

# TRABAJO FINAL DE MÁSTER

## Predict The Rift



Mario Martínez Garrido

# Índice

- 1) Resumen
- 2) Objetivos
- 3) Introducción
- 4) Datos
- 5) Fundamentos teóricos
- 6) Desarrollo del proyecto
- 7) Conclusiones
- 8) Escalabilidad del proyecto
- 9) Manual de usuario front end
- 10) Bibliografía y referencias

# 1) Resumen

El proyecto se centra en intentar predecir el resultado de una partida con unos datos estadísticos, para ello, se aplicaran modelos de machine learning y se medirán sus métricas para conocer cuál es el mejor modelo y si de verdad es posible predecirlo.

# 2) Objetivos

Predecir la probabilidad de victoria del Blue Team en cada minuto de la partida, del minuto 0 al 15. Usando machine learning.

# 3) Introducción

League of Legends es un videojuego del género multijugador de arena de batalla en línea (MOBA) y deporte electrónico el cual fue desarrollado por Riot Games en el año 2009.

Durante estos más de 10 años de vida del juego, el interés por el videojuego y por su esport ha crecido de manera inimaginable. Hoy en día cuenta con una player base media mensual de más de 130 millones de jugadores, llegando a picos diarios de más de 15 millones de jugadores.

En cuanto a su esport, en muy poco tiempo se ha profesionalizado a niveles de deporte tradicional, cuenta con figuras, instalaciones y sueldos competitivos. Sobre su viewers se ha llegado a picos de más de 80 millones de espectadores viendo un partido, lo que demuestra el crecimiento de esta industria.

Es un mercado en auge y en constante crecimiento, por lo que desarrollar un proyecto funcional y que cuente con un buen mercado puede llegar a ser muy interesante y con alta rentabilidad, ya que los costes de producción deberían de ser bajos.

Es un mundo en el que la mayoría de aplicaciones o proyectos interesantes son privados y desarrollados por analistas que trabajan en los diferentes equipos de esports. Por lo que dada la competencia es difícil compartir proyectos o hacerlos públicos.

## 4) Datos

- \* gameId = Id de la partida y nuestra key del dataframe
- \* kd + n = Diferencia de kills entre los 2 equipos
- \* gd + n = Diferencia de oro entre los 2 equipos
- \* xpd + n = Diferencia de experiencia total entre los 2 equipos
- \* csd + n = Diferencia de minions total entre los 2 equipos
- \* dd + n = Diferencia de dragones entre los 2 equipos
- \* hd + n = Diferencia de heraldos entre los 2 equipos
- \* td + n = Diferencia de torres destruidas entre los 2 equipos
- \* pd + n = Diferencia de placas destruidas entre los 2 equipos
- \* tdd + n = Diferencia de daño total entre los 2 equipos
- \* wd + n = Diferencia de wards puestos entre los 2 equipos
- \* BlueWin = Boolean sobre la victoria del equipo azul

temp_names	kd10	gd10	xpd10	csd10	dd10	hd10	td10	pd10	tdd10	wd10	BlueWin
gameId											
EUW1_5763325310	10	16400	10552	-36	-1	0	0	4	43293	16	True
EUW1_5763410685	5	5005	-5491	11	1	-1	0	-1	-3935	73	True

## 5) Fundamentos teóricos

Modelos usados:

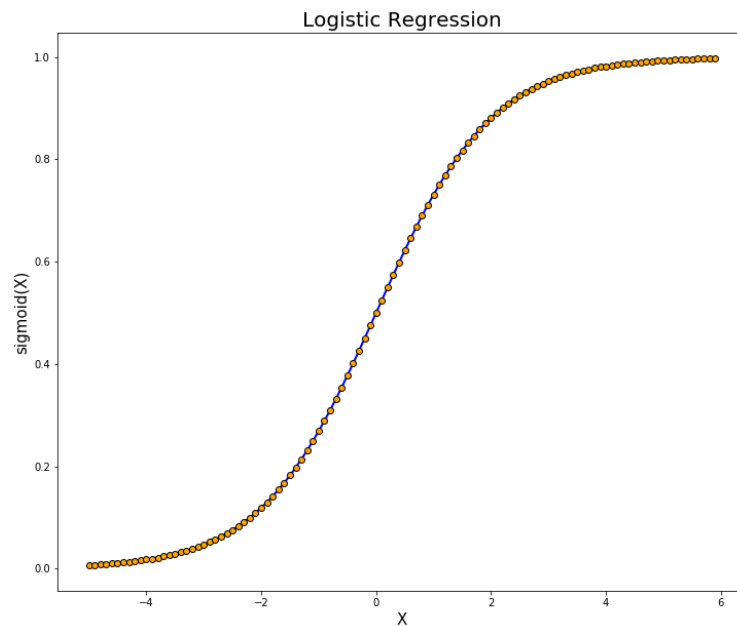
- Logistic Regression

Es un modelo estadístico normalmente usado para problemas de clasificación o análisis predictivo. La regresión logística estima la probabilidad de que un evento ocurra, como en nuestro caso que gane el BlueTeam o no.

Se puede representar con las siguientes fórmulas:

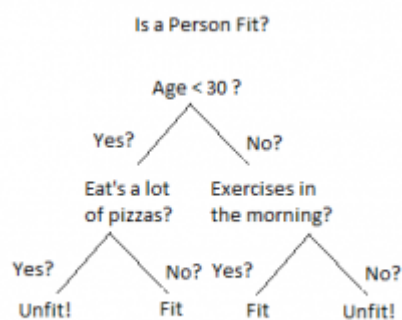
$$\text{Logit}(\pi) = 1/(1 + \exp(-\pi))$$

$$\ln(\pi/(1-\pi)) = \text{Beta}_0 + \text{Beta}_1 * X_1 + \dots + B_k * K_k$$



## - Decision Trees

Es un modelo supervisado de machine learning donde los datos se dividen constantemente basado en un parámetro, estas decisiones pueden llamarse nodos de decisión u hojas.



Hay 2 clases de decision trees, los de clasificación y los de regresión:

Clasificación - Son usados para decisiones en las que las variables son categóricas.

Regresión - Usados cuando la variable es constante como por ejemplo un número.

## Entropía

La entropía es la cantidad de desorden de la información, o la cantidad de aleatoriedad en los datos.

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

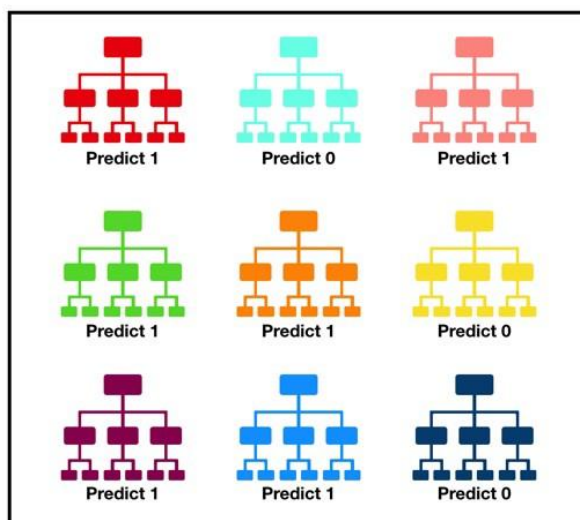
## Ganancia de la información

Es la cantidad de información que se gana gracias a una determinada feature.

$$IG(S, A) = H(S) - \sum_{i=0}^n P(x) * H(x)$$

### - Random Forest

Es un algoritmo de clasificación que funciona realmente bien, el modelo crea una gran cantidad de árboles de decisión que predicen una clase, se juntan los resultados y nuestra predicción es la clase con más votos.



- Artificial Neural Networks

Simulan el cerebro humano a través de una serie de algoritmos, están compuestas de cuatro elementos principales:

Inputs - Cantidad de datos que llega a la neurona

Weights - Los inputs son multiplicados por una variable llamada weights que determina la conexión entre 2 neuronas. Cada neurona tiene su propio peso que es determinado en el proceso de aprendizaje.

