

Entrega final de proyecto

Ariel Eduardo Bedoya Marín.
Identificación: 71275506
Mayo 2023.

Universidad de Antioquia.
Ingeniería de Sistemas.
Introducción a la inteligencia artificial para las ciencias e ingenierías.

1. INTRODUCCION

Resultados alcanzados

Con los análisis iniciales del dataset, se llega a entender que este posee información muy básica para realizar predicciones a cerca de qué géneros de videojuegos podrían ser potencialmente deseables para que las comercializadoras y/o editoras quieran iniciar un desarrollo. Sin embargo, se hace interesante la idea de abordar experimentos que permitan buscar relaciones o patrones de los géneros de videojuegos más vendidos en determinados años, para cuáles plataformas fueron lanzados y cuáles has sido sus empresas publicadoras.

De momento, uno de los mayores obstáculos para manipular el dataset es, irónicamente, su simpleza, debido a que contiene pocas columnas, es decir, hay pocas variables con las cuales buscar correlaciones que lleguen a generar un proyecto complejo. Lo anterior no quiere decir que no sea posible implementar modelos predictivos interesantes y complejos que permitan ver algún tipo de resultado sea positivo o no y así llegar a conclusiones y análisis.

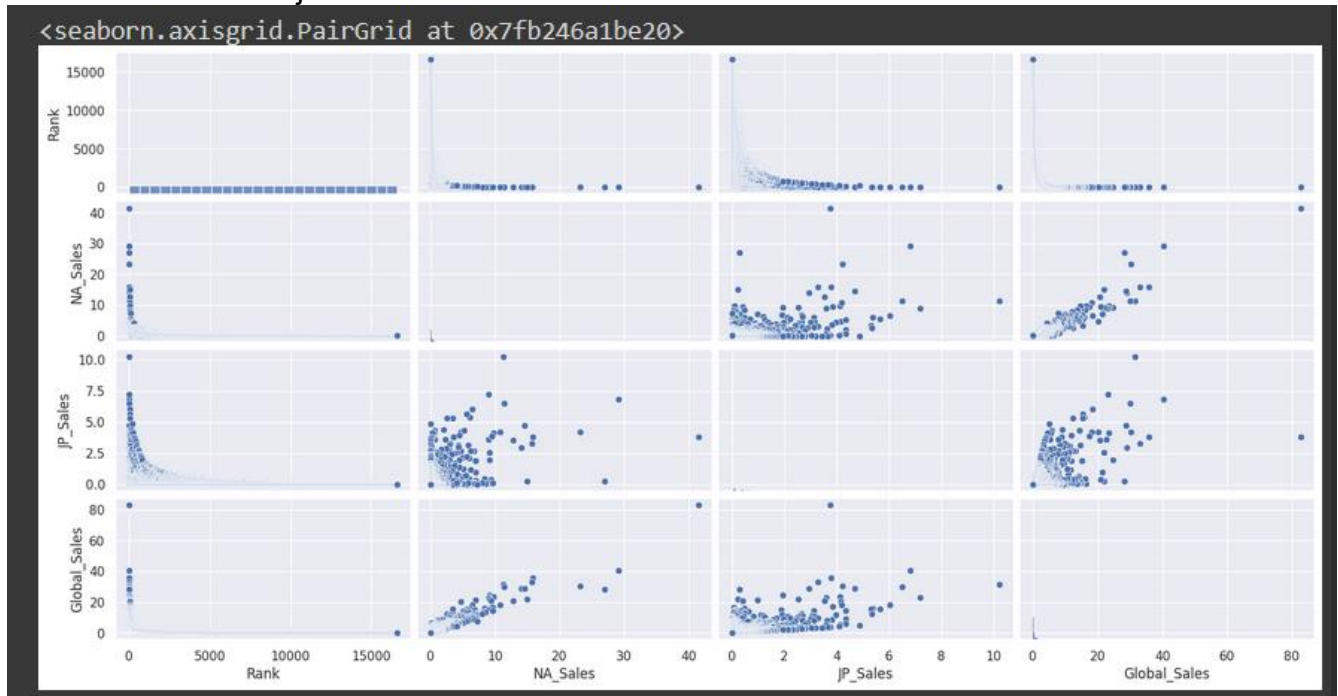
Ya dicho lo anterior, se puede decir que, al momento de hacer este informe, se ha realizado una exploración de los datos, incluyendo procesos para llamar el dataset desde kaggle (por medio de su API). Dentro de esta exploración se ha notado que hay años de publicación faltantes para algunos videojuegos y así mismo faltan algunos nombres de sus publicadores.

```
Rank          0
Platform      0
Year          271
Genre         0
Publisher     58
NA_Sales      0
EU_Sales      0
JP_Sales      0
Other_Sales   0
Global_Sales  0
dtype: int64
```

```
Year          271
Publisher     58
dtype: int64
```

