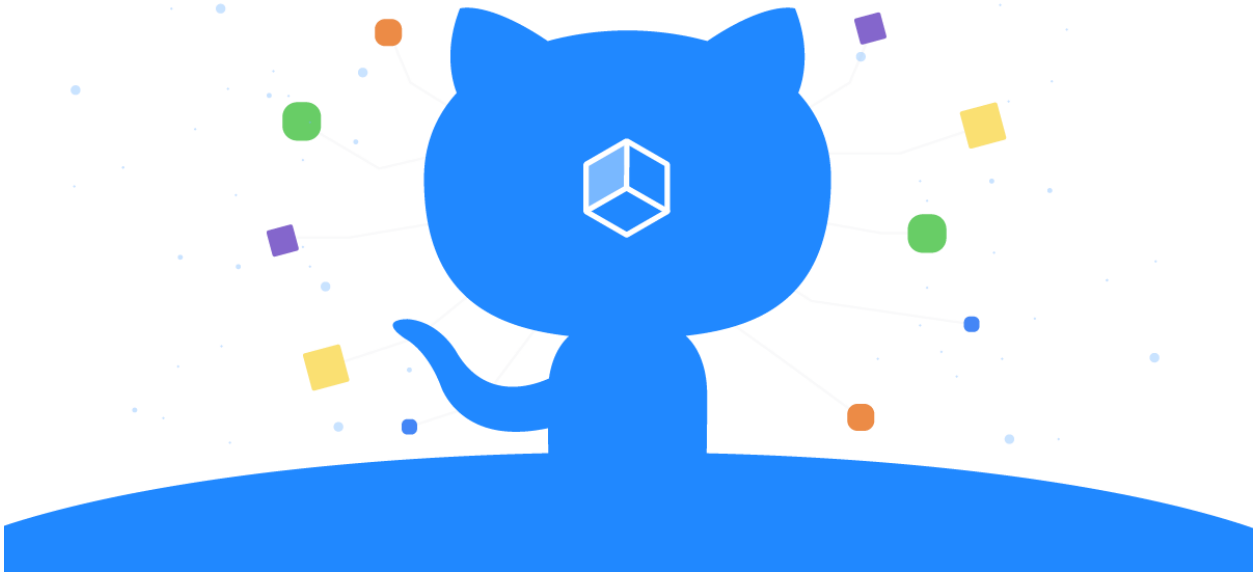


GitHub RepoReco

21/09/2019

Eduardo Nogueira 4.604.294-5
Daniela Andrade 4.966.382-1
Matías Manrique 4.994.465-9
Carolina Reolon 4.720.173-4

Grupo 09
Recuperación de Información y Recomendaciones en la Web

Introducción

Es común en lo sistemas de información actuales encontrar funcionalidades de recomendaciones sobre las entidades u objetos que el mismo maneja. Por ejemplo una red social que le recomienda a un usuario posibles contactos, o un sistema e-commerce que le recomienda a los compradores productos que le pueden interesar. Esto se logra utilizando diferentes algoritmos y técnicas que ordenan los objetos de estudio de acuerdo a distintos criterios basándose en la información disponible de cada usuario, intentando presentarle los objetos que les resulten de mayor interés.

Estos sistemas resultan de gran utilidad para los usuarios, ya que ayudan a evaluar y filtrar la enorme cantidad de información disponible en la Web.

En este trabajo se toma GitHub como caso de estudio, donde se implementará un sistema que recomiende repositorios a los usuarios. Se analizará las facilidades del API de GitHub para obtener la información necesaria para realizar las recomendaciones, intentando buscar soluciones aceptables bajo las restricciones que dicha API tiene.

Objetivos

1. Adquirir conocimiento de recolección de datos públicos de la web
2. Adquirir conocimientos de algoritmos de recomendaciones
3. Crear un proyecto que agregue valor a los usuarios de GitHub

Problema

Debido a lo útil que son los repositorios para el intercambio de información para los diferentes contextos en los que uno trabaja y considerando que GitHub no cuenta con ningún buscador de proyectos aplicando filtros interesante para el usuario, sería valioso que cada usuario pudiera recibir recomendaciones de diferentes repositorios que en cierta forma están vinculados con los contextos en que éste se maneja.

Es por esto que surgió la necesidad de implementar alguna solución que permita lo antes descripto.

