DBSCAN. Este sistema fue evaluado experimentalmente con una base de conocimiento de 90.475 repositorios Java y 56.435 librerías.

CrossRec es el sistema de recomendación de librerías Java más reciente [6]. A partir de una muestra de proyectos, se extraen sus dependencias Maven para construir un grafo con la relación entre los proyectos y las librerías que estos incluyen. Basándose en la matriz de adyacencia de dicho grafo, CrossRec aplica una técnica de filtrado colaborativo basada en similitud entre proyectos. Dado un proyecto P para el cual se dispone de un conjunto de librerías en uso, CrossRec realiza dos pasos para recomendar nuevas librerías que podrían ser útiles en dicho proyecto: 1) Se seleccionan los n proyectos más similares a P y 2) se estima la relevancia de las librerías no incluidas en P, pero sí utilizadas por los n proyectos similares. Como resultado, CrossRec devuelve una lista de l librerías recomendadas por orden de relevancia. CrossRec fue evaluado frente a LibRec y LibCUP utilizando sus mismos conjuntos de datos, así como uno propio construido a partir de 1.200 repositorios. Como parte del estudio experimental, los autores también analizaron la influencia de los parámetros n (5, 10, 15, 20, 25) y l (1, 3, 5, 7 y 10). La herramienta está disponible en GitHub[6].

El campo de los sistemas de recomendación de librerías sigue activo, con trabajos recientes que buscan incorporar más información al proceso para mejorar

[6] https://github.com/crossminer/CrossRec (Último acceso: 20/06/2023)

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

el rendimiento. Por un lado, se ha estudiado cómo añadir la retroalimentación del desarrollador para mejorar la precisión en función de si este considera que las recomendaciones de CrossRec son adecuadas o no [9]. Por otro lado, se ha avanzado en la detección y reducción del sesgo de popularidad reajustando las recomendaciones iniciales de sistemas como CrossRec y LibRec [7].

4. Marco experimental

En este trabajo ha sido necesario considerar tanto una metodología para la extracción y análisis de los datos provenientes de GitHub, como una metodología experimental para la evaluación de CrossRec. Para lo primero se han seguido las pautas de elaboración de estudios MSR propuesta por Vidoni [13]. Primero se ha llevado a cabo una fase de *planificación*, en la que se han formulado las preguntas de investigación que guiarán el estudio (Sección 4.1). A partir de ellas se continúa con la fase de *ejecución*, encaminada a la construcción del conjunto de datos a partir de la información extraída de GitHub (Sección 4.2). Finalmente, es necesaria una fase de *análisis* que permita evaluar la calidad de la información obtenida (Sección 4.3). Tras ello, estaremos en disposición de utilizar el conjunto de datos en CrossRec, cuyo entrenamiento y evaluación se detallan en la Sección 4.4.

4.1. Preguntas de investigación

A continuación se formulan las preguntas de investigación y se aborda cómo se llevará a cabo el estudio experimental para darles respuesta.

RQ1: ¿Cuál es el grado de coherencia acerca del uso de librerías entre el código de los repositorios y su fichero de dependencias (*requirements.txt*)?

Para dar respuesta a la RQ1, hemos reproducido el proceso de selección de repositorios de código en lenguaje Python alojados en GitHub que aplicamos en nuestro trabajo anterior [2]. En esta ocasión, además de extraer las librerías del fichero de dependencias, se han identificado las librerías realmente importadas en dicho código. Comparando ambos listados, podremos comprobar si la especificación del fichero coincide con el uso real de las librerías en el código o, por el contrario, el fichero está desactualizado o incluye dependencias de segundo nivel.

RQ2: ¿Cómo afecta la cantidad de repositorios utilizados para entrenar al sistema de recomendación a su rendimiento?

Para dar respuesta a la RQ2, hemos construido diferentes conjuntos de datos variando los filtros de selección en base a tres criterios: número de *commits*, número de estrellas y número de contribuidores. Con ello se pretende observar cómo varía la muestra de repositorios devuelta por la herramienta de búsqueda *SEART GitHub Search*. Cabe esperar que se observen diferencias entre las muestras no solo en el total de repositorios, sino también en la cantidad y diversidad de librerías que estos utilizan. A su vez, es necesario estudiar si tales diferencias repercuten o no en la precisión del sistema de recomendación.

