[IIC2433] Entrega 1 Proyecto Minería de datos

Maximiliano Friedl y Felipe Gómez

Domingo 28 de octubre de 2018

1 Introducción: Heroes of the Storm (HOTS)

Se desea hacer un proyecto de optimización en torno al videojuego Heroes of the Storm. Este es un videojuego multijugador masivo en línea de Blizzard Entertainment, que se juega en formato 5 vs 5 jugadores, en donde cada uno controla a un héroe con el objetivo de destruir el Nexo enemigo (el edificio principal). El videojuego entra en a categoría de *Mobile Online Battle Arena* (MOBA), del mismo estilo que otros títulos famosos como *League of Legends*, *Dota II* o *Smite*.

Actualmente el videojuego cuenta con una comunidad de casi 6.5 millones de jugadores y torneos con más de 11 millones de dólares en premios.

2 Investigación del estado del arte y reestructuración del problema

2.1 Resultados de la investigación

En nuestro primer encuentro con el ayudante encargado, se nos recomendó hacer una investigación exhaustiva del estado del arte actual con respecto al problema que queríamos solucionar. En opinión del ayudante, es un problema bastante complejo como para partir desde cero, por lo que nos sugirió seguir adelante con la idea original solo si encontrábamos una base sólida desde la cual comenzar nuestro proyecto.

Al realizar la investigación sobre el estado del arte actual del problema, nos dimos cuenta de que no hay investigaciones sobre nuestro tema de interés: la recomendación de un héroe durante el *draft*. Tampoco hay trabajos del mismo género y ni siquiera hay algo similar para otros videojuegos. Las investigaciones

actuales se centran en cómo deben ser los jugadores para maximizar la eficiencia de un equipo y no en la elección de los personajes dentro del videojuego. Después de mucho investigar, decidimos que para el problema escogido no es factible partir desde algún trabajo realizado por expertos simplemente porque dicho trabajo no existe o no aplica a esta situación.

2.2 Nuevo problema a solucionar, supuestos y relación con el dilema original

De acuerdo a lo anteriormente señalado, decidimos seguir el consejo del ayudante encargado y cambiamos el enfoque del problema. Ya no intentaremos recomendar el próximo héroe a elegir dado el estado actual del *draft* como se pretendía hacer en un comienzo (al menos no directamente). En lugar de esto, lo que se hará es *dado un estado del draft, estimar la probabilidad de victoria de un equipo del otro*. Para ello, se tomarán en consideración los siguientes supuestos:

- En una partida no existen empates, es decir, el equipo 1 es ganador de una partida si y solo si el equipo 2 fue el perdedor de esa partida. En otras palabras, se niega la posibilidad de empate
- Se pretende hacer un clasificador binario sobre el resultado de la partida para el equipo 1, es decir, a cada resultado $y \in 0, 1$ se le asignará 1 si el equipo 1 fue el ganador y 0 en caso contrario
- Se asume que existe una forma válida de encontrar la probabilidad con la que un dato pertenece a la categoría y que esto se puede interpretar como la probabilidad de victoria de un equipo. Es otras palabras, si el resultado de la clasificación para un estado X del *draft* es 1 con una probabilidad de 0.6, esto significa que el equipo 1 tiene un 60% de probabilidades de victoria dado el escenario X
- El videojuego ha lanzado héroes nuevos constantemente desde su lanzamiento hace varios años. Por tanto, es muy probable que los héroes más nuevos no hayan existido al momento en el que se jugaron partidas más antiguas. Por ejemplo, el último héroe en salir, Malganis, está disponible desde octubre, por lo que estrictamente hablando no sería válido que se recomendara para una partida jugada en el 2016. A pesar de esto, se ha

decidido ignorar esta restricción, pues se asume que el modelo será ocupado para partidas "en el presente" (con las últimas actualizaciones de héroes, mapas y balance del juego), por lo tanto, no tiene sentido hacer recomendaciones para un estado del videojuego en el que nadie volverá a jugar jamás