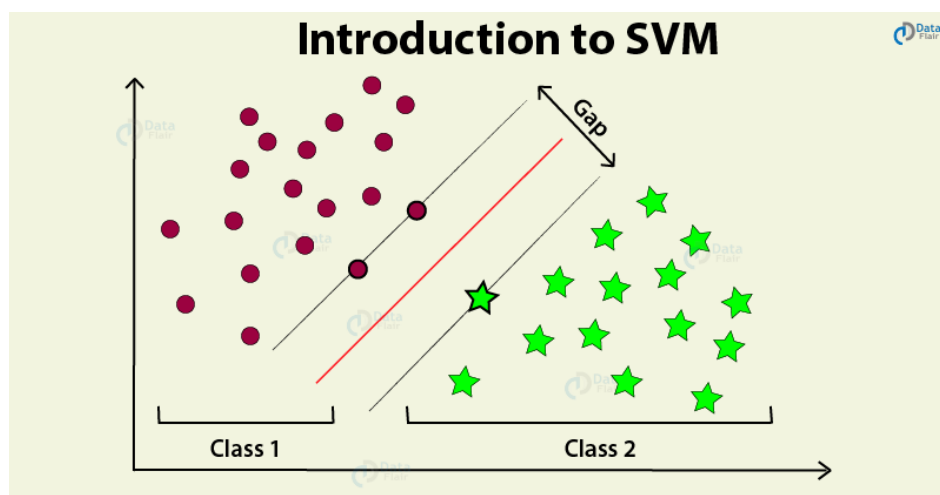
Bias - Valor que se añade al valor total calculado, se elige antes de la fase de aprendizaje y no viene dado de ninguna neurona.

Threshold - Margen de error

$$\sum_{i=1}^m w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

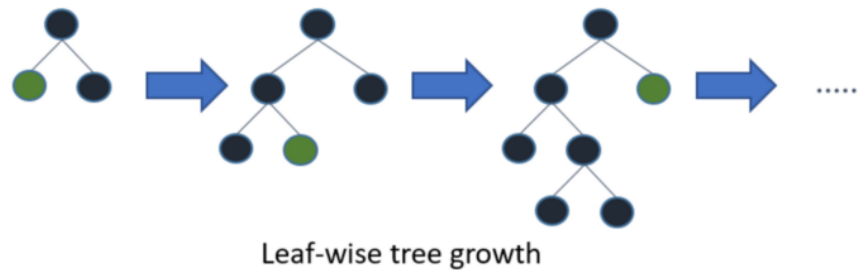
- SVM

SVM or Support Vector Machine es un modelo lineal para problemas de clasificación o regresión. Puede resolver problemas lineales o no líneas y funciona bien para problemas prácticos. El algoritmo crea una línea o un hiperplano que separa los datos en clases.



## - LightGBM

Es un modelo que usa gradient boosting basado en un algoritmo de árboles de decisión. En LightGBM el árbol crece verticalmente mientras que en otros modelos similares lo hace de manera horizontal, usa leaf-wise en vez de level-wise para decidir cómo crecer.



## Cross Validation

Es una técnica utilizada para evaluar los resultados de un algoritmo o modelo, consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar la precisión de un modelo que se llevará a cabo a la práctica.

## 6) Desarrollo del proyecto

**Iniciamos el proyecto con la data acquisition, cleansing and preparation (AddData)**



Para ello, tenemos un array de gameIdIds para no repetir partidas y así evitar los datos repetidos. Usaremos pickle para guardarlo sin necesidad de un servidor y de manera local.

Tras esta confirmación, empezaremos a lanzar peticiones a la api de riot, de la que obtendremos los 300 mejores jugadores de europa que usaremos para sacar sus partidas.

Sacamos los puuid de cada jugador para posteriormente sacar sus últimas 20 partidas. Añadiremos al array de gameIdIds para no repetirlas en un futuro.

Tras esto, crearemos un dataframe con la información de cada partida, incluyendo el gameId, el ganador de la partida y 10 variables a lo largo de los primeros 15 minutos.

Necesitaremos hacer un explode de las features ya que se han guardado en un array, las separaremos individualmente para poder analizarlas mejor posteriormente.

Una vez comprobado que el dataframe se ha guardado como queremos, pasaremos al siguiente notebook donde se hará el análisis.

### **Análisis de los datos y del problema (main)**

Empezamos realizando los import necesarios y la carga del dataframe. Aprovecharemos para realizar una visualización básica e inicial. También comprobaremos si las clases están repartidas equitativamente para evitar que las conclusiones estén afectadas.