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

4.2. Construcción del conjunto de datos

La herramienta *SEART GitHub Search* permite la localización de repositorios alojados en GitHub mediante una gran variedad de filtros de búsqueda. Hemos definido dos filtros estáticos para fijar el lenguaje de programación (Python) y la fecha del *commit* más reciente (posterior al 1 de enero de 2021). Este segundo filtro es recomendable para garantizar que se analizan repositorios activos [13]. Para los filtros dinámicos planteados en la RQ2, se han utilizado los siguientes valores: 1) Mínimo número de *commits* entre 100 y 1000, con un salto de 50; 2) Mínimo número de estrellas entre 10 y 100, con un salto de 5; y 3) Mínimo número de contribuidores entre 1 y 10, con salto de 2. Combinando los valores de los tres filtros, se obtienen un total de 1.805 combinaciones. Cabe señalar que los umbrales mínimos de *commits* y estrellas se han establecido a esos valores para evitar la inclusión de repositorios poco relevantes [13]. De hecho, la propia herramienta *SEART GitHub Search* establece el mínimo número de estrellas en 10. Con los criterios establecidos, se han obtenido repositorios con un máximo de 210.736 estrellas, 4.314.502 *commits* y hasta 755 contribuidores.

Tras la selección de los repositorios, se ha procedido a descargar su contenido para extraer la información necesaria para nuestro estudio. Para la RQ1 es necesario que los repositorios contengan el fichero de dependencias *requirements.txt*. Como se analizó en nuestro trabajo anterior [2], no todos los repositorios contienen dicho fichero, algunos incluyen varios, y no siempre el formato es correcto como para permitir su tratamiento automático con la herramienta *Requirements Parser*[7]. No obstante, en este nuevo estudio se ha incrementado notablemente la muestra inicial de repositorios (de 3.347 a 45.155), lo que aporta más garantías de que el número final de repositorios tras descartar los que no pueden procesarse automáticamente es adecuado para el objetivo de nuestro estudio.

Con independencia de la presencia o no del fichero de dependencias, se ha procedido a extraer el listado de librerías importadas en los ficheros de código fuente Python incluidos en cada repositorio. Para ello se ha utilizado la herramienta *pipreqs*. Aquellos repositorios en los que el fichero está presente servirán para el análisis de la RQ1. Para la RQ2 se utilizará directamente la información de librerías extraídas del código para poder utilizar todos los repositorios.

4.3. Análisis del conjunto de datos

La Figura 2 muestra el número de repositorios obtenido para cada combinación de criterios de filtrado (estrellas, *commits* y contribuidores). Se puede observar que el número de estrellas y el número de contribuidores afectan más a la obtención de repositorios que el número de *commits*. El filtro más permisivo (10 estrellas, 100 *commits* y 1 contribuidor) retorna más de 45.000 repositorios, mientras que el más restrictivo (100 estrellas, 1.000 *commits* y 10 contribuidores) se cumple para 3.373 repositorios. Esta diferencia remarca la importancia de valorar distintos umbrales, puesto que la muestra de repositorios es sin duda

[7] https://requirements-parser.readthedocs.io/ (Último acceso: 20/06/2023)

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

un factor que afecta a la obtención de conclusiones más o menos generalizables en estudios MSR.

