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.

```
pokemon = pd.read_csv('Pokemon.csv')
pokemon.head(20)
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
5	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
9	7	Squirtle	Water	NaN	314	44	48	65	50	64	43	1	False
10	8	Wartortle	Water	NaN	405	59	63	80	65	80	58	1	False
11	9	Blastoise	Water	NaN	530	79	83	100	85	105	78	1	False
12	9	BlastoiseMega Blastoise	Water	NaN	630	79	103	120	135	115	78	1	False
13	10	Caterpie	Bug	NaN	195	45	30	35	20	20	45	1	False
14	11	Metapod	Bug	NaN	205	50	20	55	25	25	30	1	False
15	12	Butterfree	Bug	Flying	395	60	45	50	90	80	70	1	False
16	13	Weedle	Bug	Poison	195	40	35	30	20	20	50	1	False
17	14	Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	False
18	15	Beedrill	Bug	Poison	395	65	90	40	45	80	75	1	False
19	15	BeedrillMega Beedrill	Bug	Poison	495	65	150	40	15	80	145	1	False

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.

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

     #Opción 1

     #pokemon['Type 2'].fillna(pokemon['Type 1'], inplace=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:

```
pokemon.head(20)
```

	Name	Type 1	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	Bulbasaur	Grass	318	45	49	49	65	65	45	1	False
1	Ivysaur	Grass	405	60	62	63	80	80	60	1	False
2	Venusaur	Grass	525	80	82	83	100	100	80	1	False
3	VenusaurMega Venusaur	Grass	625	80	100	123	122	120	80	1	False
4	Charmander	Fire	309	39	52	43	60	50	65	1	False
5	Charmeleon	Fire	405	58	64	58	80	65	80	1	False
6	Charizard	Fire	534	78	84	78	109	85	100	1	False
7	CharizardMega Charizard X	Fire	634	78	130	111	130	85	100	1	False
8	CharizardMega Charizard Y	Fire	634	78	104	78	159	115	100	1	False
9	Squirtle	Water	314	44	48	65	50	64	43	1	False
10	Wartortle	Water	405	59	63	80	65	80	58	1	False
11	Blastoise	Water	530	79	83	100	85	105	78	1	False
12	BlastoiseMega Blastoise	Water	630	79	103	120	135	115	78	1	False
13	Caterpie	Bug	195	45	30	35	20	20	45	1	False
14	Metapod	Bug	205	50	20	55	25	25	30	1	False
15	Butterfree	Bug	395	60	45	50	90	80	70	1	False
16	Weedle	Bug	195	40	35	30	20	20	50	1	False
17	Kakuna	Bug	205	45	25	50	25	25	35	1	False
18	Beedrill	Bug	395	65	90	40	45	80	75	1	False
19	BeedrillMega Beedrill	Bug	495	65	150	40	15	80	145	1	False

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.

```
[13] pokemon['Ataque_total'] = pokemon['Attack'] + pokemon['Sp. Atk']
     pokemon['Defensa_total'] = pokemon['Defense'] + pokemon['Sp. Def']
```

```
pokemon.head()
```

	Name	Type 1	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Ataque_total	Defensa_total
0	Bulbasaur	Grass	318	45	49	49	65	65	45	1	False	114	114
1	Ivysaur	Grass	405	60	62	63	80	80	60	1	False	142	143
2	Venusaur	Grass	525	80	82	83	100	100	80	1	False	182	183
3	VenusaurMega Venusaur	Grass	625	80	100	123	122	120	80	1	False	222	243
4	Charmander	Fire	309	39	52	43	60	50	65	1	False	112	93

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.

```
[16] Stats_legendario= pokemon.loc[pokemon['Legendary']==True]

     pd.DataFrame(Stats_legendario)
```

	Name	Type 1	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Ataque_total	Defensa_total
168	Articuno	Ice	580	90	85	100	95	125	85	1	True	180	225
167	Zapdos	Electric	580	90	90	85	125	90	100	1	True	215	175
163	Moltres	Fire	580	90	100	90	125	85	90	1	True	225	175
182	Mewtwo	Psychic	680	106	110	90	154	90	130	1	True	264	180
183	MewtwoMega Mewtwo X	Psychic	780	106	190	100	154	100	130	1	True	344	200
...	...	...	...	...	...	...	...	...	...	...	...	...	...
786	Diancie	Rock	600	50	100	150	100	150	50	6	True	200	300
788	DiancieMega Diancie	Rock	700	50	160	110	160	110	110	6	True	320	220
787	HoopaHoopa Confined	Psychic	600	80	110	60	150	130	70	6	True	260	190
788	HoopaHoopa Unbound	Psychic	680	80	160	60	170	130	80	6	True	330	190
780	Volcanion	Fire	600	80	110	120	130	90	70	6	True	240	210

65 rows × 13 columns

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 estadísticas 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:

```
Se hace la implementación por KNN

[32] knn = KNeighborsClassifier(n_neighbors=3)
     knn.fit(X_train, y_train)

     print('Accuracy of K-NN classifier on training set: {:.2f}'
          .format(knn.score(X_train, y_train)))
     print('Accuracy of K-NN classifier on test set: {:.2f}'
          .format(knn.score(X_test, y_test)))

     pred = knn.predict(X_test)
     print(confusion_matrix(y_test, pred))
     print(classification_report(y_test, pred))
```

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:

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

print("Accuracy of K-NN classifier on training set: {:.2f}"
     .format(knn.score(X_train, y_train)))
print("Accuracy of K-NN classifier on test set: {:.2f}"
     .format(knn.score(X_test, y_test)))

pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
Accuracy of K-NN classifier on training set: 0.96
Accuracy of K-NN classifier on test set: 0.95
[[181   1]
 [  9   9]]
              precision    recall  f1-score   support

       False       0.95      0.99      0.97       182
        True       0.90      0.50      0.64        18

    accuracy                           0.95       200
   macro avg       0.93      0.75      0.81       200
weighted avg       0.95      0.95      0.94       200
```

Árbol de decisiones:

Se hace la implementación por árbol de decisiones, se cambia el conjunto de pruebas y entrenamiento.

```
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))
```

```
Accuracy on training set: 0.978
Accuracy on test set: 0.960
[[180   4]
 [  4  12]]
              precision    recall  f1-score   support

       False       0.98      0.98      0.98       184
        True       0.75      0.75      0.75        16

    accuracy                           0.96       200
   macro avg       0.86      0.86      0.86       200
weighted avg       0.96      0.96      0.96       200
```

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))
```

```
Accuracy on training set: 0.943
Accuracy on test set: 0.935
[[183   1]
 [ 12   4]]
              precision    recall  f1-score   support

       False       0.94      0.99      0.97       184
        True       0.80      0.25      0.38        16

    accuracy                           0.94       200
   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.