Clasificador de aplicaciones Play Store

Cristian Stivens Villarreal Parra - 2204132

Joan Sebastian Patiño Jaimes - 2202052

Jorge Eduardo Suarez Cortes - 2205561

Visión general

El proyecto aborda la Clasificación de aplicaciones de Android en la Play Store mediante el uso de técnicas de inteligencia artificial. Siguiendo directrices establecidas y buscando la optimización del proceso de clasificación con el fin de poder hallar las aplicaciones mas relevantes del dataset empleado, hemos aplicado una combinación de métodos, incluyendo una red neuronal secuencial, el algoritmo de Análisis de Componentes Principales (PCA), y la exploración de técnicas de clustering como K-Means.

El conjunto de datos de Google Play Store Apps es un conjunto de datos web que contiene información sobre 10,841 aplicaciones disponibles en la tienda de aplicaciones de Google Play

Google Play Store Apps | Kaggle

Objetivo del proyecto

El objetivo del proyecto es diseñar un modelo de clasificación que nos permita hallar la aplicación más popular en el conjunto de datos según diferentes variables como, su categoría, género, tipo, instalaciones, etc. Una vez entrenado, el clasificador podrá identificar la mejor aplicación en función de los criterios seleccionados.

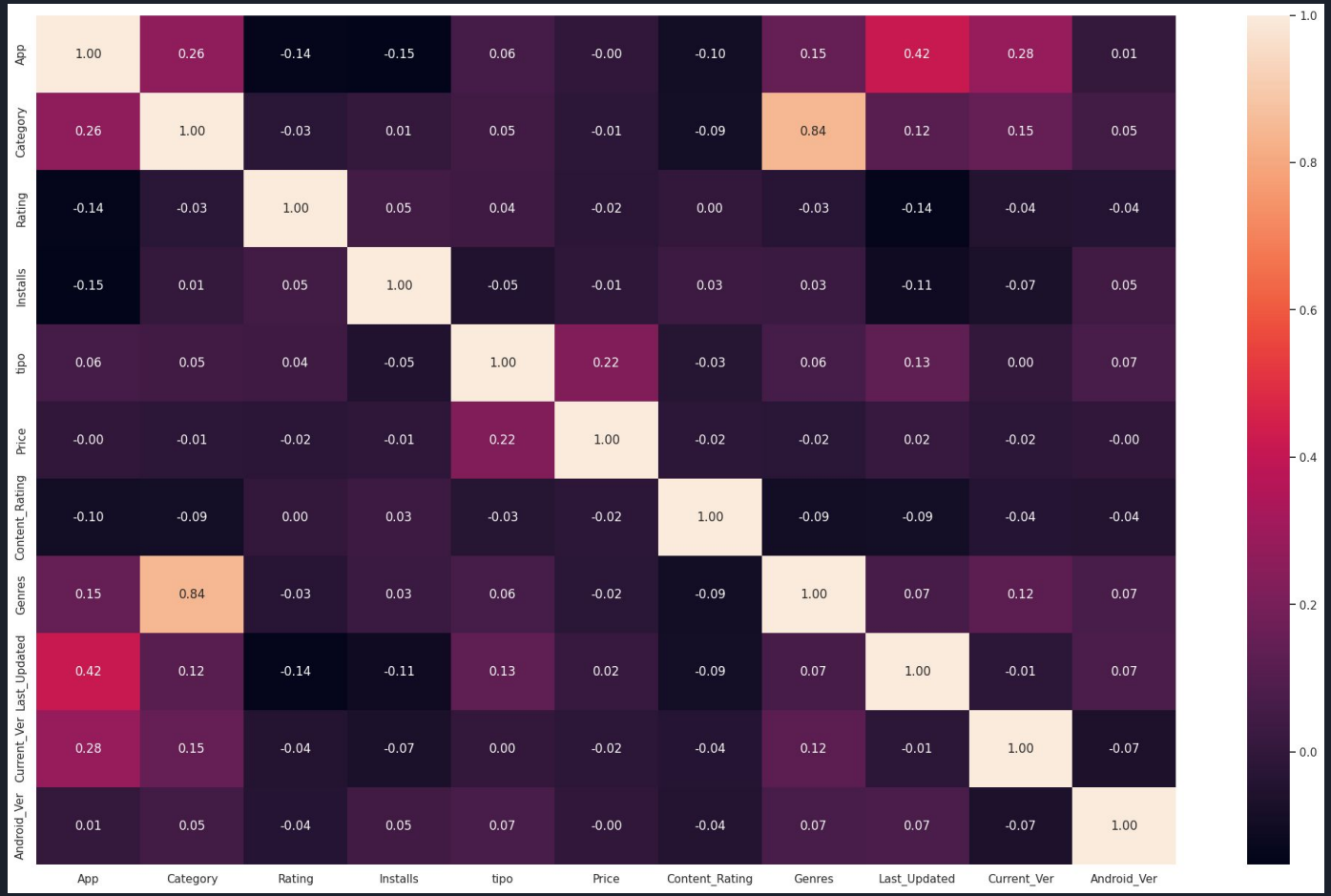
Variables empleadas