- La predicción de la victoria de un equipo por sobre el otro asumirá que ambos equipos se encuentran en igualdad de condiciones: los jugadores de ambos equipos son igualmente hábiles, se coordinan con la misma eficiencia y tienen el mismo grado de estabilidad en la conexión. La idea es asumir en la medida de lo razonable que el resultado de la partida puede ser explicado mediante la elección de héroes en el *draft*

- La información de cada *replay* tiene una descripción de eventos en donde sale, entre otras informaciones, cada uno de los héroes escogidos y prohibidos. Se asume que esta información viene en orden, es decir, el primer héroe que sale en el archivo fue la primera elección en el *draft*. Aunque no hay confirmación de los desarrolladores sobre este punto, todo parece indicar que esto efectivamente se cumple

En un principio, este problema parece ser de bastante menos complejidad que el definido inicialmente, ya que ahora es solo un problema de clasificación de un escenario en la categoría ganar/perder y eso, a priori, debería ser un problema más manejable, de un nivel de complejidad similar a lo visto en el curso.

Fue cuando estábamos decidiendo los detalles de los entregables finales que surgió la siguiente idea: el nuevo problema recibirá como input un escenario del *draft* y entregará 1 o 0 si el equipo 1 gana o pierde de acuerdo a ese escenario. Si fuera posible hacer un modelo que entregue la probabilidad con la que se está clasificando al dato en una determinada categoría, eso podría interpretarse como la probabilidad de victoria. Ahora, imagínese la situación en donde se han hecho todos los *bans* y todos los *pick* a excepción del último héroe del equipo 2. En este caso solo falta un héroe por decidir. Una forma ingenua, pero válida de encontrar al mejor héroe es simplemente probar todos las elecciones válidas de héroe y elegir la que maximice el *winrate*. De esta manera, estaríamos resolviendo el problema inicial, en apariencia inabordable, a partir de uno que sí sabemos cómo solucionar.

Es importante destacar que esta idea surgió recientemente y que es necesario

discutirla a fondo con el ayudante para asegurar la factibilidad y que efectivamente esto otorga una solución al problema original.

2.3 Proyecciones Entrega 2

Para la Entrega 2, se espera entregar los siguiente:

- Base de datos con al menos 100.000 *replays* para entrenamiento y 10.000 para *testing*. Los números pueden variar de acuerdo a las pruebas a realizar
- Al menos dos formas diferentes de solucionar el problema: el modelo hecho a partir de *Support Vector Machines* (SVM) y una Red Neuronal con Capa *Softmax*. Se discutirá con el ayudante sobre la posibilidad de agregar una tercera forma de resolver el problema
- Comparación entre los diferentes modelos. Se pretende establecer cuál de las implementaciones es la que resuelve de mejor forma el problema. Para ello, se probarán todos los modelos con nuevas repeticiones y se establecerá cuál modelo predice mejor la victoria de un equipo dado un determinado *draft*. Se analizará también el alcance del modelo, cómo reacciona cuando se filtra por liga [1] o cuando se toma en cuenta las preferencias de cada jugador (recuerde que la mayoría de los jugadores no tiene a todos los héroes y de todos los disponibles solo sabe jugar bien con algunos de ellos)
- Hacer un análisis de los resultados desde el punto de vista del jugador: qué tanto sentido hacen las predicciones realizadas por los modelos, si concuerda con lo que se ve realmente en el juego. Se pondrá énfasis en aquellos patrones que no son frecuentemente utilizados y se analizará el *winrate* obtenido a partir de ellos. Este es uno de los criterios de éxito del modelo: la capacidad de descubrir patrones valiosos que están escondidos a simple vista y que, por lo tanto, la mayoría de los jugadores pasa por alto
- Discusión final sobre cómo podría expandirse el modelo para ajustarse aun más a la realidad del juego. Hablar sobre cómo estos modelos podrían extrapolarse para la toma de decisiones de otros videojuegos similares y si hay alguna forma de utilizar esto en algún otro contexto

[1] En HOTS, los modos de juego *Hero League* y *Team League* separan a los jugadores por ligas: bronce, plata, oro, diamante, maestro y Gran Maestro, donde bronce son los jugadores de más bajo nivel y Gran Maestro tienen un nivel similar al de un profesional del videojuego.