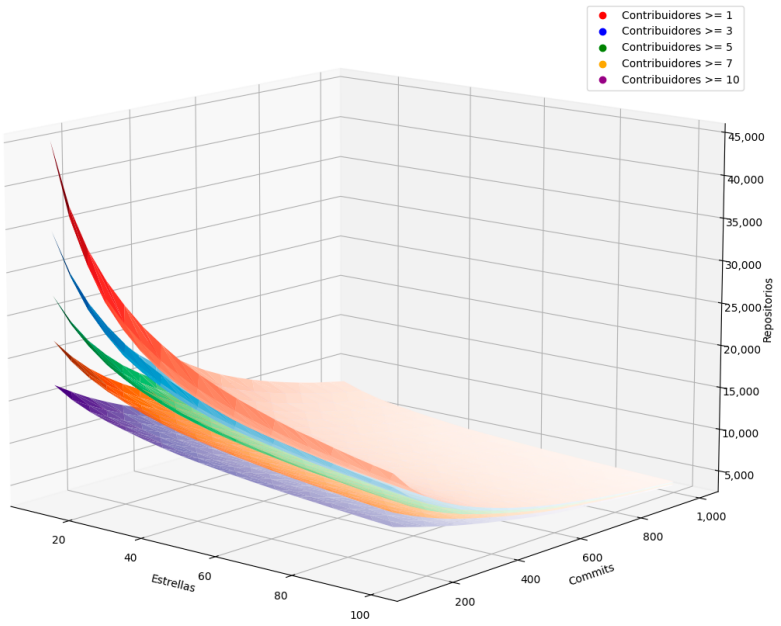


Figura 2: Número de repositorios obtenidos con diferentes filtros.

4.4. Entrenamiento y evaluación de CrossRec

Para el estudio experimental de CrossRec, se han seleccionado tres conjuntos de datos con diferente número de repositorios y, en consecuencia, de librerías:

- *Dataset 1.* Contiene 2.757 repositorios y 15.591 librerías, resultante del siguiente filtrado: estrellas=100, *commits*=1.000, contribuidores=10.
- *Dataset 2.* Contiene 13.597 repositorios y 27.268 librerías, resultante del siguiente filtrado: estrellas=50, *commits*=100, contribuidores=5.
- *Dataset 3.* Contiene 36.968 repositorios y 45.005 librerías, resultante del siguiente filtrado: estrellas=10, *commits*=100, contribuidores=1.

Siguiendo el diseño experimental de los autores de CrossRec, se ha aplicado una estrategia de validación cruzada (*k-folds*) con $k = 10$ [6]. Es decir, cada conjunto de datos se ha dividido en 10 partes, de forma que 9 de ellos se utilizarán para conformar la base de conocimiento del sistema de recomendación (lo que denominaremos "partición de entrenamiento") y la restante para evaluar

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

el rendimiento del sistema ("partición de prueba"). Para cada repositorio en la partición de prueba, se eliminan aleatoriamente la mitad de sus librerías, las cuales actuarán como *"ground-truth"* para la evaluación. A partir de las librerías restantes, CrossRec determinará un listado de librerías recomendadas en base a los 10 repositorios más similares de la partición de entrenamiento. Se considera que CrossRec acierta en su predicción si una librería previamente eliminada (*"ground-truth"*) aparece en el listado de librerías recomendadas. El proceso de recomendación se repite k veces, alternando la partición de prueba, con el fin de obtener resultados más generalizables.

Una vez obtenida la lista de librerías recomendadas, se puede evaluar la calidad de la recomendación en base a varias medidas habituales de los sistemas de recuperación de información. Los autores de CrossRec también han propuesto medidas específicas [6]. En total, las medidas consideradas son:

- *SuccessRate*. Para el conjunto de repositorios de la partición de prueba, devuelve el número de veces que la lista de librerías recomendadas incluye al menos una librería del conjunto *ground-truth*. Su valor máximo es 1.
- *Recall*. Es el cociente entre las librerías del conjunto *ground-truth* que aparecen en la lista recomendada y el total de librerías en el conjunto *ground-truth*. Su valor máximo es 1.
- *Precision*. Es el cociente entre el número de librerías del conjunto *ground-truth* que aparecen en la lista recomendada y el total de librerías recomendadas. Su valor máximo es 1.
- *Entropy*. Es una medida de dispersión en base al total de librerías que podrían recomendarse, de forma que evalúa si las recomendaciones se concentran en un conjunto pequeño de ellas. Un menor valor de entropía indica una mayor variedad de librerías recomendadas en el total de repositorios probados.
- *Coverage*. Es el cociente entre el número de librerías que son recomendadas para todos los repositorios de la partición de prueba y el total de librerías que podrían recomendarse. Su valor máximo es 1.

Cada medida se calcula para distintos valores de l, de forma que se pueda estudiar la influencia del número de librerías recomendadas. Originalmente en la herramienta se calculan para l={3, 5, 7, 10, 15}. Hemos considerado valores mayores dado que el número de librerías máximo que conforman un conjunto *ground-truth* en nuestros conjuntos de datos es de 498 librerías. En concreto, se reportan los valores de todas las medidas para l={5, 10, 25, 50, 100}.

5. Resultados experimentales

A continuación, se presentan los resultados experimentales y su análisis para la RQ1 (Sección 5.1) y la RQ2 (Sección 5.2).

