Inteligencia Artificial
Act 11: Programando Regresión Logística en Python

1 Introducción

La regresión logística es un algoritmo de aprendizaje supervisado que se usa para problemas de clasificación, es muy útil para clasificar categorías de datos. Se pretende predecir qué probabilidad le corresponde a una instancia de pertenecer a una categoría dentro de una lista de posibles categorías.

La regresión logística es un modelo estadístico que estima la probabilidad de que ocurra un evento, en función de un conjunto de datos determinado de variables independientes, la variable dependiente está entre 0 y 1.

Este algoritmo trabaja con una variable dependiente binaria (0 o 1) y un conjunto de variables independientes, para calcular la probabilidad utiliza una función llamada sigmoide que transforma valores en un rango de 0 a 1. A partir de este entrenamiento el algoritmo pretende clasificar nuevas instancias de la mejor manera posible.

2 Metodología

Para llevar a cabo la actividad, se siguieron las indicaciones del libro *Aprende Machine Learning*, en la página 39. Primero, se creó una carpeta llamada "Regresión Logística" y se descargó el archivo de entrada CSV. En la carpeta se guardó el archivo CSV y se creó el archivo que va a contener el código Python, para realizar el código se hizo uso de Visual Studio Code.

Empezamos importando las librerías y paquetes necesarios:

```
import pandas as pd
import numpy as np
from sklearn import linear_model, model_selection
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score
import matplotlib.pyplot as plt
import seaborn as sb
```

Nota: Puede ocurrir el caso que se necesiten instalar varias librerías con pip, paea eso basta abrir la terminar y escribir *pip install nombre de la librería*, por ejemplo *pip install seaborn*.

Leemos el archivo CSV y lo cargamos como una dataframe de pandas

```
# carga y lectura del archivo CSV
dataframe = pd.read_csv(r"usuarios_win_mac_lin.csv")
```

Al cargar el archivo, podemos explorar los datos, como los primeros registros, las estadísticas de los datos y el tipo de datos que tenemos, para eso imprimimos:

```
print(dataframe.head()) # primeros registros
print("\n", dataframe.describe()) # estadisticas de los datos
print("\n", dataframe.groupby('clase').size()) # tipos de datos
```

Listing 1: Exploración de datos

Antes de empezar a procesar los datos, visualizamos rápidamente las características de entrada, que nos ayuda a comprender mejor las características de la información que tenemos, para hacerlo se elimina una columna ('clase') que no es relevante para el análisis futuro y generamos un histograma para cada una de las columnas restantes para posteriormente visualizar las gráficas.

```
# visualizacion de datos
dataframe.drop(['clase'], axis = 1).hist()
plt.show()
```

Listing 2: Visualización de datos

Además de visualizar los datos en histogramas podemos interrelacionar las entradas para ver como se concentran linealmente las salidas de usuarios por colores, donde Windows es azul, Macintosh es verde y Linux es rojo.

```
# interrelacionar para ver la relacion lineal
sb.pairplot(dataframe.dropna(), hue='clase', height=4, vars=["duracion", "
    paginas", "acciones", "valor"], kind="reg")
plt.show()
```

Listing 3: Visualización de la relacion de los datos

Ahora cargamos las variables en X excluyendo la variable clase que no es relevante para nosotros en este variable pero sí es relevante para la variable y, aparte vamos a comprobar la dimensión de la matriz con datos de entrada de 170 registros por 4 columnas.

```
X = np.array(dataframe.drop(['clase'],axis=1))
y = np.array(dataframe['clase'])
print("\n", X.shape) # dimension de la matriz
```

Listing 4: Dimensión de la matriz

Procedemos a crear el mejor de regresión logística, los ajustamos a nuestro conjunto de entradas y salidas y hacemos predicciones.

```
model = linear_model.LogisticRegression(max_iter=1000) # objeto de regresion
    logistica
model.fit(X,y) # ajustamos el modelo
predictions = model.predict(X) # predicciones
```

Una vez compilado el modelo, revisamos algunas de las salidas de las predicciones y verificamos si coinciden con las salidas reales del archivo CSV, también verificamos la precisión media de las predicciones para saber que tan bueno fue el modelo.

```
print("\n", predictions[0:5]) # revisamos algunas salidas
print("\n", model.score(X,y)) # revisamos la presicion media de las
    predicciones
```

Listing 5: Revisión del modelo

Volvemos a validar nuestro modelo, es bien sabido que una buena práctica es dividir nuestros datos, una parte en entrenamiento y otra parte de validación, en nuestro caso, vamos a dividir nuestros datos de entrada de forma aleatoria utilizando un 80% para entrenamiento y el otro 20% para validación.

```
# validamos el modelo dividiendo los datos 80, 20
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.
    train_test_split(X, y, test_size=validation_size, random_state=seed)
```