3 Preprocesamiento y base de datos

3.1 Construcción de la base de datos

Como se mencionó en la entrega anterior, para la construcción de la base de datos se utilizó HotsAPI. Particularmente, utilizamos el *endpoint*:
`hotsapi.net/api/v1/replays/paged`
A continuación, se hablará en detalle de cada paso realizado en un *script* de *Python 2* para poder llevar a cabo la construcción de la base de datos. Este *script* se puede encontrar en nuestro repositorio público, bajo la carpeta `Preprocessing/main.py`.

3.1.1 HotsAPI

Con el uso de esta API fuimos capaces de obtener cierta información básica sobre las *replays* almacenadas, pero lamentablemente, esta información no era suficiente para lo que queríamos lograr. Afortunadamente, en cada respuesta de la API, se incluye un *link* para poder descargar el archivo `.StormReplay` original, desde el cual es posible extraer la información completa respecto a la partida relacionada a esa *replay*. Esto hizo que el problema aumentara de tamaño, pues cada repetición es un archivo bastante grande (alrededor de 2MB), por lo que hacer un análisis para cientos de miles de repeticiones se volvió una tarea considerablemente más difícil.

La forma de descargar las repeticiones se explica en la siguiente subsección.

3.1.2 Amazon AWS S3 bucket

El servicio de almacenaje que utiliza la API para guardar las replays, es *S3* provisto por *Amazon Web Services*, por lo que es necesario tener una cuenta en AWS. Además, la configuración del *bucket* es de *Requester pays for traffic*.

Para unas pocas repeticiones, se puede utilizar el servicio gratuito de AWS. De todas formas, una limitante para utilizar este servicio es que la inscripción requiere de una tarjeta de crédito.

Recomendación: si no se desea colocar los datos de una tarjeta de crédito real, existen bancos (por ejemplo, el Banco de Chile) que crean una tarjeta de crédito virtual con un saldo predefinido (por ejemplo, 1 dólar). Esta es una buena práctica para asegurarse de poder registrarse en la aplicación sin temor a gastar dinero por un uso intensivo de los servicios.

3.1.3 *Heroprotocol* de *Blizzard*

Una vez que ya tenemos el archivo `.StormReplay` es necesario extraer la información de este. Estos archivos están codificados completamente a nivel de bytes, por lo que es imposible lograr descubrir el significado de estos. Afortunadamente, *Blizzard*, la compañía creadora de *Heroes of the Storm*, tiene un proyecto público en *GitHub* llamado *Heroprotocol*, el cual es, justamente, un *parser* para este tipo de archivos, escrito en *Python 2*.

Haciendo uso de este recurso, fuimos capaces de extraer toda la información que consideramos relevante de cada *replay*. Dentro de esta información se encuentra: héroes escogidos, héroes prohibidos, nivel en que tenía cada jugador el héroe que escogió en el momento del *draft*, tipo de partida, mapa de juego y resultado final de la partida.

Finalmente, nuestro *script* elimina el archivo `.StormReplay` para no acumular un exceso de archivos de alto peso (entre uno y dos megabytes por *replay*).

3.1.4 Almacenamiento

Para el almacenamiento de toda la información utilizamos *PostgresSQL*. Cada *replay* es ingresada a una tabla llamada `replay`, con toda la información mencionada anteriormente. Decidimos usar una base de datos como *postgres* en vez de simples archivos `.csv` ya que la API cuenta con más de 12 millones de *replays*, por lo que eventualmente el archivo podía volverse demasiado grande y generarnos problemas.

Dentro de nuestro repositorio público es posible encontrar el archivo `database.SQL`, con el cual se puede crear una réplica de la tabla que estamos utilizando para guardar la información. Además, hay un archivo llamado `database_to_csv.py`, el cual es un *script* que toma la totalidad de nuestra base de datos, y lo transforma a un archivo `.csv`. Además, con el uso de este *script*, se incluyó en el repositorio el archivo `data.csv`, en el cual se encuentra la información de las más de 5900 *replays* que logramos procesar hasta este momento.