Enfoque de la solución

Nos basamos en la API pública de GitHub donde se puede recolectar datos sobre los usuarios y sus espacios de trabajo. Estos datos pueden luego ser utilizados para hallar relaciones que permitan, mediante un algoritmo adecuado, recomendar repositorios que pueden ser de su interés.

En principio se nota que es necesario utilizar el endpoint de la API de Usuarios para obtener información de estos, y el endpoint de la API de Repositorios para obtener los usuarios que han participado en estos.

Análisis de la API

En primer lugar necesitamos analizar los datos que podemos obtener con el API pública de GitHub para poder armar relaciones entre usuarios y repositorios. Consideraremos estas relaciones con dirección, debido a que en general el API se centra en obtener los datos de una entidad (por ejemplo un usuario) y las relaciones de la misma con otras entidades (por ejemplo los repositorios de este usuario).

Se presenta a continuación la relación, y las formas de obtener sus instancias.

Usuario -> Usuario

- **/users/:user/followers**

  Devuelve los seguidores de un usuario.

- **/users/:user/following**

  Devuelve los usuarios seguidos por el usuario actual.

Usuario -> Repositorio

- **/users/:user/starred**

  Un usuario puede marcar con una estrella un repositorio de su interés. Este endpoint nos permite obtener los repositorios marcados por un usuario.

- **/users/:user/repos**

  Lista los repositorios de un usuario.

Repositorio -> Usuario

- **/repos/:owner/:repo/contributors**

Devuelve los usuarios que han contribuído en un repositorio mediante commits.

Sistemas de recomendación

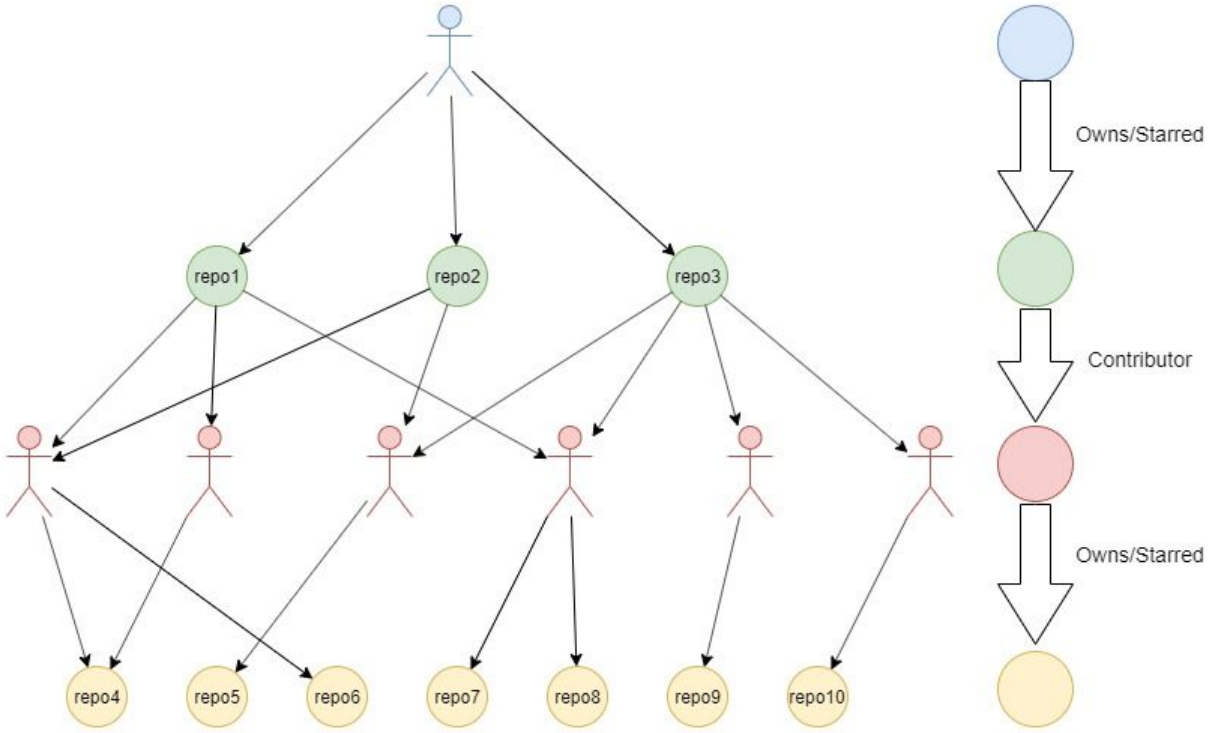
En general existen dos formas en la que un sistema de recomendaciones puede funcionar. Por un lado se puede basar en las características de los elementos con los que el usuario se relaciona (en este caso repositorios), de modo que se le recomiendan elementos con características similares a los que ya tiene relacionados. Por otro lado, se puede basar en usuarios con gustos similares, en lo que se conoce como sistema de filtros colaborativos.