- App: El nombre de la aplicación.
- Category: La categoría a la que pertenece la aplicación.
- Rating: La calificación promedio de la aplicación.
- Installs: El número de instalaciones de la aplicación.
- tipo: El tipo de aplicación (gratuita o de pago).
- Price: El precio de la aplicación.
- Content_Rating: La clasificación de contenido de la aplicación.
- Genres: Los géneros a los que pertenece la aplicación.
- Last_Updated: La fecha en que se actualizó por última vez la aplicación.
- Current_Ver: La versión actual de la aplicación.
- Android_Ver: La versión mínima de Android requerida para ejecutar la aplicación.

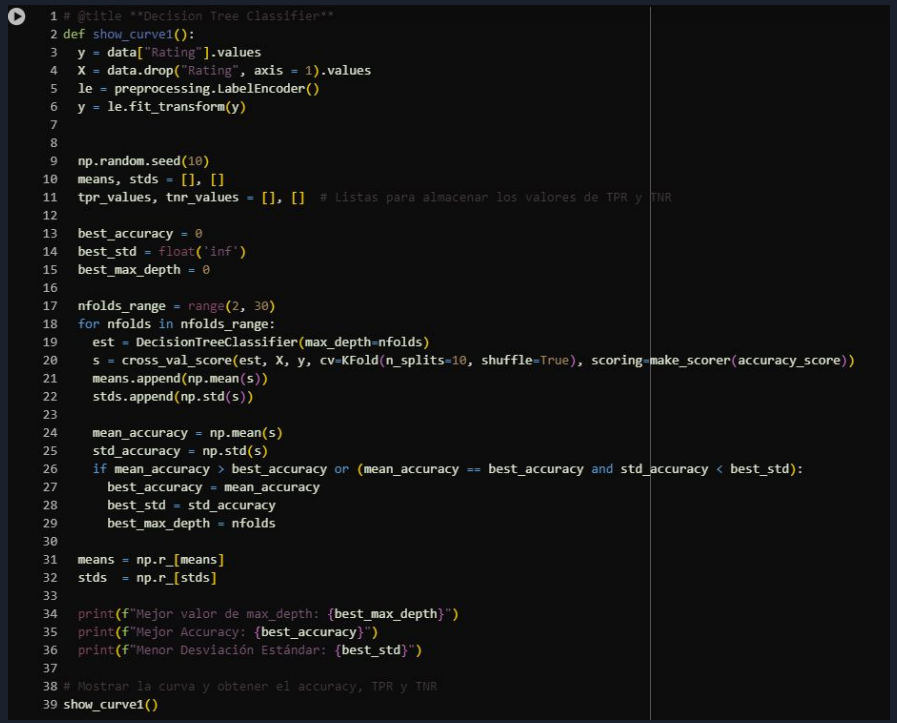
Dataset

App	Category	Rating	Installs	tipo	Price	Content_Rating	Genres	Last_Updated	Current_Ver	Android_Ver
0	0	4.1	10000.0	0	0.0	0.0	0	0	0	0
1	0	3.9	500000.0	0	0.0	0.0	1	1	1	0
2	0	4.7	5000000.0	0	0.0	0.0	0	2	2	0
3	0	4.5	50000000.0	0	0.0	1.0	0	3	3	1
4	0	4.3	100000.0	0	0.0	0.0	2	4	4	2
5	0	4.4	50000.0	0	0.0	0.0	0	5	5	3
6	0	3.8	50000.0	0	0.0	0.0	0	6	4	0
7	0	4.1	1000000.0	0	0.0	0.0	0	7	6	1
8	0	4.4	1000000.0	0	0.0	0.0	0	8	7	4
9	0	4.7	10000.0	0	0.0	0.0	2	9	8	0
10	0	4.4	1000000.0	0	0.0	0.0	0	10	9	5
11	0	4.4	1000000.0	0	0.0	0.0	0	11	10	6
12	0	4.2	10000000.0	0	0.0	1.0	0	12	11	5
13	0	4.6	100000.0	0	0.0	0.0	0	13	9	2
14	0	4.4	100000.0	0	0.0	0.0	0	14	12	3