3.2 Preprocesamiento

Con todo el proceso explicado anteriormente, fuimos capaces de poder construir nuestra propia base de datos, además de tener un mecanismo para poder ampliar constantemente esta base de datos (la API recolecta nuevas *replays* todos los días, aportadas por muchos jugadores en todo el mundo), pero, el formato en

que se encuentra esta información no es apto para poder entrenar los distintos modelos que pretendemos crear.

Por esto, es que es necesario encontrar una codificación para la información que tenemos disponible, que sea entregable a nuestros modelos. Lo más tradicional, es utilizar *encodings* en forma de vectores, es por esto, que intentaremos vectorizar toda nuestra información.

Primero, el mapa escogido, será un vector de n dimensiones, donde n es la cantidad de distintos mapas presentes en el juego. Luego, el tipo de juego, en nuestro caso no es relevante, ya que nos estamos concentrando en el *draft* de una partida, y en nuestro caso, recopilamos solo información de partidas con *draft*, por lo que no hay variación de un tipo a otro.

Luego, hay que codificar los héroes escogidos, su nivel y los héroes prohibidos. Para logra esto, se identifican dos métodos. El primero es utilizar cuatro vectores de dimensión m (donde m es la cantidad de héroes presentes en el juego) cada uno, dos por cada equipo. Cada equipo tendrá un vector que representará los héroes escogidos y prohibidos (con un 1 en la posición de los héroes escogidos, y un -1 en la posición de los prohibidos), y un vector que representará el nivel de los héroes escogidos (con un valor entre 1 y 20 para representar el nivel en la posición del héroe al que corresponde).

Una segunda opción sería el utilizar 16 vectores en total, 10 de dimensión $m+1$, donde m nuevamente es la cantidad de héroes presentes en el juego, y 6 de dimensión m. Cada uno de los vectores $m+1$ representará un héroe escogido durante el *draft*, donde su último valor será el nivel asociado a ese héroe, y cada uno de los vectores m, será un héroe prohibido.

El problema de este *encoding* es que no se está haciendo diferenciación de qué héroe pertenece a qué equipo, o por qué equipo fue prohibido, más que por la posición en que está entrando al modelo. Quizás una variación a considerar de este sería agregar una dimensión extra a cada vector que represente el equipo por el que fue escogido.

Dentro de todo, durante el desarrollo de los modelos será necesario el probar cual dará mejores resultados. Cada uno tiene sus ciertas ventajas. El segundo está dando mayor relevancia a en qué fase del *draft* se escogió a cada héroe, lo que puede ser relevante, mientras que el primero deja de lado todo esto y da más importancia a el resultado final, pero disminuye la complejidad del modelo considerablemente, haciendo que este sea mejor generalizando, y haciéndolo más resistente al *overfitting*.

4 Modelos a considerar

Tal como se mencionó anteriormente, la idea es construir modelos clasificadores, en donde existen dos resultados posibles: 1 si el modelo cree que el equipo será ganador y 0 si el modelo cree que el equipo será perdedor. Luego, con este resultado, extraer la "confianza", o la posibilidad con que el modelo cree que este será el resultado, y este valor será interpretado como la posibilidad de ganar de un equipo. Para esto, hemos considerado la utilización de dos [2] técnicas de *machine learning* distintas, descritas a continuación.

4.1 SVM

SVM es una técnica de *machine learning* muy utilizada debido a sus buenos resultados. El concepto básico de esta es, tal como en la regresión logística, lograr trazar una recta que separe nuestro *dataset* en dos clases, pero a diferencia de la regresión, esta recta se obtiene a través de la solución de un problema de optimización con el objetivo de obtener *la mejor* recta que separe las dos clases.