5.1. RQ1: Coherencia del fichero de dependencias

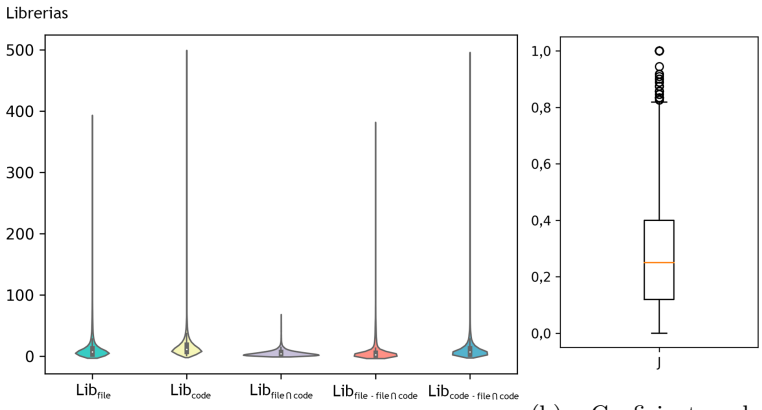
Partiendo del conjunto de datos más grande (estrellas=10, *commits*=100, contribuidores=1), que consta de 45.135 repositorios, se han aislado los 18.414

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

repositorios que contenían un único fichero *requirements.txt* válido y, por tanto, procesable. Por otro lado, se ha ejecutado *pipreqs* para obtener el listado de librerías a partir del código. La Figura 3 muestra la coherencia que existe entre el fichero *requirements.txt* original y el generado a partir del código. En la Figura 3a, Lib_{file} es el conjunto de librerías en el fichero *requirements.txt*, Lib_{code} es el conjunto de librerías en el listado generado por *pipreqs*, y $Lib_{file \cap code}$ es la intersección de ambos conjuntos. Por último, $Lib_{file-file \cap code}$ y $Lib_{code-file \cap code}$ son los dos conjuntos disjuntos resultantes. Además, se ha calculado el coeficiente de similitud de Jaccard para medir el grado de similitud entre los dos conjuntos:

$$J(Lib_{file}, Lib_{code}) = \frac{|Lib_{file} \cap Lib_{code}|}{|Lib_{file} \cup Lib_{code}|} \quad (1)$$



(a) Distribución del número de librerías.

(b) Coeficiente de similitud de Jaccard.

Figura 3: Diferencias entre los conjuntos de librerías en el fichero *requirements.txt* y el generado por *pipreqs*.

Un primer hecho que llama la atención es que el número de librerías importadas en el código (Lib_{code}) tiende a ser superior al número de librerías en el fichero *requirements.txt*. En concreto, la mediana es 11 frente a 7, y la diferencia puede llegar a ser de hasta 372 librerías en el caso del repositorio *tomerwolgithub_question-decomposition-to-sql*. Observando la distribución de $Lib_{file \cap code}$, la mediana tiene un valor igual a 4, lo cual nos indica una muy baja consistencia entre lo que indican ambos ficheros. En cuanto a las librerías en las que no coinciden ambos ficheros ($Lib_{file-file \cap code}$ y $Lib_{code-file \cap code}$), la mediana es de 2 y 7 respectivamente. El primer dato nos sugiere que se están dando casos donde el fichero contiene librerías de más, esto es, que no son utilizadas en el código. Puede deberse a que se dejaron de utilizar en algún momento y no se actualizó el fichero de dependencias en consecuencia, o que este fue generado con *pip freeze*.

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

En cuanto al segundo dato nos encontramos con que, en mayor medida incluso, hay librerías en el código que no se especifican como dependencias en el fichero. En este caso, lo más probable es que se deba a la falta de actualización del fichero conforme se registran nuevas versiones del código. Esto provocaría errores a la hora de instalar y ejecutar el código del repositorio, obligando al desarrollador a localizar las dependencias restantes en base a los *import*.