Ahora, volvemos a compilar el modelo de regresión logística, con la única diferencia que ahora usaremos el 80% de los datos, los cuales son los de entrenamiento y calculamos la precisión del modelo.

```
# compilamos el modelo pero solo con el 80 de los datos
name='Logistic Regression'
kfold = model_selection.KFold(n_splits=10, random_state=None)
cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold
    , scoring='accuracy')
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print("\n", msg) # calculamos nueva presicion media
```

Listing 6: Compilación con el 80% de los datos

Al terminar de entrenar el modelo, hacemos las predicciones (clasificamos), utilizando nuestro "cross validation set", es decir usando el conjunto que separamos previamente. Imprimimos los resultados de precisión y además un reporte de los resultados del modelo.

```
predictions = model.predict(X_validation) # predicciones c
print("\n", accuracy_score(Y_validation, predictions)) # presicion de los
    datos
print("\n", confusion_matrix(Y_validation, predictions)) # reporte de matriz
print("\n", classification_report(Y_validation, predictions)) # reporte de
    clasificacion
```

Listing 7: Resultados y reporte del modelo

Para terminar, vamos a probar el algoritmo que acabamos de crear, inventamos datos de entrada de un usuario ficticio que tiene los siguientes valores:

- Tiempo de duración: 10
- Páginas visitadas: 3
- Acciones al navegar: 5
- Valoración: 9

```
# clasificacion de nuevos valores
X_new = pd.DataFrame({'duracion': [10], 'paginas': [3], 'acciones': [5], '
    valor': [9]}) # nuevos valores
prediction = model.predict(X_new.values) # predecimos
print("\n", "Prediccion: ", prediction) # prediccion
```

Listing 8: Prediccion del algoritmo

3 Resultados

A continuación, se van a presentar los resultados obtenidos a partir del código mostrado.

3.1 Exploración de Datos

En esta parte (Listing 1) se realizó una exploración del dataset. Los resultados obtenidos fueron los siguientes:

- Los primeros 5 registros del dataset son:

	duracion	paginas	acciones	valor	clase
0	7.0	2	4	8	2
1	21.0	2	6	6	2
2	57.0	2	4	4	2
3	101.0	3	6	12	2
4	109.0	2	6	12	2

- Las estadísticas de los datos son:

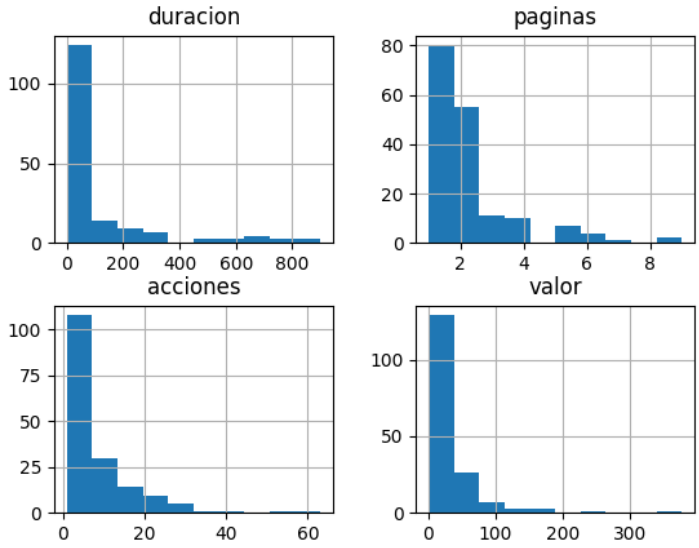
	duracion	paginas	acciones	valor	clase
count	170.000000	170.000000	170.000000	170.000000	170.000000
mean	111.075729	2.041176	8.723529	32.676471	0.752941
std	202.453200	1.500911	9.136054	44.751993	0.841327
min	1.000000	1.000000	1.000000	1.000000	0.000000
25%	11.000000	1.000000	3.000000	8.000000	0.000000
50%	13.000000	2.000000	6.000000	20.000000	0.000000
75%	108.000000	2.000000	10.000000	36.000000	2.000000
max	898.000000	9.000000	63.000000	378.000000	2.000000

- Los resultados de cada tipo de datos de cada clase son:

```
 clase
0    86
1    40
2    44
dtype: int64
```