Además, de igual forma que como se vio en clases para la regresión, se utiliza algo conocido como *kernel trick* para aumentar la dimensionalidad del problema, y ser capaces de encontrar hiperplanos que realizan la separación de la mejor forma posible, que cuando es reducido de vuelta a dos dimensiones, ya no se trata de una recta, sino que de una línea con cualquier forma.

¿Por qué SVM? Pues, como ya se ha dicho, estos son conocidos por sus buenos resultados, pero este no es el único motivo. Si bien el concepto detrás de los SVM es bastante simple (obtener una "recta" que realice la mejor separación de dos conjuntos), el proceso matemático para resolver este problema no es tan sencillo, pero afortunadamente, existen implementaciones ya hechas, y eficientes, en librerías tales como *scikit learn*.

Ahora, lo importante no es solo que existen implementaciones ya hechas, si no que estas además incluyen funciones tales como *predict_proba()* de *scikit learn*, que nos entrega información de cual es la probabilidad con que el modelo está escogiendo ese resultado, con la cual podemos obtener justamente el resultado que necesitamos para nuestra problemática: conocer qué tan probable es la victoria de un equipo dada la situación actual del *draft*

[2] Incluir más de dos técnicas está aun en evaluación y depende de los resultados obtenidos en los dos primeros modelos

4.2 Red neuronal

Las redes neuronales son un método de *machine learning* que se basa en la construcción de una función matemática, usualmente altamente complejas, utilizando unidades conocidas como neuronas. Cada neurona tiene valores de entrada, una función de activación, y valores de salida. Básicamente lo que esta realiza es aprender del *set* de datos para encontrar los mejores pesos para cada valor de entrada para cada una de las neuronas de la red.

Esta es una explicación muy [3] sencilla de el funcionamiento de una red neuronal, pero para objetivos de esta entrega el funcionamiento de una red no es lo importante, si no por qué estas pueden resultar útiles en nuestra problemática.

El objetivo al utilizar redes neuronales es ser capaces de construir un clasificador con ellas. Para esto, podemos utilizar dos neuronas en nuestra capa de salida, donde cada una representará qué equipo será el ganador. Si la primera neurona es un 1, entonces el equipo ganador sería el equipo uno, si la segunda neurona es un 1, entonces el segundo equipo sería el ganador.

Ahora, para obtener las probabilidades con que nuestra red neuronal cree que ganará cada equipo, podemos utilizar una capa de salida con función de activación softmax. Esta función de activación nos ayudará a que nuestro clasificador se convierta en un predictor de qué tan probable es que gane un equipo o el otro. Es decir, exactamente lo que queríamos lograr.

[3] muy, MUY

5 Dificultades

En la siguiente sección se listarán las distintas dificultades con las que nos encontramos durante el desarrollo del progreso actual en que se encuentra el proyecto.

5.1 HotsAPI

Durante la construcción de la base de datos ocurrió un problema con la API en utilización. Las *requests* solicitando información de *replays* comenzaron a ser respondidas con *arrays* vacíos. Esto causó que no pudiésemos seguir *parseando* información hasta que la API retornó a su comportamiento original. Imagen (imgur) con una *request* ejecutada desde *Postman* mostrando los resultados vacíos.

Para solucionar este problema, se abrió una *issue* en el repositorio con el código fuente de la API, ya que por el momento es algo que se encuentra totalmente fuera de nuestras manos. Esperamos que el problema (o quizás no es un problema, solo algún factor que desconocíamos) pueda ser solucionado lo antes posible para poder continuar con la ampliación de nuestra base de datos, ya que es un proceso lento de ejecutar, porque debemos descargar y *parsear* una gran cantidad de archivos de tamaño no insignificante.

De no solucionarse el problema, se va a cambiar el lugar desde donde se están descargando las *replays*, ya que afortunadamente hay más sitios desde donde realizar las descargas. De ser esta la medida a tomar, habrá que cambiar varios aspectos en el código del preprocesamiento actual.

5.2 *Requester pays*

Para la construcción del *parser* el poder descargar las *replays* fue una gran dificultad, ya que significaba que debíamos crearnos y configurar una cuenta en *Amazon Web Services*, donde además estaríamos utilizando un servicio que es de pago.