2. EXPLORACION DEL DATASET

Se ha realizado un reemplazo del índice, el cual estaba numerado, por el nombre del videojuego. Esto debido a que el nombre es único y no proveerá mucha información dentro de las predicciones. Se verificó el tipo de datos, dentro de estos se observó que el año era de tipo 'Float64' y por lo tanto se convirtió a un tipo entero ('Int64'). Se verificaron los datos faltantes y también se contrastó a través de gráficas, las relaciones que podrían o no, tener las variables del conjunto de datos.



Los análisis aquí se hacen evidentes y además es el comportamiento natural, dado que en todos los casos el ranking de un juego es directamente proporcional al número de copias vendidas. También es evidente que el número de copias vendidas a nivel global es directamente proporcional al número de copias vendidas a nivel de regiones y/o países. Hay una pequeña excepción con Japón, en el cual se puede notar una mayor cantidad de copias vendidas respecto a otras regiones y a nivel global.

Debido a la naturaleza de los datos de las columnas "Year" y "Publisher" se ha decidido borrar las filas que no poseen estos datos ya que no es posible reemplazarlos por valores calculados. A continuación, se puede observar la cantidad de entradas que quedaron respecto a las que habían antes de la limpieza.

```

1 ### Limpieza de datos, solución a datos faltantes ###
2 #df.Year.fillna(df.Year.mean(), inplace=True)
3 dfn = df.dropna()
4 #dn = dn[[i for i in dn.columns if df[i].dtype!=object]]
5 print ("Dimensiones de dataframe original: ", df.shape)
6 print ("Dimensiones de dataframe nuevo: ", dfn.shape)
7 #dlr = dn.copy()
8 #aux = df["Year"].copy()
9 #aux[aux.isna()] = np.random.normal(loc=np.mean(aux), sca
10 #dlr["Year"] = aux
11 #dlr
12 dfn

```

Dimensiones de dataframe original: (16598, 10)
Dimensiones de dataframe nuevo: (16291, 10)

Seguidamente se verifica que ya no existan datos faltantes dentro del dataset.

```

Rank      0
Platform  0
Year      0
Genre     0
Publisher  0
NA_Sales  0
EU_Sales  0
JP_Sales  0
Other_Sales  0
Global_Sales  0
dtype: int64

```

El siguiente paso es convertir el tipo de dato de año en un entero ya que se encuentra en un formato de Float.

```

1 ### Conversión a tipo de dato entero de la columna "Year" y verificación ###
2 dfn['Year'] = dfn['Year'].astype(np.int64)
3 print(dfn.dtypes)
4 print(dfn['Year'].unique())
5 len(dfn['Year'].unique())

```

```

Rank      int64
Platform  object
Year      int64
Genre     object
Publisher  object
NA_Sales  float64
EU_Sales  float64
JP_Sales  float64
Other_Sales  float64
Global_Sales  float64
dtype: object

```

Ya con el dataset preparado se procede a generar un conjunto de datos con las variables dependientes y la variable independiente o de salida.

```

1 ### Separación de las variables dependientes y la variable independiente ###
2 x = dfn.drop(['Genre'], axis=1)
3 y = dfn['Genre'].values
4 print(x.head(5), type(x), '\n')
5 print(y, type(y))

```

Name	Rank	Platform	Year	Publisher	NA_Sales	EU_Sales	\
Wii Sports	1	Wii	2006	Nintendo	41.49	29.02	
Super Mario Bros.	2	NES	1985	Nintendo	29.08	3.58	
Mario Kart Wii	3	Wii	2008	Nintendo	15.85	12.88	
Wii Sports Resort	4	Wii	2009	Nintendo	15.75	11.01	
Pokemon Red/Pokemon Blue	5	GB	1996	Nintendo	11.27	8.89	

Name	JP_Sales	Other_Sales	Global_Sales
Wii Sports	3.77	8.46	82.74
Super Mario Bros.	6.81	0.77	40.24
Mario Kart Wii	3.79	3.31	35.82
Wii Sports Resort	3.28	2.96	33.00
Pokemon Red/Pokemon Blue	10.22	1.00	31.37

```

<class 'pandas.core.frame'>
['Sports' 'Platform' 'Racing' ... 'Racing' 'Puzzle' 'Platform'] <class 'numpy.ndarray'>

