

Aprendizaje Automático y Minería de Datos

Clasificador de Pokémons

Diego Baratto Valdivia

Leonor Cuesta Molinero

Contenidos

[1. Support Vector Machine (SVM): 2](#_Toc30518020)

[1.1. Descripción del Proyecto: 2](#_Toc30518021)

[1.2. Resultados Obtenidos: 2](#_Toc30518022)

[1.2.1. Clasificador de Legendarios 2](#_Toc30518023)

[1.2.1. Clasificador de tipos 5](#_Toc30518024)

[1.3. Conclusiones: 6](#_Toc30518025)

[1.3.1. Clasificador de Legendarios 6](#_Toc30518026)

[1.3.2. Clasificador de tipos 7](#_Toc30518027)

[2. Regresión logística: 8](#_Toc30518028)

[2.1. Descripción del Proyecto: 8](#_Toc30518029)

[2.2. Resultados Obtenidos: 8](#_Toc30518030)

[2.3. Conclusiones: 8](#_Toc30518031)

[3. Redes Neuronales: 8](#_Toc30518032)

[3.1. Descripción del Proyecto: 8](#_Toc30518033)

[3.1.1. Clasificador de Legendarios 8](#_Toc30518034)

[3.1.2. Clasificador de tipos 8](#_Toc30518035)

[3.2. Resultados Obtenidos: 8](#_Toc30518036)

[3.2.1. Clasificador de Legendarios 8](#_Toc30518037)

[3.2.2. Clasificador de tipos 12](#_Toc30518038)

[3.3. Conclusiones: 13](#_Toc30518039)

[3.3.1. Clasificador de legendarios 13](#_Toc30518040)

[3.3.2. Clasificador de tipos 14](#_Toc30518041)

[4. Referencias 14](#_Toc30518042)

# 1. Support Vector Machine (SVM):

## 1.1. Descripción del Proyecto:

Haciendo uso de las SVM, se ha creado un clasificador de pokémon legendarios en función de las características elegidas por el usuario.

Se ha tomado como referencia la práctica realizada en clase sobre SVM, modificando aquellos fragmentos de códigos necesarios para lograr el objetivo, por ejemplo, dividir los datos en tres grupos diferentes (entrenamiento, validación y testeo) para realizar *cross-validation* (evitando así el sobreajuste) y elección de los parámetros C y sigma.

Se ha añadido también soporte de introducción de datos del usuario, es decir, tras el entrenamiento de la SVM, es posible testear la solución incluyendo por consola nuevos pokémon que no se encuentren en el grupo de los datos utilizados para entrenar.

Además de la clasificación anterior, se ha tratado de realizar clasificación multi-clase de los diferentes tipos de pokémon, entre ellos agua, tierra, roca, entre otros, en base a las diferentes características proporcionadas en el conjunto de datos. Se ha hecho uso de la misma metodología que en el caso anterior, añadiendo c*ross-validation*, elección de los mejores parámetros C y sigma y adición de nuevas características polinomiales. Debido a las complicaciones a la hora de elegir las características más representativas para la clasificación, se ha utilizado el paquete de Python denominado como Boruta, el cual ha ayudado a discriminar aquellos datos más importantes a la hora de la clasificación, mediante los *Random Forest* (se establecen varios árboles de decisión y se elige el que mejor resultado ofrezca):

## 1.2. Resultados Obtenidos:

A continuación se incluyen los diferentes resultados obtenidos, así como imágenes de apoyo y explicaciones al pie de las mismas.