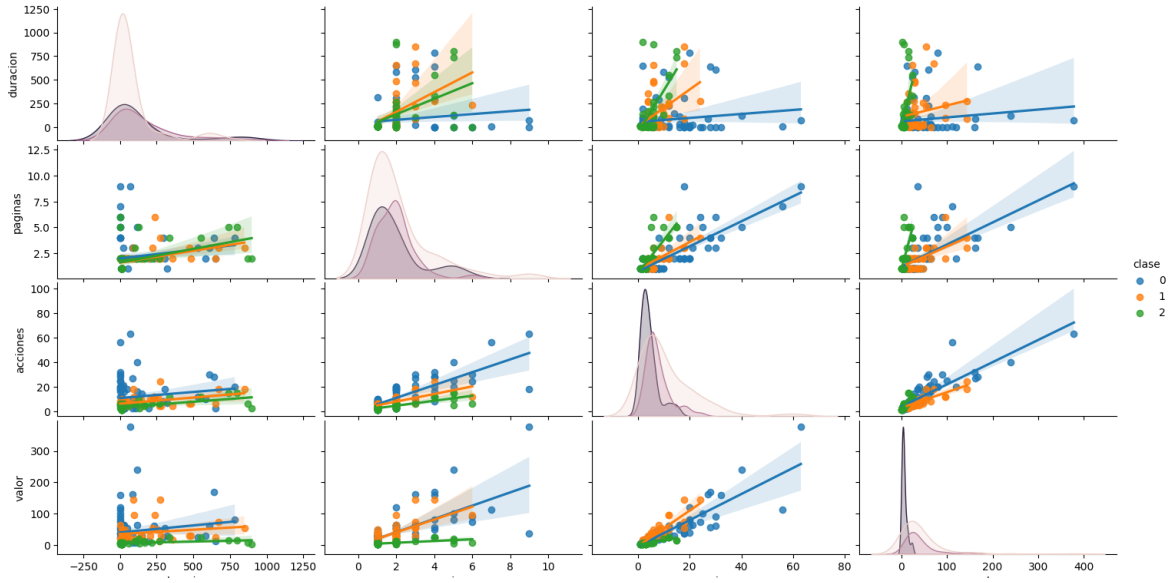
3.2 Visualización de Datos

Los histogramas generados en el Listing 2 son los siguientes:



3.3 Visualización de la relación de los Datos

Así se ve como se concentra linealmente las salidas de los usuarios (Listing 3):



3.4 Dimensión de la matriz

La dimensión de la matriz con 170 registros por 4 columnas es (Listing 4):

```
(170, 4)
```

3.5 Revisión del modelo

Las salidas para verificar si coinciden con nuestro archivo CSV son las siguientes (Listing 5):

```
[2 2 2 2 2]
```

Si checamos bien, podemos ver que sí coinciden con nuestro archivo. Entonces, la precisión media de las predicciones del modelo es de:

```
0.7764705882352941
```

3.6 Compilación con el 80% de los datos

El resultado de compilar con el 80% de los datos, que son los datos que separamos de forma aleatoria y etiquetamos como datos de entrenamiento (Listing 6), donde el primer valor es la media y el segundo la desviación estándar:

```
Logistic Regression: 0.728571 (0.094186)
```

3.7 Resultados y Reporte del modelo

En esta parte (Listing 7) se imprimieron resultados y reporte del modelo. La salida fue la siguiente:

- La precisión de los datos fue de:

```
0.8529411764705882
```

- El reporte de la matriz:

```
[[16  0  2]
 [ 3  3  0]
 [ 0  0 10]]
```

- El reporte de clasificación:

	precision	recall	f1-score	support
0	0.84	0.89	0.86	18
1	1.00	0.50	0.67	6
2	0.83	1.00	0.91	10
accuracy			0.85	34
macro avg	0.89	0.80	0.81	34
weighted avg	0.87	0.85	0.84	34

El 85% de los datos fueron acertados, no obstante el tamaño de los datos era muy pequeño. La matriz de confusión nos muestra cuantos resultados equivocados se tuvo de cada clase, por ejemplo predijo a 2 usuarios Linux cuando eran de Windows. En el reporte de clasificación podemos ver que se utilizar como "soporte" 18 registros windows, 6 de mac y 10 de linux, también podemos ver la precisión de cada una de las clases, el recall y F1-score.

3.8 Predicción del algoritmo

Finalmente (Listing 8) el resultado de inventar datos de entrada donde el usuario ficticio tiene un tiempo de duración de 10, 3 páginas visitadas, 5 acciones al navegar y una valoración de 9 es de:

```
Prediccion:  [2]
```

Nuestro modelo predijo que el usuario es de tipo 2, es decir es un usuario de Linux.

4 Conclusión

En esta actividad, utilizamos un archivo CSV para entrenar un modelo de regresión logística. Hicimos lo básico que hemos hecho en las actividades pasadas, que es leer el archivo y cargarlo como dataframe de pandas, para posteriormente explorar un poco los datos que tenemos y visualizar los datos en un histograma para comprender un poco mejorar las características de la información que tenemos, además realizamos gráficas para ver la relación lineal que existe. Posteriormente empezamos creando un modelo de regresión logística, lo entrenamos y se hicieron predicción, el porcentaje de predicción fue de un 77% donde no fue malo pero podía mejorar. De esta manera nos pasamos a validar de nuevo el modelo, pero dividiendo nuestros datos en un ratio 80/20 donde el 80% son de entrenamiento y los restantes de validación, entonces hicimos lo mismo, entrenamos el modelo con el .8 de los datos y lo validamos con el .2 de validación que nos dió una precisión del 85%, la cual es mejor que la pasada. Además imprimimos un reporte de nuestro modelo para tener un mejor entendimiento del modelo y realizamos una predicción.