Respecto al grado de coincidencia entre los ficheros, la Figura 3b muestra la distribución del coeficiente de similitud de Jaccard para la muestra de repositorios analizados. En media, este coeficiente es igual a 0.27, lo que indica que menos del 30 % de las librerías que aparecen en ambos listados son realmente dependencias del código (librerías importadas) que están correctamente especificadas en el fichero *requirements.txt* alojado en el repositorio. La parte superior del boxplot deja patente que solo unos pocos repositorios (identificados como *outliers*) presentan un grado de coincidencia superior al 80 %. Independiente de la causa por la cual los listados no coinciden, nuestro estudio deja patente la necesidad de extraer la información sobre las librerías directamente del código fuente en su última versión. De lo contrario, el sistema de recomendación podría incurrir en recomendaciones poco útiles.

Respuesta a RQ1. Existe un grado bajo de coherencia (27 % en media) entre la especificación de librerías supuestamente requeridas por un repositorio y el conjunto de librerías utilizadas realmente en el código.

5.2. RQ2: Rendimiento de CrossRec

La Tabla 1 muestra los resultados de las cinco medidas de rendimiento para los tres conjuntos de datos construidos. Para cada uno se incluyen los resultados en función de los cinco valores del parámetro l (número de librerías recomendadas). Cada celda representa la media y desviación típica de los 10 *folds*. En términos de *SuccessRate*, las recomendaciones son más acertadas conforme aumenta l, pues hay más librerías entre las que se pueden encontrar las pertenecientes al *ground-truth* del repositorio de prueba. Es interesante resaltar que la mejora es mayor en los conjuntos de datos más pequeños. Posiblemente sea necesario aumentar el valor de l conforme más librerías diferentes existen en la partición de entrenamiento y mayor diversidad de repositorios haya, pues es más difícil localizar las realmente relevantes.

Respecto a *recall* y *precision*, vemos que en general se obtienen mejores valores de la primera que de la segunda. Esto se traduce en que CrossRec es más sensible al número de falsos positivos, los cuales suelen crecer conforme el tamaño de la lista de recomendaciones aumenta. En comparación con los resultados reportados por los autores de CrossRec [6], los resultados aquí obtenidos son inferiores en ambas medidas, lo cual asociamos a las características de nuestros conjuntos de datos. Nuestro conjunto de datos más pequeño tiene el triple de repositorios que el conjunto más grande con el que se evaluó CrossRec. También sería necesario analizar cómo difieren en cuanto a la distribución de librerías por repositorios, lo cual a su vez afecta al tamaño de los conjuntos *ground-truth*. Tal

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Tabla 1: Rendimiento de CrossRec para los tres conjuntos de datos.

dataset	**l**	**SucessRate**	**Recall**	**Precision**	**Coverage**	**Entropy**
dataset1	5	0,281 ± 0,010	0,042 ± 0,002	0,112 ± 0,006	0,036 ± 0,018	13,954 ± 0,316
dataset1	10	0,378 ± 0,002	0,080 ± 0,001	0,113 ± 0,003	0,078 ± 0,040	16,352 ± 0,424
dataset1	25	0,495 ± 0,003	0,154 ± 0,002	0,096 ± 0,003	0,210 ± 0,114	19,335 ± 0,368
dataset1	50	0,604 ± 0,008	0,195 ± 0,002	0,069 ± 0,002	0,380 ± 0,210	22,692 ± 0,339
dataset1	100	0,678 ± 0,006	0,232 ± 0,003	0,049 ± 0,001	0,584 ± 0,321	26,662 ± 0,498
dataset2	5	0,266 ± 0,008	0,065 ± 0,004	0,095 ± 0,003	0,056 ± 0,027	20,344 ± 0,223
dataset2	10	0,337 ± 0,008	0,101 ± 0,004	0,091 ± 0,001	0,121 ± 0,059	23,744 ± 0,222
dataset2	25	0,411 ± 0,005	0,151 ± 0,005	0,078 ± 0,001	0,255 ± 0,126	27,741 ± 0,302
dataset2	50	0,439 ± 0,005	0,168 ± 0,005	0,066 ± 0,000	0,350 ± 0,174	30,230 ± 0,425
dataset2	100	0,445 ± 0,005	0,175 ± 0,006	0,061 ± 0,000	0,415 ± 0,208	31,271 ± 0,515
dataset3	5	0,222 ± 0,001	0,055 ± 0,000	0,086 ± 0,001	0,052 ± 0,025	25,719 ± 0,170
dataset3	10	0,282 ± 0,002	0,085 ± 0,001	0,084 ± 0,000	0,106 ± 0,051	31,185 ± 0,205
dataset3	25	0,333 ± 0,001	0,122 ± 0,000	0,075 ± 0,000	0,207 ± 0,097	35,172 ± 0,330
dataset3	50	0,343 ± 0,001	0,131 ± 0,000	0,067 ± 0,000	0,265 ± 0,123	36,327 ± 0,443
dataset3	100	0,344 ± 0,001	0,134 ± 0,000	0,065 ± 0,000	0,293 ± 0,135	36,688 ± 0,467