Diagrama de correlación



Decision Tree Classifier



```python
# @title **Decision Tree Classifier**
def show_curve1():
  y = data["Rating"].values
  X = data.drop("Rating", axis = 1).values
  le = preprocessing.LabelEncoder()
  y = le.fit_transform(y)


  np.random.seed(10)
  means, stds = [], []
  tpr_values, tnr_values = [], []  # Listas para almacenar los valores de TPR y TNR

  best_accuracy = 0
  best_std = float('inf')
  best_max_depth = 0

  nfolds_range = range(2, 30)
  for nfolds in nfolds_range:
    est = DecisionTreeClassifier(max_depth=nfolds)
    s = cross_val_score(est, X, y, cv=KFold(n_splits=10, shuffle=True), scoring=make_scorer(accuracy_score))
    means.append(np.mean(s))
    stds.append(np.std(s))

    mean_accuracy = np.mean(s)
    std_accuracy = np.std(s)
    if mean_accuracy > best_accuracy or (mean_accuracy == best_accuracy and std_accuracy < best_std):
      best_accuracy = mean_accuracy
      best_std = std_accuracy
      best_max_depth = nfolds

  means = np.r_[means]
  stds  = np.r_[stds]

  print(f"Mejor valor de max_depth: {best_max_depth}")
  print(f"Mejor Accuracy: {best_accuracy}")
  print(f"Menor Desviación Estándar: {best_std}")

# Mostrar la curva y obtener el accuracy, TPR y TNR
show_curve1()
```

El modelo de árbol de decisión con una profundidad máxima de 29 mostró un rendimiento de baja precisión con un accuracy del 31%, lo que sugiere que el modelo tiene dificultades para predecir correctamente las clases de las muestras. A pesar de una desviación estándar baja que indica estabilidad en el rendimiento, el bajo accuracy señala que el modelo puede no estar generalizando bien a datos nuevos o que está sobreajustado a los datos de entrenamiento.

Resultados del algoritmo:

Mejor valor de max_depth: 29

Mejor Accuracy: 0.3101476014760148

Menor Desviación Estándar: 0.014019712355360049

Random Forest Classifier

```python
#@title  **Random Forest Classifier**
def show_RandomForestClassifier():
  np.random.seed(10)
  means, stds = [], []

  best_accuracy = 0
  best_std = float('inf')
  best_max_depth = 0

  nfolds_range = range(2,20)
  for nfolds in nfolds_range:
    est=RandomForestClassifier(max_depth=nfolds)
    s = cross_val_score(est, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
    means.append(np.mean(s))
    stds.append(np.std(s))
    media = np.mean(s)
    std_dev = np.std(s)
    mean_accuracy = np.mean(s)
    std_accuracy = np.std(s)
    if mean_accuracy > best_accuracy or (mean_accuracy == best_accuracy and std_accuracy < best_std):
      best_accuracy = mean_accuracy
      best_std = std_accuracy
      best_max_depth = nfolds

  means = np.r_[means]
  stds  = np.r_[stds]

  print(f"Mejor valor de max_depth: {best_max_depth}")
  print(f"Mejor Accuracy: {best_accuracy}")
  print(f"Menor Desviación Estándar: {best_std}")

show_RandomForestClassifier()
```

El modelo Random Forest Classifier con max_depth 19 tiene una precisión relativamente baja en la clasificación de las muestras, ya que clasifica correctamente aproximadamente el 32% de ellas. A pesar de tener una baja precisión, la consistencia de los resultados (reflejada en la baja desviación estándar) indica que el modelo tiene un rendimiento estable y constante en diferentes divisiones de los datos de prueba.

