

# Informe proyecto final: Clasificador de Pokémon

## Elaborado por:

Juan Esteban Campos Avellaneda

Fecha de Entrega: 27 de noviembre del 2022

Presentado a: Francisco Carlos Calderón Bocanegra

Inteligencia Artificial
Departamento de Electrónica
Facultad de Ingeniería
Bogotá D.C.
2022-3

### Introducción

En la serie de Pokémon, en toda la historia han existido más de 800 Pokémon. Estos tienen diferentes estadísticas, tipos, generaciones, entre otros que permiten clasificar a los Pokémon. Adicionalmente, existen unos tipos que son más raros y exclusivos que otros, los cuales son los Legendarios. Estos son unos tipos de Pokémon que son más fuertes, más rápidos y en general que cuentan con mejores estadísticas que los demás.

Para el proyecto final de inteligencia artificial, lo que se busca es, buscar un método de ML que permita realizar un clasificador de los Pokémon legendarios. Para hacer esto, primero se necesita obtener un dataset. En Kaggle, se encontró el siguiente dataset: https://www.kaggle.com/datasets/abcsds/pokemon, el cual contiene diferentes estadísticas y categorías que permiten trabajar para realizar el clasificador.

En el desarrollo del proyecto, se implementaron 3 diferentes métodos que permitían hacer el clasificador se midieron con las respectivas métricas, y finalmente se sacaron las conclusiones.

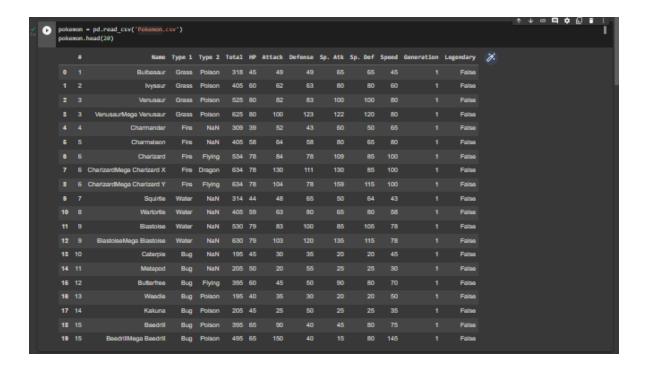
### Desarrollo

Para el desarrollo del proyecto, se utilizó la plataforma de Google Colaboratory, y se implementó en el lenguaje de programación Python. El código está estructurado de la siguiente manera:

Lo primero que se hace es hacer el llamado de las librerías, las que se usan son Pandas, para el tratamiento de datos, Numpy, también para el tratamiento de los datos, y finalmente Sklearn, para la implementación de los algoritmos de Machine Learning.

```
[38] import numpy as np
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
```

La siguiente parte del código está destinada al tratamiento de los datos, pues se debe hacer un preprocesamiento de datos para poder tratarlos de manera correcta.



De graficar los datos se puede ver que hay columnas que sobran, como la de "#", y hay columnas que contienen muchos NaN, correspondientes a los Pokémon que no tienen segundo tipo. Aquí se hace un procesamiento para eliminar estas columnas y los NaN.

```
[ii] pokemon = pokemon.drop(['#'], axis=1)

#Opción 1

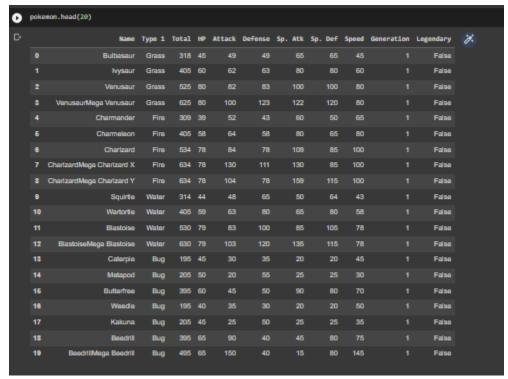
#pokemon['Type 2'].fillna(pokemon['Type 1'], implace=True)

#Opción 2

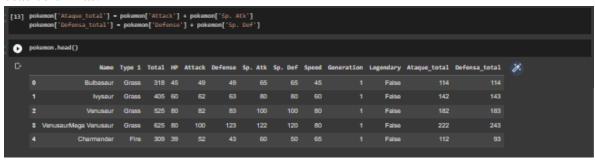
pokemon = pokemon.drop(['Type 2'], axis=1)
```

Se tenían dos opciones, la primera reemplazar los valores de la columna 2 con los de la columna 1, sin embargo, se optó por eliminar la columna, pues llenando la columna no se ganaba nada, y quitándola se ahorraba memoria.