### 1.2.1. Clasificador de Legendarios

De las primeras pruebas que se realizaron fue con kernel lineal, útil si el número de atributos (n) es grande comparado con el número de ejemplos (m):

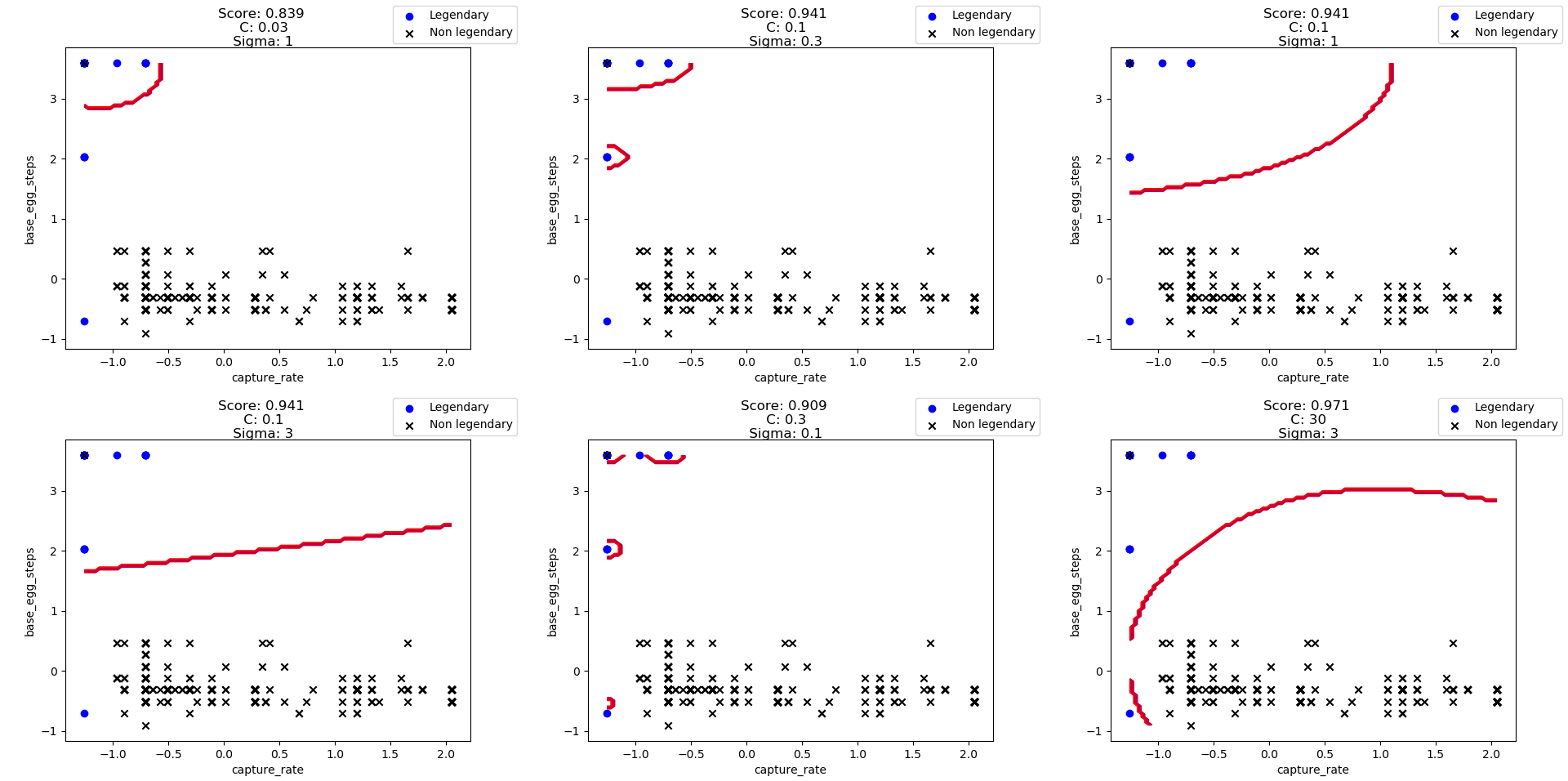
|  |  |
| --- | --- |
|  | Kernel lineal entrenado con los atributos *base egg steps* y *capture rate*. En este caso, al ser atributos diferenciados entre ellos, el kernel lineal funciona de manera aceptable. |
|  | Kernel lineal entrenado con los atributos *base happiness* y *attack*. En este caso, al ser atributos linealmente no diferenciables entre ellos, el kernel lineal no funciona de manera aceptable. |
| C:\Users\Usuario\AppData\Local\Microsoft\Windows\INetCache\Content.Word\lineal3.png | Kernel lineal entrenado con los atributos *speed* y *weight\_kg*. En este caso, al ser atributos linealmente no diferenciables, el kernel lineal no funciona de manera aceptable. |

En todos los casos con más de un atributo, el kernel lineal no funciona (*score* entre 0.4 y 0.6) debido a que el número de casos de entrenamiento es mayor que el número de atributos.