y como los autores estudiaron, dicho tamaño afecta notablemente al rendimiento de CrossRec, haciendo que la medida *recall* oscile entre 0,05 y 0,35, y la medida *precision* entre valores inferiores a 0,1 y 0,6.

Finalmente, la entropía crece conforme más librerías se recomiendan, al igual que lo reportado en el artículo original. No obstante, los valores absolutos son aquí notablemente mayores, pues en el estudio original oscilan entre 0,1 y 1,2. Alineado con esto, el *coverage* rara vez supera el valor 0,5, esto es, el conjunto de recomendaciones se compone de menos del 50 % de las librerías presentes en los repositorios de la partición de prueba. Esto nos indica que nuestro conjunto de datos tiene mucha más diversidad de repositorios y librerías, lo cual era un aspecto que intuíamos por el cambio de lenguaje de programación.

Respuesta a RQ2. El rendimiento de CrossRec se ve afectado por el número de repositorios que componen la partición de entrenamiento y el número de librerías a recomendar. El cambio de lenguaje de programación supone un cambio notable en el funcionamiento de CrossRec.

6. Discusión

En relación al análisis para RQ1, la discrepancia observada abre nuevos interrogantes sobre las causas que la provocan. Sería interesante inspeccionar el historial de *commits* para ver la evolución que sufre el fichero de dependencias. En general, también podría estudiarse si las discrepancias observadas son más o menos severas en función de características del repositorio como su antigüedad, frecuencia de actualización o número de contribuidores.

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

En cuanto a los resultados para RQ2, el tamaño y la distribución de librerías en los conjuntos de datos utilizados es claramente diferente a los estudiados hasta la fecha en la literatura. El cambio de lenguaje, de Java a Python, ha revelado diferencias en el funcionamiento de CrossRec. El más evidente es la necesidad de generar listas de recomendación más extensas (mayores valores del parámetro l) para alcanzar valores similares de *precision* y *recall* a los obtenidos en el estudio original. Dada la diversidad de repositorios en nuestro estudio, sería interesante estudiar el impacto del porcentaje de dependencias que componen el *ground-truth*, así como el número de proyectos similares (n) utilizado por CrossRec, que podrían necesitar ser adaptados. En el estudio inicial, los autores concluyeron que este último valor no era muy influyente, recomendando $n = 10$. El tiempo de ejecución también se ve afectado por el tamaño del conjunto de datos, pasando de unos pocos minutos para el *Dataset 1* a casi 6h para el *Dataset 3*.

Los conjuntos de datos para recomendación de librerías no suelen tener una distribución uniforme, sino que sufren el efecto *long-tail*. Esto es, existen unas pocas librerías muy populares (en el sentido de que son utilizadas por muchos repositorios) y un alto número de librerías presentes en muy pocos repositorios. Consideramos que este efecto puede ser incluso muy influyente en Python, pues en alguno de nuestros conjuntos de datos nos encontramos con que más de la mitad de las librerías aparecen solo en dos repositorios. Será por tanto necesario analizar la influencia del sesgo de popularidad en CrossRec u otros sistemas de recomendación para nuestros conjuntos de datos, así como otras métricas de evaluación más específicas. Fijar umbrales máximos de los filtros podría ser interesante para controlar el sesgo de popularidad. Otra alternativa que consideramos interesante es la de incluir meta-información sobre las librerías, como su frecuencia de actualización, número de contribuidores o estrellas, para ofrecer recomendaciones más completas y que no se basen únicamente en la popularidad.