Sin duda la variable más importante es la diferencia de oro y todas las demás giran en torno a ella, ya que cualquier diferencia positiva en esas variables se traducirá a más oro. La que tiene una relación más directa y fuerte es la diferencia de kills o “kd”.

Separaremos en train y test sets para el posterior análisis.

En este proyecto usaremos 6 modelos de clasificación que compararemos para posteriormente analizar la partida que queramos. Para esta comparación hemos usado cross validation y a su vez la roc curve para tener una referencia más visual.

El modelo que usaremos será el de LightGBM, en el que hemos modificado sus parámetros más claves para intentar conseguir una buena precisión y evitar el overfitting.

También parece que la logistic regression es un gran modelo para este proyecto y podríamos haber usado este modelo, gran parte de culpa la tiene que si comparamos las variables numéricas del dataframe todas apuntan a una logistic regression.

Tras realizar el análisis y la comparación de los modelos, podemos observar que ronda el 65-75% en la mayoría y que son similares.

En una partida competitiva, los objetivos como dragones, heraldos o las torres son las más importantes pero como nuestro análisis es de partidas clasificatorias o SoloQ, las kills parece ser el parámetro clave en todos nuestros modelos. Independientemente del minuto o franja de tiempo en los primeros 15 min.

También como es obvio en los primeros 3-5 minutos suele ser una partida 50/50 ya que normalmente los jugadores empiezan a jugar al minuto 1:40≈.

La última parte del proyecto es la de visualización, en la que os dejo varias maneras de hacerlo, tanto de manera estática sin UI como una manera en una web app con ui. (Aplicación flask)

## 7) Conclusiones

Un partido de League of Legends o de cualquier deporte tradicional incluyen muchas variables intangibles, por lo que obtener un porcentaje de precisión alto es difícil en este tipo de problemas.

Por lo que he investigado en proyectos similares, la precisión del modelo suele estar entre el 70-80%. Estamos hablando de una precisión similar a mi caso por lo que nos podemos dar por satisfecho, aunque hay maneras de mejorarlo que hablaremos a continuación.

## 8) Escalabilidad del proyecto

Hay diversos aspectos que mejorarían el proyecto y por lo que creo que tiene una alta escalabilidad:

Lo primero de todo, se podría hacer una interfaz que fuera en tiempo real y que estuviera en pantalla mientras juegas la partida, pero necesitamos que la empresa Riot Games nos facilitara los meta datos y poder acceder en tiempo real a las stats.

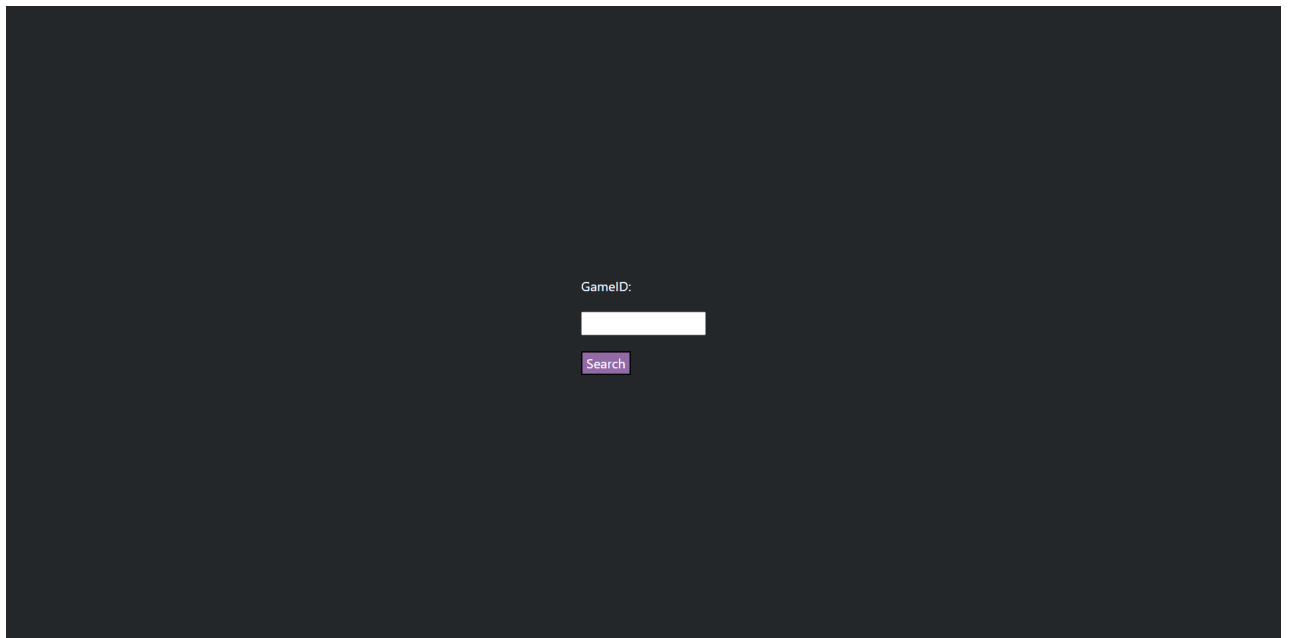
La segunda es que gracias a estos meta datos en tiempo real, tendríamos más datos y de mayor calidad, incluido información sobre las teclas presionadas o los clicks del ratón por lo que mi hipótesis es que habría mayor precisión en el modelo.