```

El siguiente paso ha sido el de codificar las variables categóricas. Es de resaltar que en este punto, no se debe olvidar que al etiquetar las variables categóricas con números, los algoritmos podrían dar un peso numérico, lo cual puede acarrear lecturas incorrectas.

```

1 ### Tratamiento de las variables categóricas ###
2 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
3 from sklearn.compose import ColumnTransformer
4 label_encoder_y = LabelEncoder()
5 label_encoder_x = LabelEncoder()
6 y[:, :] = label_encoder_y.fit_transform(y)
7 x.iloc[:, 1] = label_encoder_x.fit_transform(x.values[:, 1])
8 x.iloc[:, 3] = label_encoder_x.fit_transform(x.values[:, 3])
9 #ct = ColumnTransformer([("Publisher", OneHotEncoder(), [3])],
10 #x = ct.fit_transform(x)
11 print(type(y))
12 print(y.dtype)
13 #for i,x in enumerate(y_pred):
14 #    y_pred[i]=x.astype('int')
15 print(y, '\n')
16 y=y.astype('int')
17 print(type(y))
18 print(y.dtype)
19 print(np.unique(y))

```

```

<class 'numpy.ndarray'>
object
[10 4 6 ... 6 5 4]

<class 'numpy.ndarray'>
int64

```

Se procede a asignar los datos que serán para entrenamiento y los que serán para prueba del conjunto de datos

```

1 ### Selección de los datos de entrenamiento y los datos de prueba ###
2 from sklearn.model_selection import train_test_split, KFold
3 (x_train, x_test, y_train, y_test) = train_test_split(x, y)
4 print("Tamaño de dataset de variables dependientes: ", x.shape)
5 print("Tamaño de vector de variable independiente: ", y.shape)
6 print("Tamaño de datos de entrenamiento: ", x_train.shape)
7 print("Tamaño de datos de prueba: ", x_test.shape, '\n')
8 print("Salida de entrenamiento: ", y_train.shape)
9 print("Salida de prueba: ", y_test.shape)

```

```

Tamaño de dataset de variables dependientes: (16291, 9)
Tamaño de vector de variable independiente: (16291,)
Tamaño de datos de entrenamiento: (12218, 9)
Tamaño de datos de prueba: (4073, 9)

Salida de entrenamiento: (12218,)
Salida de prueba: (4073,)

```

Para los datos de entrenamiento se emplearán 12218 entradas del total de 16291. Las entradas restantes (4073), serán empleados como datos de test o prueba.

3. ITERACIONES

Los modelos con los que se trabajó fueron **árboles de decisión de clasificación y regresión logística**. Se intentó ejecutar un modelo de máquinas de soporte vectorial, pero esta presentó problemas al momento de realizar los ajustes por lo cual se descartó.

```

1 ### Selección de los modelos a emplear###
2 from sklearn import model_selection
3 from sklearn.preprocessing import PolynomialFeatures, StandardScaler
4 import matplotlib.pyplot as plt
5 from sklearn.tree import DecisionTreeClassifier
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.svm import SVC
8 p_score = lambda model, score: print('Performance of the %s model is %0.2f%%' % (model, score * 100))
9 sc = StandardScaler()
10 x_train_decitree = sc.fit_transform(x_train) #Ajuste de escalas
11 x_test_decitree = sc.transform(x_test) #Ajuste de escalas
12 y_train_decitree = y_train
13 ### Decision Tree Classifier
14 classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
15 classifier.fit(x_train_decitree, y_train_decitree)