7. Conclusiones

En este trabajo hemos abordado el problema de la recomendación de librerías Python aplicando técnicas de minería de repositorios software sobre GitHub. Se ha analizado la influencia de los criterios de filtrado a la hora de localizar repositorios. Además, se han comparado dos formas distintas de extraer las librerías utilizadas por cada repositorio, detectando una alta discrepancia en la información encontrada en cada caso. Todo ello ha permitido generar un conjunto de datos más completo y fiable con el que entrenar al sistema de recomendación CrossRec. El hecho de que este sistema fue evaluado con repositorios Java nos ha permitido a su vez estudiar la influencia del cambio de lenguaje y de las características del conjunto de datos en su rendimiento.

La investigación en la recomendación automática de librerías sigue vigente y debe ser explorada en más detalle para lenguajes como Python por su actual relevancia. Entre los temas discutidos se abren varias líneas de trabajo futuro: el análisis temporal de las dependencias en los repositorios; la mejora o propuesta

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

de las técnicas de recomendación; y la inclusión de información adicional sobre las librerías para evitar sesgos hacia las más populares.

Referencias

1. Dabic, O., Aghajani, E., Bavota, G.: Sampling Projects in GitHub for MSR Studies. In: 18th Int. Conf. Mining Software Repositories. pp. 560–564 (2021). https://doi.org/10.1109/MSR52588.2021.00074
2. García-Pozo, P.P., Ramírez, A.: Análisis de librerías Python en repositorios GitHub con minería de patrones. In: Actas de las XXVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD) (2022), hdl.handle.net/11705/JISBD/2022/861
3. He, Q., Li, B., Chen, F., Grundy, J., Xia, X., Yang, Y.: Diversified Third-Party Library Prediction for Mobile App Development. IEEE Transactions on Software Engineering **48**(1), 150–165 (2022). https://doi.org/10.1109/TSE.2020.2982154
4. Huang, K., Chen, B., Xu, C., Wang, Y., Shi, B., Peng, X., Wu, Y., Liu, Y.: Characterizing usages, updates and risks of third-party libraries in java projects. Empirical Software Engineering **27**(4) (2022). https://doi.org/10.1007/s10664-022-10131-8
5. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The Promises and Perils of Mining GitHub. In: 11th Working Conference on Mining Software Repositories (MSR). p. 92–101 (2014). https://doi.org/10.1145/2597073.2597074
6. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Di Penta, M.: Crossrec: Supporting software developers by recommending third-party libraries. Journal of Systems and Software **161**, 110460 (2020). https://doi.org/10.1016/j.jss.2019.110460
7. Nguyen, P.T., Rubei, R., Rocco, J.D., Sipio, C.D., Ruscio, D.D., Penta, M.D.: Dealing with Popularity Bias in Recommender Systems for Third-party Libraries: How far Are We? In: 20th Int. Conf. Mining Software Repositories (2023)
8. Rocco, J.D., Ruscio, D.D., Sipio, C.D., Nguyen, P.T., Rubei, R.: Development of recommendation systems for software engineering: the CROSSMINER experience. Empirical Software Engineering **26**(29) (2021). https://doi.org/10.1007/s10664-021-09963-7
9. Rubei, R., Di Sipio, C., Di Rocco, J., Di Ruscio, D., Nguyen, P.T.: Endowing third-party libraries recommender systems with explicit user feedback mechanisms. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 817–821 (2022). https://doi.org/10.1109/SANER53432.2022.00099
10. Saied, M.A., Ouni, A., Sahraoui, H., Kula, R.G., Inoue, K., Lo, D.: Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software **145**, 164–179 (2018). https://doi.org/10.1016/j.jss.2018.08.032
11. Thung, F., Lo, D., Lawall, J.: Automated library recommendation. In: 20th Working Conf. Reverse Engineering. pp. 182–191 (2013). https://doi.org/10.1109/WCRE.2013.6671293
12. Valiev, M., Vasilescu, B., Herbsleb, J.: Ecosystem-Level Determinants of Sustained Activity in Open-Source Projects: A Case Study of the PyPI Ecosystem. In: 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 644–655 (2018). https://doi.org/10.1145/3236024.3236062
13. Vidoni, M.: A systematic process for mining software repositories: Results from a systematic literature review. Information and Software Technology **144**, 106791 (2022). https://doi.org/10.1016/j.infsof.2021.106791

García-Pozo, P. P., Ramírez, A., Romero, J. R.: Minería de repositorios para la recomendación de librerías Python con CrossRec. In: Durán Toro, A. (ed.) Actas de las XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023). https://hdl.handle.net/11705/JISBD/2023/135

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.