## 9) Manual del usuario

Después de encender la aplicación flask en localhost (instrucciones repo), deberemos acudir a la siguiente url, “localhost/tfm/all/all”, en mi caso es la siguiente:

<http://127.0.0.1:5000/tfm/all/all>

Al acceder nos encontraremos esta pantalla:

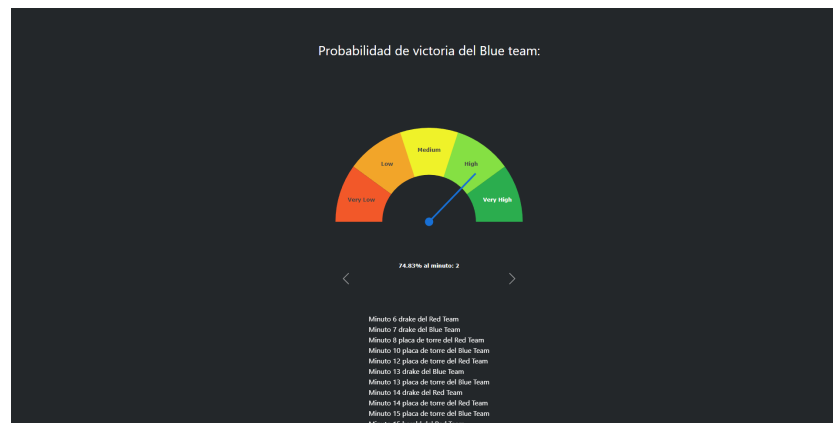
The image shows a dark-themed web interface. In the center, there is a label "GameId:" in a small, light-colored font. Below this label is a white rectangular input field. Underneath the input field is a purple button with the word "Search" written in white. The entire interface is set against a solid dark background.

Está compuesta por una celda donde deberemos introducir un gameId y un botón que debemos pulsar para buscar esa partida.

El formato de los gameId es el siguiente: EUW1\_5926384779

Se puede usar cualquier gameId si entras en el videojuego pero como ejemplo os dejo los siguientes:

EUW1\_5926384779 | EUW1\_5924551872 | EUW1\_5925960334 |  
EUW1\_5925999500 | EUW1\_5926921682



Después de buscar nos encontraremos esta pantalla compuesta por un carrusel de gráficos con la probabilidad de victoria del equipo azul y una timeline con eventos ocurridos en la partida del minuto 1 al 15.

En el carrusel las imágenes pasan solas con el paso del tiempo pero también hay un botón a cada lado para pasar la imagen.

## 10) Bibliografía

Para el desarrollo de este proyecto se ha utilizado:

La web oficial de sklearn

<https://scikit-learn.org/stable/>

Material usado en clase de kschoool

<https://kschool.myopenlms.net/>

Towards data science para documentación sobre los modelos y diversos problemas

<https://towardsdatascience.com/>

Stackoverflow, errores, documentación e información

<https://stackoverflow.com/>