Tras estas pruebas, se decidió usar el kernel Gaussiano, apto para un número de atributos pequeño y un número de casos de ejemplo intermedio.

|  |  |
| --- | --- |
|  | Kernel Gaussiano entrenado con los atributos *base egg steps* y *capture rate*. |
|  | Kernel Gaussiano entrenado con los atributos *base happiness* y *attack*. |
|  | Kernel Gaussiano entrenado con los atributos *speed* y *weight\_kg*. |
|  | Kernel Gaussiano entrenado con los atributos *attack* y *defense*. |

Gracias a la visualización de estas gráficas, se puede observar cuales son los atributos que mejor definen a los pokémon legendarios y cuales ensucian el aprendizaje automático de la SVM. A continuación se muestra la elección de los parámetros C y sigma con los atributos que mejor definen a los legendarios, *capture rate* y *base egg steps*:



Se puede observar como varia la frontera de decisión de la SVM dependiendo de los valores que reciba C y sigma. Se aprecia de manera detallada el sobreajuste que se da en, por ejemplo, con C = 0.3 y sigma = 0.1.

### 1.2.1. Clasificador de tipos

A pesar de haber hecho uso de Boruta para elegir las características más importantes para la clasificación, no se consigue una media armónica mayor a 0.4 en cada tipo. Dentro de los mismos, se observa mucha diferencia de precisión en la predicción entre ellos.

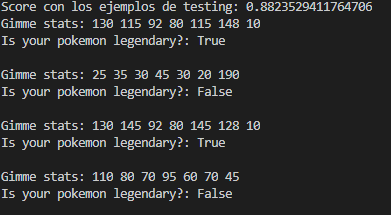
A continuación se muestran las gráficas obtenidas más significativas con diferentes atributos y tipos:

|  |  |
| --- | --- |
|  | Se puede observar el intento de clasificación del tipo planta a partir de las características ataque y vida. Viendo la gráfica y los resultados obtenidos, se puede deducir que no son atributos que se puedan usar para separar por tipo. |
|  | Mismo resultado que la gráfica anterior para el tipo agua. |
|  | Otro intento de clasificación del tipo bicho, esta vez con las características de ataque especial y velocidad. |
|  | Misma clasificación que la gráfica anterior para el tipo normal. A partir de los resultados obtenidos dependiendo de las características referidas, se deduce que algunos atributos clasifican mejor a cierto tipos que a otros. |
|  | Clasificación de tipo siniestro a partir de dos de sus debilidades. Es una forma muy precisa de clasificación pero no es la más acertada ya que prácticamente se le ofrece una descripción detallada del tipo al que pertenece. |

## 1.3. Conclusiones:

### 1.3.1. Clasificador de Legendarios

La técnica empleada con las SVM funciona de manera precisa y evita el *overfitting* / underfitting correctamente. Tras proporcionarle a los resultados de entrenamiento diferentes pokémon nunca vistos por ella, se logran unos resultados acertados y precisos, además de usar un conjunto de testeo como se ha dicho anteriormente para obtener el *score* del mismo (media armónica), logrando una media de 0.85 en dicho *score*. A continuación, se proporciona una captura de pantalla donde se consulta los datos (*attack*, *defense*, *hp*, *sp\_attack*, *sp\_defense*, *speed*, *capture\_rate*) de cuatro pokémon diferentes de la octava generación (Zacian, Snom, Zamazenta, Flapple), la cual no se encuentra en el *dataset*, siendo el primero y el tercero legendarios, mientras que el segundo y el cuarto no.