```

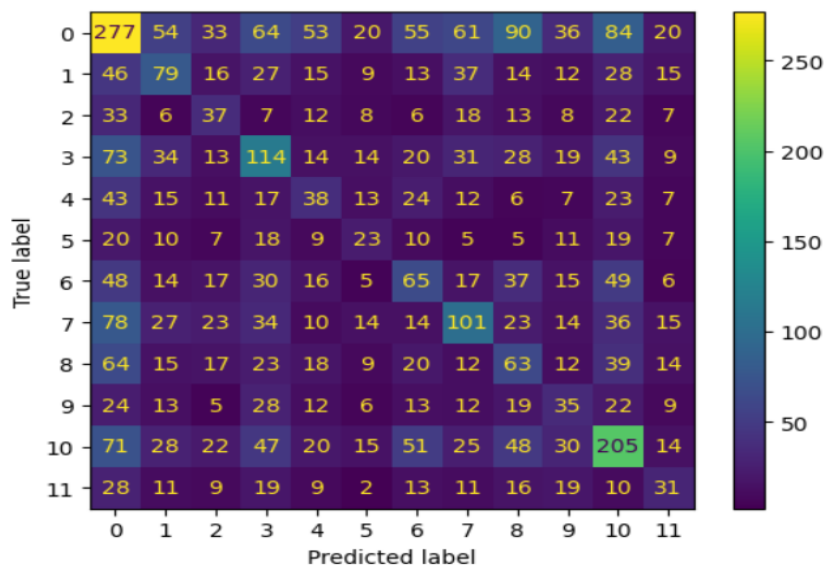
Se realiza el ajuste y la predicción con el árbol de decisión y se visualiza una matriz de confusión como métrica. Los resultados se pueden observar a continuación.

```

1 # Predicción
2 y_pred = classifier.predict(x_test_decitree)
3 # Matriz de confusión
4 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
5 import matplotlib.pyplot as plt
6 cm = confusion_matrix(y_test, y_pred)
7 print(cm.shape)
8 disp = ConfusionMatrixDisplay(confusion_matrix=cm)
9 disp.plot()

```

(12, 12)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9c2e2ae470>



Después hacer el análisis, se concluye que la cantidad de aciertos es baja en este modelo, siendo un total de aciertos de 1068 elementos del total de 4073 del conjunto de datos de salida. Por lo tanto, se observa que tan solo el 26.22% de los datos han sido verdaderos positivos, así que se puede decir que es un porcentaje bastante bajo de acierto. Se puede observar en la matriz que los géneros que mayor cantidad de aciertos tuvieron fueron "Action" y "Sports". El género y su codificación se puede observar a continuación.

```

1 ### No se logró decodificar el arreglo de salida por lo que se hizo
2 ### un diccionario para relacionar los géneros de videojuegos ###
3 y_n=label_encoder_y.inverse_transform(np.unique(y))
4 genero = {
5     "0": "Action",
6     "1": "Adventure",
7     "2": "Fighting",
8     "3": "Misc",
9     "4": "Platform",
10    "5": "Puzzle",
11    "6": "Racing",
12    "7": "Role-Playing",
13    "8": "Shooter",
14    "9": "Simulation",
15    "10": "Sports",
16    "11": "Strategy"
17 }
18 print(cm.shape)
19 print(type(cm))
20 diag=cm.sum()
21 print(diag)
22 true_positive = np.trace(cm)
23 print(true_positive)
24 porcentaje = (true_positive*100)/diag
25 print(porcentaje)

(12, 12)
<class 'numpy.ndarray'>
4073
1068
26.22145838448318

```

Para el modelo de regresión logística, se realiza el ajuste del modelo con los datos de entrenamiento al igual que con el modelo anterior y se realiza una validación cruzada K-fold. Estos son los resultados a continuación.

```

1 ### Regresión logística ###
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import accuracy_score
4 x_train_lr = sc.fit_transform(x_train) #Ajuste de escalas
5 x_test_lr = sc.transform(x_test)#Ajuste de escalas
6 y_train_lr = y_train
7 ### Clasificadores ###
8 model1 = LogisticRegression()
9 #model1 = SVC(kernel="linear", C=0.025)
10 #model2 = SVC(gamma=2, C=1)
11 #scores = cross_val_score(model1, x_train, y_train, cv=3)
12 model1.fit(x_train_lr, y_train_lr)
13 name='Logistic Regression'
14 kfold = KFold(n_splits=10, random_state=None)
15 cv_results = model_selection.cross_val_score(model1, x_train_lr, y_train_lr, cv=kfold, scoring='accuracy')
16 score = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
17 print(score)
18 y_pred_rl = model1.predict(x_test_lr)
19 print(accuracy_score(y_test, y_pred_rl))

Logistic Regression: 0.224095 (0.012199)
0.22759636631475572
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```


CONCLUSIONES

Como se puede observar, los resultados de los puntajes tanto de la media y la desviación estándar de la validación cruzada y de la precisión del modelo, no son muy buenos siendo estos de 0.224, 0.012 y 0.227 respectivamente.

Como conclusión se puede decir que se requiere el uso de otros diferentes modelos y de mas pruebas de ensayo y error con diversos parámetros de los modelos usados y otros modelos a implementar.

Se pudo observar que los resultados de acierto en ambos modelos no fueron satisfactorios y se puede decir que esto puede ser debido a que el conjunto de datos debió ser tratado de otra manera, como agregar más variables por medio de convertir una de las características en varias variables o simplemente buscar otros modelos y/o hiper parámetros que se ajusten mejor al conjunto de datos.