La API tiene además un límite de 5000 consultas por hora por usuario autenticado. Esto hace que sea difícil obtener y actualizar el universo de usuarios y repositorios. Revisando algunos números de forma apresurada, tenemos que en cada consulta al api de Usuarios podemos obtener 30 usuarios, dado que en GitHub hay más de 31 millones de usuarios para obtener todos los usuarios de github necesitamos más de 1.030.000 consultas. Por otro lado podemos consultar cuales son los repositorios de cada usuario en una consulta por usuario, lo que nos da al menos 31 millones de consultas más. 32 millones de consultas realizadas a 5000 por hora nos da que para obtener solo los usuarios y sus repositorios necesitamos más de 6400 horas. En el mejor de los casos se puede paralelizar si tenemos más de un usuario de GitHub para autenticarnos. Nuestro grupo está formado por 4 personas, lo que nos deja en más de 1600 horas, es decir unos 66 días solo para obtener la lista completa de usuarios y repositorios. Evidentemente esta opción no es viable para el plazo que estamos manejando, y hace imposible la obtención de datos en tiempo real, en caso de que esto sea necesario.

Necesitamos encontrar una forma de realizar recomendaciones de modo que podamos obtener los datos directamente del API de GitHub, y sin necesidad de disponer de todos los datos de Usuarios y Repositorios. En principio esto nos lleva también a descartar la opción de realizar recomendaciones basadas en los atributos de los repositorios, ya que no podemos obtener la información total de los mismos.

En el caso de los filtros colaborativos, es posible obtener relaciones Usuario -> Repositorio, lo que nos da una lista de intereses del usuario actual, y Repositorio -> Usuario, lo que nos permite conocer usuarios con intereses similares.

Por lo que una posible aproximación para realizar las recomendaciones de repositorios puede ejemplificarse en la siguiente imagen.



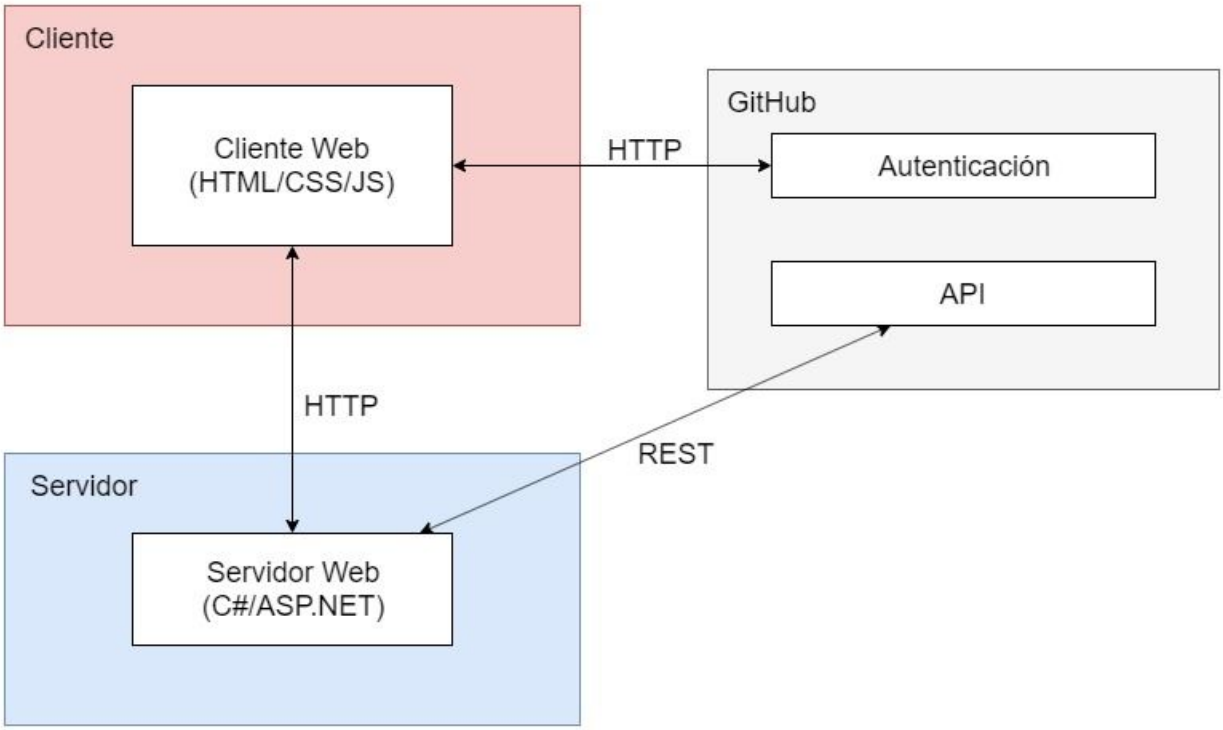
Donde, dado un usuario y los repositorios que le pertenecen o que marcó como “starred”, se puede obtener los usuarios que contribuyeron en los mismo. Luego se obtienen los repositorios pertenecientes o starred de dichos usuarios. De modo que al usuario podemos recomendarle los repositorios 4 al 10. El orden de relevancia que tomaremos es la cantidad de caminos que encontramos desde el usuario que consulta y los repositorios recomendados. Por ejemplo en el caso anterior el orden sería el siguiente.