### 1.3.2. Clasificador de tipos

Como ya ha sido mencionado anteriormente, se ha conseguido discriminar aquellas características que mejor clasifican por tipo de pokémon. A pesar de ello, no se consiguen resultados equilibrados entre ejecuciones y con una gran diferencia de precisión obtenida en el clasificador de Legendarios.

Con todo ello, se puede concluir que ciertos tipos de pokémon se pueden clasificar de manera más o menos óptima a partir de, por ejemplo, el ataque especial, como el tipo psíquico, pero no es una generalización en todos los tipos. Incluso ciertos tipos son muy difíciles de clasificar siguiendo el algoritmo desarrollado ya que en los ejemplos de entrenamiento o bien hay muy pocos pokémon de ese tipo, por ejemplo de tipo dragón, o bien no existe ninguno con ese tipo primario, refiriéndose al caso de tipo volador. Por lo tanto, teniendo en cuenta solamente el *dataset* elegido, se obtienen resultados con falta de precisión y credibilidad. Para mejorar dichos resultados, habría que contar con el apoyo de otro *dataset*, como cualquiera que contenga imágenes de los pokémon, facilitando la tarea ya que se cuenta con datos de la paleta de colores y de la forma, atributos que son más significativos a la hora de clasificarlos por tipo.

# 2. Regresión logística:

## 2.1. Descripción del Proyecto:

## 2.2. Resultados Obtenidos:

## 2.3. Conclusiones:

# 3. Redes Neuronales:

## 3.1. Descripción del Proyecto:

Utilizando redes neuronales, hemos creado tanto un clasificador de legendarios en función de sus otras características como un predictor del tipo de Pokémon que es en función de su relación con otros tipos y características.

### 3.1.1. Clasificador de Legendarios

Tomando como referencia distintas características de entre *[attack, base\_egg\_steps, base\_happiness, base\_total, capture\_rate, defense, experience\_growth, height\_m, hp, percentage\_male, sp\_attack, sp\_defense, weight\_kg]*.

Se ha utilizado como referencia la práctica de laboratorio basada en redes neuronales, así como el módulo de Python “Keras” con “Tensorflow”.

En este clasificador, entrena una red neuronal dividida en grupos de entrenamiento, validación y testeo, cogiendo ejemplos del grupo total de forma aleatoria y barajándose para evitar que los legendarios y no legendarios se agrupen al principio o final de los grupos.

Para tratar de asegurar el mejor *score* de entrenamiento, se repite el proceso un número establecido de veces y guarda el mejor.

### 3.1.2. Clasificador de tipos

Tomando como referencia los “stats de batalla” mencionados antes, no se ha encontrado una forma fiable de separar los pokémon por tipos, sin embargo, al igual que en SVM, al utilizar los atributos de “against\_X” el tipo se clasifica con una seguridad alta.

Se ha utilizado la práctica de laboratorio de redes neuronales, al igual que en la clasificación por legendarios, así como Boruta para distinguir las características más relevantes ya que no conseguíamos una combinación que los clasificara correctamente.

## 3.2. Resultados Obtenidos:

### 3.2.1. Clasificador de Legendarios

En las primeras pruebas se utilizó el grupo de pokémons completo, habiendo una cantidad de entorno 70 legendarios de un total de 802 pokémons, por lo que clasificaba a todos ellos como pokémons normales, que daba una precisión del 91% (que variaba ligeramente con los pesos aleatorios de theta, cayendo en mínimos locales) pero un score de 0.

Como primera intención, se probaron algunos atributos por separado, destacando el “*capture\_rate”* con una precisión constante del 97.253%, pero aun clasificaba gran parte de los legendarios como pokémons normales.