De acuerdo al panel de control de AWS hay un límite de 20.000 *GET requests* gratuito mensual por cuenta, por lo que considerando que tenemos dos cuentas (una de cada uno), podíamos ejecutar un total de 40.000 *requests*, y por lo tanto obtener 40.000 *replays*.

Para aumentar este número, hicimos uso del paquete de *GitHub student* y su beneficio en *Amazon Web Services*. Si bien aún no ha sido necesario, ya que

no hemos llegado al máximo gratuito, eventualmente esto nos será de mucha ayuda.

5.3 *Heroprotocol* y *replays*

Muchas de las *replays* analizadas no traían toda la información respecto a los héroes prohibidos. Si bien estos debiesen ser seis, algunas veces *heroprotocol* entregaba información de menos.

Una posibilidad para este problema es que se trate de *replays* antiguas, cuando en el *draft* efectivamente existían solo cuatro héroes a prohibir, pero esta posibilidad se descartó ya que nuestro *parser* se construyó para utilizar solo *replays* recientes, ya que nos entregarán información más veridica del estado actual del juego, además de que el número de héroes prohibidos entregado por *heroprotocol* no era siempre cuatro cuando no era seis, si no que podía ser cualquier número entre uno y seis.

Es por esto, que se decidió el poder aceptar columnas con valor `null` para los héroes prohibidos, y así no perder información de una gran cantidad de *replays*. Si a futuro se considera que la cantidad de *replays* que se tiene es lo suficientemente grande como para poder filtrar aquellas con datos incompletos, entonces se puede filtrar la base de datos antes de entrenar los modelos, pero por el momento se consideró que esta era la mejor opción.

6 Conclusiones y aprendizajes de la entrega

A continuación, se entrega un comentario general de las principales conclusiones y lecciones aprendidas luego de realizar esta entrega:

- Para la próxima vez que se haga un proyecto similar, hay que hacer un mejor análisis del riesgo que implica basar todo el proyecto en una API de la cual no se tiene control. Pasó en esta entrega que la API se caía a ratos por razones desconocidas, lo que introducía una clara incertidumbre en el proyecto y que en más de una ocasión originó retrasos en las funcionalidades a implementar

- Con relación al punto anterior, es de prioridad máxima definir medidas de mitigación ante la activación de un riesgo de alto impacto, como lo es el que HotsAPI dejara de funcionar. Es un buen servicio para bajar cantidades grandes de *replays* del videojuego, pero no es la única. En esta misma línea, se podría modificar el Pre-procesamiento para que intercale entre distintos servicios de descarga de repeticiones. De esta manera, cuando un servicio esté de baja, el parser seguirá funcionando, pues hará la consulta a algún otro servicio de descargas

- Si se descargan *replays* desde varios sitios diferentes, se deberá introducir una medida que evite duplicación de los datos, es decir, que se descargue y procese varias veces el mismo archivo. Esto implica un cambio menor en el pre-procesamiento, pues por defecto cada *replay* ya incluye un identificador

- Para modelar un problema complejo, una buena forma de partir es enfocarse solo en las restricciones más importantes e ir iterativamente agregando más y más restricciones hasta acercarse a la situación del problema en la realidad. Para este caso, conviene ignorar en un comienzo aspectos como el mapa, el nivel de cada héroe para cada jugador, la liga o el *pool* de héroes de cada participante. Si se intentar modelar la totalidad del problema de una vez, muy posiblemente no se tenga claridad de los errores que pasen en el camino. Otra ventaja de agregar restricciones de forma iterativa es que el cambio en las predicciones del modelo otorga una intuición de cómo está afectando cada una de las restricciones que se van agregando a la decisión final del clasificador

7 Anexo

7.1 El *Draft*

Dentro del modo de juego competitivo, también conocido como *ranked game*, antes de comenzar la partida, existe un proceso conocido como *draft* [4], en el cual los jugadores de ambos equipos proceden a escoger los héroes con que desarrollarán la partida.