Repositorio	Cantidad de caminos
repo4	3
repo5	2
repo6	2
repo7	2
repo8	2
repo9	1
repo10	1

El criterio que utilizamos en nuestro caso para buscar usuario “similares” es su participación o interés en los distintos repositorios, pero debido a las limitaciones antes mencionadas del API el vínculo entre Usuario->Repositorio no es el mismo que el de Repositorio->Usuario. En el primer caso un Usuario se relaciona con un Repositorio si es dueño del mismo, o si lo marcó como starred, mientras que en el segundo caso un usuario se relaciona con un repositorio si realizó algún commit en el mismo. Por tal motivo no sería correcto hablar de un sistema de recomendaciones de filtrado colaborativo “puro”.

Diseño

A continuación se detalla la arquitectura utilizada para implementar la solución:



Consta de una aplicación web sencilla. La autenticación de los usuarios es mediante el estándar OAuth. Cuando el usuario se conecta se le redirige al Login de GitHub, durante el cual se le solicitan permisos para que la aplicación pueda acceder a la información de sus repositorios. Esta forma de autenticación le asegura el usuario que solo él y GitHub tienen acceso a sus credenciales.

Una vez el usuario se autentica se le redirige nuevamente a nuestra aplicación, la cual muestra un listado de los repositorios recomendados.

Para cada repositorio se muestra el nombre, el valor de relevancia (correspondiente a la cantidad de caminos que conectan el usuario al repositorio, como se explicó en la sección anterior) y un enlace para acceder al repositorio en el sitio de GitHub.

Implementación

La implementación está desarrollada en Visual Studio en lenguaje C#, utilizando MVC como arquitectura. Para poder trabajar con la API de GitHub se tuvo que instalar el paquete Octokit que nos brinda todas las funcionalidades de la API.

Lo primero que se debe realizar es registrar nuestra aplicación en GitHub.com para obtener el secreto del cliente de nuestra aplicación. Para ellos se accede al link https://github.com/settings/applications/new. Luego de registrar la aplicación obtenemos el identificador de cliente y el secreto de cliente.

Nuestro sitio redirige al usuario a la URL de inicio de sesión de GitHub Oauth con información de identidad de nuestra aplicación y a su vez la lista de permisos que solicita en la cadena de la consulta. La página de inicio de sesión de GitHub Oauth le solicita al usuario que acepte o rechace la solicitud de autenticación, si el usuario aún no ha iniciado sesión en GitHub se le solicitará que inicie sesión primero.

Si el usuario hace clic en “Autorizar aplicación”, esta página redirige a nuestro sitio con un código de sesión especial. Luego, nuestro sitio realiza una solicitud de servidor a servidor intercambiando el código de sesión especial y el secreto del cliente de nuestra aplicación por un token de acceso OAuth. Luego, con el token y la libreria Octokit.Net se podrá realizar otras solicitudes a la API.