Decidimos calcular el score del proceso para que se tenga en cuenta el recall y la precisión, ya que solo la precisión aportaba resultados engañosos, así como separar en grupos de entrenamiento (1/4 de normales, ½ de legendarios), validación (1/2 nomales, ¼ legendarios) y testeo (1/4 nomales, ¼ legendarios). Esto dio mejores resultados en cuanto al score en los ejemplos de entrenamiento (0.933), aunque bajaba en los ejemplos no entrenados (0.799). Seguimos cayendo en mínimos locales que enturbiaban los pesos de theta.

Determinamos que la forma de evitarlos era barajar los grupos para evitar la “condensación” de legendarios, añadido a ejecutar el proceso un número finito de veces y guardar el resultado mejor. Con ello, el score subió a 0.914 constante con todos los atributos, sorteando los mínimos locales en los que se caía anteriormente.

A partir de este momento, las pruebas se centraron en variar los atributos observados, la cantidad de intentos y el valor de lambda.

|  |  |
| --- | --- |
|  | *(Capture rate, Base egg steps) lambda = 1*  Se puede ver que son unas características que separan claramente los legendarios y no legendarios, teniendo éstos un nivel muy “agrupado” de estas variables. |
|  | *(special attack, height) lambda = 1*  Aquí se aprecia como los valores de los atributos están mucho más igualados entre legendarios y no legendarios, por lo que se clasifican mucho peor. |
|  | *(weight, height) lambda = 1*  Esta combinación parecía buena a priori debido al gran tamaño gráfico de los distintos pokémons legendarios y la baja media de los normales, pero como se aprecia, algunos normales “inflaban” los valores. |
|  | *(base egg steps, height) lambda = 1*  Hemos comprobado que al añadir los atributos *“base egg steps”* o “*capture rate”* los otros atributos se agrupan con ellos y el score aumenta independientemente de los otros atributos. |
|  | *(height, special attack) lambda = 1*  Polinomio grado 10  Existe sobreajuste a los ejemplos de entrenamiento. |

Utilizando los “stats de batalla” y lambda = 1 se obtuvieron scores que iban de 0.940 a 1.0, pero se halló que los atributos más significativos son *“capture\_rate”* y *“base\_egg\_steps”* con lambda = 1, con un score del 0.973.

Al continuar apareciendo mínimos locales en ocasiones, normalizamos los atributos, bajando el score a 0.941 en función de los atributos, pero sorteando mayor cantidad de mínimos locales.

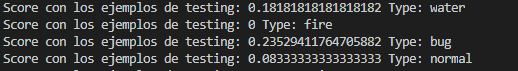
Por último, incluimos Keras para el entrenamiento y pintado del *decision boundary*.

|  |
| --- |
|  |
| *(Attack, defense) 400 vueltas*  Se puede apreciar la evolución del *decision boundary* a medida que aumentan las vueltas hasta estabilizarse. |
|  |
| *(base total, capture rate) 400 vueltas*  Como se puede observar, al añadir el atributo *capture rate*, la gráfica se estabiliza rápidamente y se mantiene muy similar el resto de vueltas. |
|  |
| *(width, height) 400 vueltas*  Se puede apreciar que con algunos valores, a pesar de acabar hallando un *decision boundary* bastante preciso, la precisión sufre grandes saltos de una vuelta a otra, llegando a estabilizarse entorno al 0.95 pero dependiendo mucho de la vuelta en la que se finalice. |

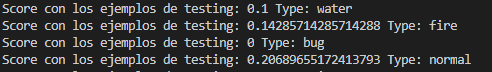
### 3.2.2. Clasificador de tipos

La clasificación de los pokémon por tipo con redes neuronales obtiene un resultado muy similar al de las SVM. Haciendo uso de Boruta, se establece que las características más relevantes son *percentage\_male* y *sp\_attack*. A pesar de ello, se obtienen resultados poco fiables y equilibrados entre tipos. Por ejemplo, se ha observado que ciertos tipos se clasifican mejor respecto a cierto tipo de atributos, mientras que otros no devuelven un resultado igual de óptimo que los demás.