El *draft* es un proceso de 12 pasos, detallados a continuación:

1. Equipo uno prohíbe un héroe (este héroe no puede ser escogido por ningún equipo durante esta partida).
2. Equipo dos prohíbe un héroe.
3. Equipo uno prohíbe un héroe.
4. Equipo dos prohíbe un héroe.
5. Equipo uno escoge un héroe (Un jugador del equipo utilizará este héroe durante la partida).
6. Equipo dos escoge dos héroes.
7. Equipo uno escoge dos héroes.
8. Equipo dos prohíbe un héroe.
9. Equipo uno prohíbe un héroe.
10. Equipo dos escoge dos héroes.
11. Equipo uno escoge dos héroes.
12. Equipo dos escoge un héroe.

Además de los héroes prohibidos, cada héroe no puede ser escogido más de una vez. Por lo tanto, un héroe puede estar presente solo una vez en toda la partida (no es posible que esté una vez por equipo, tampoco). Esto nos lleva a que, para cada paso de este proceso, además de tener una gran cantidad de héroes a escoger, tengamos ciertas limitantes a los héroes que podemos elegir.

[4] Para ver un ejemplo de *draft*, se sugiere ver hasta el minuto 4:11 del siguiente link: https://youtu.be/GuXwWtAonYI

Un factor importante a considerar es que un jugador no necesariamente sabe ocupar a los 82 héroes disponibles en el juego, además de que es necesario que el jugador haya comprado previamente el héroe para poder utilizarlo. Por lo tanto, cada *draft* está limitado por restricciones propias de cada jugador.

Finalmente, dentro del juego existen ciertos tipos de héroes que son necesarios para cada equipo. Dentro de la totalidad de héroes, podemos clasificarlos en cinco roles básicos:

- Tanque: personaje con mucha resistencia que inicia los combates en primera línea
- Daño: utilizado para hacer la mayor cantidad de daño en poco tiempo, útil para derribar héroes enemigos
- *Bruiser*: un híbrido entre Tanque y Daño, muy útil en el 1vs1 y en composiciones específicas
- *Healer*: el personaje que puede sanar rápidamente a los héroes aliados y así mantenerlos con vida
- Especialista: por sus habilidades únicas tienen un rol que no encaja en el resto de las categorías

En general, se considera que como mínimo un equipo debe contener un tanque, un *healer* y un daño. Si ahora consideramos todos los factores mencionados anteriormente y el hecho de que para cada paso del *draft* se tienen solo 25 segundos para escoger o prohibir un héroe, el proceso puede volverse muy complejo y con demasiadas variables para considerar todas las opciones posibles y escoger la ideal, de ahí la utilidad de un sistema que permita tomar rápidamente la mejor decisión.

8 Enlaces de interés

En esta sección se incluye referencias a otros sitios que pueden ayudar a la comprensión del problema y de lo hecho en esta entrega.

8.1 Contexto del problema

- Sitio oficial de *Heroes of the Storm*
- *Heroes of the Storm match Grubby*: video con una partida jugada por un ex jugador profesional. Hasta el minuto 4:11 ocurre el draft de la partida, se recomienda ver si se tiene problemas para entender el proceso de *draft*
- HotsAPI: API utilizada para descargar las repeticiones de partidas
- *Heroprotocol Blizzard*: Este link dirige al repositorio en GitHub de Hero-Protocol. El proyecto permite obtener información de las repeticiones. El código está implementado en Python 2.

8.2 Referencias de la investigación realizada

- *Investigating the Impact of Game Features and Content on Champion Usage in League of Legends*:
- *Analysis of the Heroes of the Storm*:
- *The Well-Played MOBA: How DotA 2 and League of Legends use Dramatic Dynamics*:

8.3 Otros

- Repositorio público con nuestro código. Aquí se incluye todo lo realizado hasta el momento. Incluye el parser de Blizzard, el código en Python 2 para el Preprocesamiento de la base de datos, así como el inicio de los primeros modelos a realizar