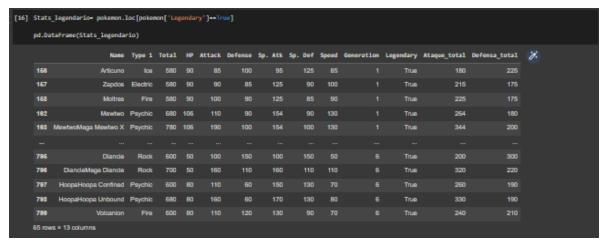
El resultado fue el siguiente:



Ahora, se deben escoger las características para el clasificador, aquí se analiza el dataset, se pueden ver dos cosas, la primera, que hay una columna llamada total que da las estadísticas totales del Pokémon, y la segunda que hay dos tipos de ataque y de defensa, la normal y la especial, esto se puede condensar en un solo valor de ataque y defensa, entonces se agregan esas columnas.



Ahora, lo que se hizo fue graficar los Pokémon legendarios, para hacer un pequeño análisis de las estadísticas y saber con cuáles haces los clasificadores.



Acá, se puede ver la cantidad de Pokémon legendarios y que las estadísticas son relativamente más altas que la de los Pokémon normales. Así que se crea un vector con las características importantes.

```
| [30] #Se crea un vector que contiene las estadiísticas principales
| caract- pokemon[['Total', 'Ataque_total', 'Defensa_total', 'HP']].to_numpy()
```

Ahora, ya con los datos tratados, se procede a hacer los conjuntos de prueba y entrenamiento:

```
Se crean los vectores de prueba y de entrenamiento

[31] X_train, X_test, y_train, y_test = train_test_split(caract, pokemon['Legendary'], random_state=0)
```

Y se hace la prueba con tres tipos de algoritmos diferentes:

### Por KNN:

### Por Árbol de decisiones:

```
[46] X_train, X_test, y_train, y_test = train_test_split(caract, pokemon['Legendary'], stratify=pokemon['Legendary'], random_state=42)

tree = DecisionTreeClassifier(max_depth=5, random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))

pred = tree.predict(X_test)
print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))
```

Por SVM:

```
Implementación por SVM

from sklearn import svm

clf = svm.SVC(kernel='linear')

clf.fit(X_train, y_train)

pred = clf.predict(X_test)

print("Accuracy on training set: (:.3f)".format(clf.score(X_train, y_train)))

print("Accuracy on test set: (:.3f)".format(clf.score(X_test, y_test)))

print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))
```

Y se comparan los diferentes métodos de ML en la sección de resultados.

### Resultados

## KNN:

## Árbol de decisiones:

**SVM:** 

```
from sklearn import svm

clf - svm.SVC(karnel='linear')

clf.fit(X_train, y_train)

pred = clf.predict(X_test)

print("Accuracy on training swt: (:.3f)".format(clf.score(X_train, y_train)))

print("Accuracy on test swt: (:.3f)".format(clf.score(X_test, y_test)))

print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))

C. Accuracy on training set: 0.943

Accuracy on training set: 0.943

Accuracy on test set: 0.935

[[38] 1]

[12] 4]]

precision recall fi-score support

False 0.94 0.99 0.97 184

True 0.80 0.25 0.38 16

accuracy

macro avg 0.87 0.62 0.67 200

weighted avg 0.93 0.94 0.92 200
```

En general, la manera en que se hizo la comparación de métodos en con el score que sklearn permite obtener de cada método, en el test set, el que mejor puntuación obtuvo fue el de árbol de decisiones, pues fue de 0.96, sin embargo, este resultado se dio debido a que se tuvo que implementar un límite de profundidad de 5, ya que se corre el riesgo de sobre entrenar el modelo. También, este fue el que, según la matriz de confusión fue el que menos falsos positivos y falsos negativos obtuvo.

En general, el dataset no era tan grande, por lo que todos los métodos corrían rápido.

## **Conclusiones**

- Durante el desarrollo del proyecto, se hicieron varias pruebas con los diferentes características, sin embargo, las que fueron relevantes fueron las que se escogieron, pues al utilizar las demás, los puntajes bajaban o no incrementaban de manera significativa.
- Para el método de KNN, se obtuvo una matriz de confusión, en donde el score para los verdaderos positivos fue muy alto, pues no presentó sino un falso positivo, sin embargo, para los falsos negativos, no fue tan exitoso, pues se presentó la misma cantidad de verdaderos falsos y de falsos negativos. Esto hace que el método no sea tan efectivo, y como se ve en el classification report, el score para los que no son legendarios es alto, pero para los legendarios es bajo.
- Para el método de árbol de decisiones, se obtuvo una matriz de confusión casi perfecta, pues tanto para falsos positivos como falsos negativos se obtuvieron nada más 4. Esto hace que el score para los que sí son legendarios sea más alto, de 0.75. Esto hace que este método sea el mejor y más efectivo.
- Finalmente, para el método de SVM, que fue el peor, lo que hizo fue que el score para los no legendarios fuera casi perfecto, mientras que para los que no eran legendarios fuera extremadamente malo, esto puede ser porque el modelo está sobre entrenado.