A partir de este momento vamos a poder trabajar tanto con la API como con los repositorios del usuario que inició sesión. Por lo que, lo primero que se hace es obtener todos los repositorios del usuario logueado, y a partir de estos todos sus contribuyentes. Debido a las limitaciones de consultas a la API, debemos limitar la cantidad de usuarios de similares intereses sobre los que trabajaremos. El límite utilizado en nuestro caso es 10, el cual nos permite probar el funcionamiento del sistema.

Luego, se obtienen los repositorios de los contribuyentes y estos son los que se le recomiendan al usuario.

Además, se obtienen repositorios a los cuales el usuario logueado le dió starred.

La lista de recomendaciones es mostrada al usuario en una tabla dinámica en la página principal de la aplicación ordenadas de forma decreciente según la cantidad de caminos posible que haya para cada repositorio. No son incluidos en la lista los repositorios donde el usuario logueado es propietario.

Además, se tiene un timer a nivel del cliente que restablece los repositorios cada una hora, esto es porque la API solo nos permite realizar una cantidad limitada de consultas por hora.

Evaluación y resultados

Se realizaron pruebas del sistema en forma local, utilizando nuestros propios usuarios de GitHub.

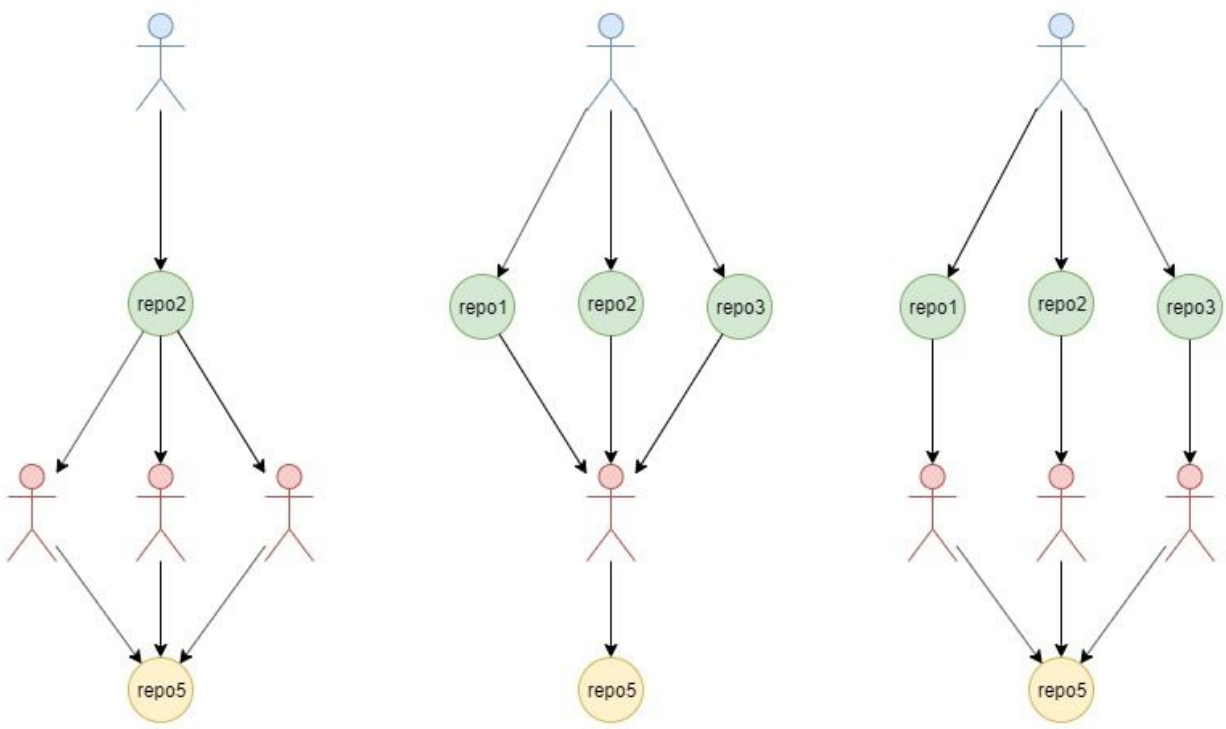
Las pruebas consistieron en marcar algunos repositorios como starred, y recargar el sitio.

Lo primero que se observó es que efectivamente se generaban nuevas recomendaciones basadas en los repositorios marcados.