Resultados del algoritmo:

Mejor valor de max_depth: 19

Mejor Accuracy: 0.3187269372693727

Menor Desviación Estándar: 0.008557485168916468

Red Neuronal

```python
# @title **Red neuronal**
import random
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

y=data['Rating']
X=data.drop(columns=['Rating'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

y_train_ohe = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test_ohe = tf.keras.utils.to_categorical(y_test, num_classes=10)

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])


model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train_ohe, epochs=20, batch_size=10)

test_loss, test_acc = model.evaluate(X_test, y_test_ohe)

probs = model.predict(X_test.to_numpy())
preds = np.argmax(probs, axis=1)
```

- La precisión alcanzada en el conjunto de datos de prueba (76.33%) es relativamente buena y muestra un rendimiento aceptable del modelo para predecir las clases correctamente en datos no vistos.
- La reducción significativa en la función de pérdida indica que el modelo está convergiendo hacia un estado más óptimo, lo que es un indicador positivo.
- La precisión de entrenamiento se mantiene constante después de la época 8 en el 75.49%. Esto podría sugerir que el modelo ha alcanzado su límite en términos de capacidad predictiva, o que puede estar convergiendo a un mínimo local.

```
Epoch 1/20
656/656 [==============================] - 9s 13ms/step - loss: 590198.5625 - accuracy: 0.6595
Epoch 2/20
656/656 [==============================] - 8s 12ms/step - loss: 104088.1250 - accuracy: 0.6507
Epoch 3/20
656/656 [==============================] - 9s 13ms/step - loss: 184855.0781 - accuracy: 0.6245
Epoch 4/20
656/656 [==============================] - 8s 12ms/step - loss: 67042.1328 - accuracy: 0.6885
Epoch 5/20
656/656 [==============================] - 9s 14ms/step - loss: 17683.6348 - accuracy: 0.7094
Epoch 6/20
656/656 [==============================] - 10s 15ms/step - loss: 1774.2946 - accuracy: 0.7131
Epoch 7/20
656/656 [==============================] - 8s 12ms/step - loss: 219.5971 - accuracy: 0.7335
Epoch 8/20
656/656 [==============================] - 9s 14ms/step - loss: 0.7498 - accuracy: 0.7549
Epoch 9/20
656/656 [==============================] - 8s 12ms/step - loss: 0.7494 - accuracy: 0.7549
Epoch 10/20
656/656 [==============================] - 9s 14ms/step - loss: 0.7494 - accuracy: 0.7549
Epoch 11/20
656/656 [==============================] - 8s 13ms/step - loss: 0.7491 - accuracy: 0.7549
Epoch 12/20
656/656 [==============================] - 8s 13ms/step - loss: 0.7493 - accuracy: 0.7549
Epoch 13/20
656/656 [==============================] - 9s 14ms/step - loss: 0.7491 - accuracy: 0.7549
Epoch 14/20
656/656 [==============================] - 8s 12ms/step - loss: 0.7494 - accuracy: 0.7549
Epoch 15/20
656/656 [==============================] - 9s 13ms/step - loss: 0.7494 - accuracy: 0.7549
Epoch 16/20
656/656 [==============================] - 8s 12ms/step - loss: 0.7493 - accuracy: 0.7549
Epoch 17/20
656/656 [==============================] - 9s 13ms/step - loss: 0.7494 - accuracy: 0.7549
Epoch 18/20
656/656 [==============================] - 9s 13ms/step - loss: 0.7495 - accuracy: 0.7549
Epoch 19/20
656/656 [==============================] - 8s 12ms/step - loss: 0.7493 - accuracy: 0.7549
Epoch 20/20
656/656 [==============================] - 9s 13ms/step - loss: 0.7494 - accuracy: 0.7549
88/88 [==============================] - 0s 3ms/step - loss: 0.7408 - accuracy: 0.7633
88/88 [==============================] - 0s 3ms/step
```

Comparativa con diversos métodos

```
656/656 [==============================] - 7s 10ms/st
88/88 [==============================] - 1s 4ms/step
88/88 [==============================] - 0s 3ms/step
```

	Method	Best Accuracy
0	Decision Tree	0.274614
1	Random Forest	0.318727
2	Neural Network adam	0.763345
3	Neural Network SGD	0.000000

El método de **Redes Neuronales con el optimizador Adam** tiene la mejor precisión con un valor de **0.763345**. El método de **Árbol de Decisión** tiene la peor precisión con un valor de **0.274614**.

Estos resultados sugieren que el método de **Redes Neuronales con el optimizador Adam** es el más adecuado para el problema de clasificación que buscamos resolver

CONCLUSIONES

Los resultados muestran que los algoritmos de Decisión Tree y Random Forest tienen una precisión relativamente baja, mientras que las redes neuronales, especialmente con el optimizador 'adam' logran una precisión significativamente mejor pero se observa una baja precisión para la red neuronal con el optimizador 'SGD' lo que podría indicar problemas en su entrenamiento.

Se resalta la eficiencia de una red neuronal en donde nos da un mayor valor para poder realizar una clasificación.