A todo lo anterior, se ha de añadir que ciertos tipos, por ejemplo veneno y dragón, tienen un número muy pequeño de ejemplos de entrenamiento como para poder clasificarlos de manera correcta, por lo tanto, usando solamente los datos de los que disponemos, se obtiene un modelo con un error muy alto y, por lo tanto, poco fiable.



Atendiendo a la imagen, se puede observar la diferencia de resultados entre los diferentes tipos. Con ello, se puede decir que los atributos seleccionados (*percentage\_male* y *sp\_attack*) clasifican mejor al tipo bicho y que no sirven para el tipo fuego. Son resultados poco equilibrados y con un error elevado.



Con esta comparativa de resultados observando otras características en los mismos tipos (*attack* y *defense*), se confirma la hipótesis anterior, ya que los resultados para el tipo fuego y normal son mejores que en el caso anterior, mientras que el tipo bicho se clasifica erróneamente.

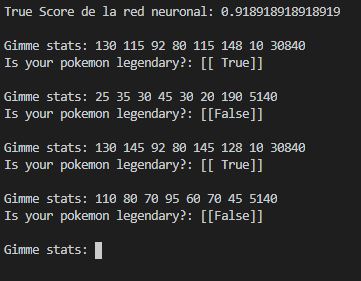
## 3.3. Conclusiones:

### 3.3.1. Clasificador de legendarios

Haciendo uso de la práctica realizada en clase, se ha conseguido evitar los mínimos locales en los que se estaba cayendo, además de realizar pruebas con diferentes valores de lambda para que la red neuronal no sufriera un *underfitting* / *overfitting* sobre los ejemplos de entrenamiento.

Tras solucionar los problemas anteriores, se concluyó que ciertas características enturbiaban los resultados y se fueron desechando hasta lograr resultados fiables y equilibrados entre diferentes ejecuciones del programa. Atributos como *base\_egg\_steps*, *capture\_rate*, *sp\_attack*, *height\_m* e incluso *base\_happiness* son relevantes a la hora de clasificar legendarios. Se puede concluir que, eligiendo los atributos más importantes y haciendo uso de ejemplos de testeo y ejemplos introducidos por el usuario, la red neuronal consigue catalogar de manera correcta un alto porcentaje de pokémon.

A continuación, se proporciona una captura de pantalla donde se consulta los datos (*attack*, *defense*, *hp*, *sp\_attack*, *sp\_defense*, *speed*, *capture\_rate*) de los mismos pokémon usados anteriormente de la octava generación (Zacian, Snom, Zamazenta, Flapple), además de proporcionar el *score* de la red neuronal, siendo este un 0.92.



### 3.3.2. Clasificador de tipos

Como ya se ha observado anteriormente en las otras técnicas utilizadas, la clasificación de tipos es complicada atendiendo solamente al conjunto de datos del que se dispone. Se llega a la misma conclusión, son resultados poco fiables y variables entre los diferentes tipos y sus características, además de no conseguir superar la media armónica de 0.4.

Se han consultado diferentes referencias de otros usuarios de Kaggle que han tratado de realizar dicha clasificación a través de redes neuronales, obteniendo resultados parecidos a los explicados anteriormente. Por lo tanto, se concluye que, a pesar de realizar correctamente el algoritmo de clasificación y de elección de parámetros más relevantes, el conjunto de datos disponible no se puede separar de manera precisa y acertada, en cuyo caso, la opción más acertada se basaría en aumentar el número de atributos, por ejemplo, una imagen de cada pokémon y realizar la clasificación a través de las mismas.

# 4. Referencias

* Jody Tubi (2018). Forecasting Pokémon Type [online]. Disponible en: <https://www.kaggle.com/jtubiunich/forecasting-pok-mon-type> [Accedido el 18 enero 2020].
* Rounak Banik (2018). The Complete Pokemon Dataset [online]. Disponible en: <https://www.kaggle.com/rounakbanik/pokemon> [Accedido el 12 diciembre 2019].
* Scikit Learn Contrib (2019). Boruta\_py [online]. Disponible en: <https://github.com/scikit-learn-contrib/boruta\_py> [Accedido el 19 enero 2020].