Lo segundo y más importante es que estos repositorios resultaron efectivamente de interés para nosotros. A modo de ejemplo, al marcar los repositorios *django/django* y *ansible/ansible*, una de las primeras recomendaciones fue el repositorio *robbyrussell/oh-my-zsh*, el cual es de particular interés alguien que acostumbra trabajar por línea de comandos utilizando la shell *zsh*.

También hay que destacar que no todos los repositorios que alcanzan un buen puntaje son interesantes. Por ejemplo puede darse el caso en que muchos repositorios alcancen un buen puntaje debido a la cantidad de caminos que hay entre el usuario que consulta y el dueño de dicho repositorio, pero el repositorio en sí puede ser poco interesante o poco popular.

Para ejemplificar lo anterior, consideremos la siguiente imagen.



En los 3 casos el repositorio **repo5** obtiene puntuación 3, pero puede que no sea igual de interesante para el usuario.

Problemas presentados

El principal problema que encontramos (y que ya discutimos), fue la limitación para obtener información, y la cantidad de información existente. Este problema nos llevó a utilizar las grafos para la generación de recomendaciones, y determinó el tipo de relaciones que representamos en dicho grafo.

Otro problema que se encontró, es que el tamaño del grafo que se genera puede ser suficientemente grande como para que el sistema sea inutilizable. Esto nos obligó a colocar límites en el tamaño del grafo, posiblemente descartando información que podría haber generado mejores recomendaciones para el usuario.

Conclusiones

Se logró implementar un sistema que se integra con la autenticación de GitHub, y realiza recomendaciones al usuario logueado de acuerdo a su actividad.

Es posible utilizar relaciones y grafos para realizar recomendaciones. La ventaja es que no es necesario disponer de todo el universo de información, sino que se construyen las

recomendaciones partiendo del usuario que consulta. La desventaja de este sistema es que al ser información limitada, la relevancia de las recomendaciones puede no ser la mejor, y que la información debe estar disponible en una forma que permita armar el grafo de forma sencilla y directa. Por lo tanto es un método que puede aplicar para casos particulares.

El uso de grafos para sistemas de recomendaciones resulta interesante, debido a que son entidades muy estudiadas y para las que existen múltiples algoritmos que nos ayudan a resolver distintos problemas sobre los mismos. El problema en nuestro caso fue contar la cantidad de caminos entre dos nodos en un grafo dirigido y sin loops.

Posibles mejoras

- Agregar una base de datos para mantener los repositorios recomendados y no realizar una consulta cada vez que se refresque la página principal. Manteniendo además para cada usuario que se loguea la fecha de la última actualización de la lista de repositorios recomendados para poder renovarlos cada cierto tiempo. Otra forma posible de lograr esto sería guardar la lista en el caché del navegador del usuario, con una expiración adecuada.
- Agregar a nivel de base de datos la posibilidad de que el usuario tenga una lista negra de repositorios. Esto es, al ingresar un usuario a visualizar los repositorios recomendados, aquellos que no le sean de interés sean descartados y almacenados como repositorio sin importancia ya que cada vez que vence el tiempo de la última actualización este repositorio sería traído nuevamente y poder así descartarlo.
- Cuando un usuario se loguea consultar la fecha de última actualización de los repositorios y si son datos viejos renovarlos consultando la API.
- Nos basamos solo en una parte de la API de GitHub, se puede ampliar el sistema y recolectar repositorios desde otras secciones para poder brindar una mayor cantidad de recomendaciones.
- Agregar un filtrado de recomendaciones para los repositorios encontrados según la similitud con los repositorios del usuario (propios y favoritos) utilizando palabras claves que coincidan.
- Priorizar los repositorios recomendados en base a su actividad, dándole una puntuación mejor a aquellos repositorios con actividad reciente.
- Puede ser interesante realizar variaciones en el algoritmo utilizado para generar las recomendaciones, por ejemplo en lugar de contar la cantidad de caminos, contar la cantidad de aristas, o la cantidad de aristas y vértices. Una vez teniendo las distintas variaciones se podrían evaluar con un conjunto de usuarios para determinar en promedio cual genera mejores recomendaciones.

Referencias

- https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine
- https://developer.github.com/v3/
- https://haacked.com/archive/2014/04/24/octokit-oauth/
- https://github.com/octokit/octokit.net/blob/master/Octokit.Tests.Integration/Clients/RepositoriesClientTests.cs