Mondragon Unibertsitatea

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Popbl6

Unai Orive, Daniel Luengo, Egoitz San Martin, Unai Mendieta y Unai Aguinaco

Mudley

Sistema de recomendación

Mondragon Unibertsitatea Escuela Politécnica Superior

Preproceso

Para empezar a analizar la base de datos se ha hecho un análisis para mirar si información faltante o líneas duplicadas:

```
import numpy as np
import pandas as pd

data = pd.read_csv('user_artists.dat', sep="\t", skiprows=1,
names=['userID','artistID', 'weight'])
data = data.replace('?', np.NaN)
print('Número de instancias = %d' % (data.shape[0]))
print('Número de atributos = %d' % (data.shape[1]))
print('Valores faltantes:')
data.isnull().sum()
```

```
Número de instancias = 92834
Número de atributos = 3
Valores faltantes:
userID      0
artistID    0
weight      0
dtype: int64
```

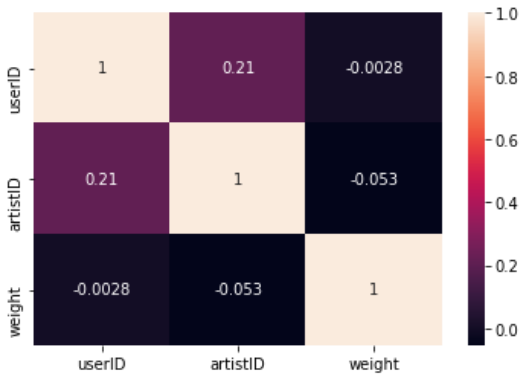
Y para las líneas duplicadas:

```
dups = data.duplicated()
print('Lineas duplicadas: %d' % (dups.sum()))
```

```
Líneas duplicadas: 0
```

Antes de analizar el modelo se va a mirar la correlación entre las variables; que, como vamos a ver, no es necesario suprimir ninguna ya que no guardan ninguna relación:

```
import seaborn as sns
Var_Corr = data.corr()
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns,
yticklabels=Var_Corr.columns, annot=True)
```

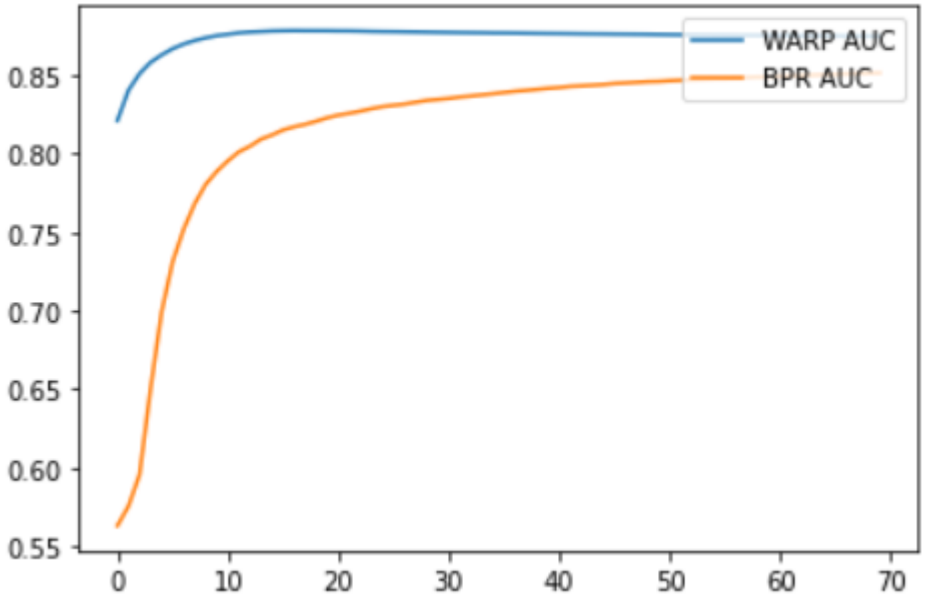


Para la selección de un modelo se han desarrollado dos IAs: una utilizando la librería de LightFM y otra creando una red neuronal (ver apéndice). Tras analizar la precisión de los dos modelos se concluye que el de LightFM es más preciso al obtener un 88% de precisión frente al 69% que ofrece la red neuronal. Otro aspecto importante a tener en cuenta es la velocidad de entrenado de los dos modelos; en este aspecto el modelo de LightFM es claramente superior al necesitar solo 25s en entrenar el modelo frente a los 17 minutos que necesita la red neuronal para entrenarse.

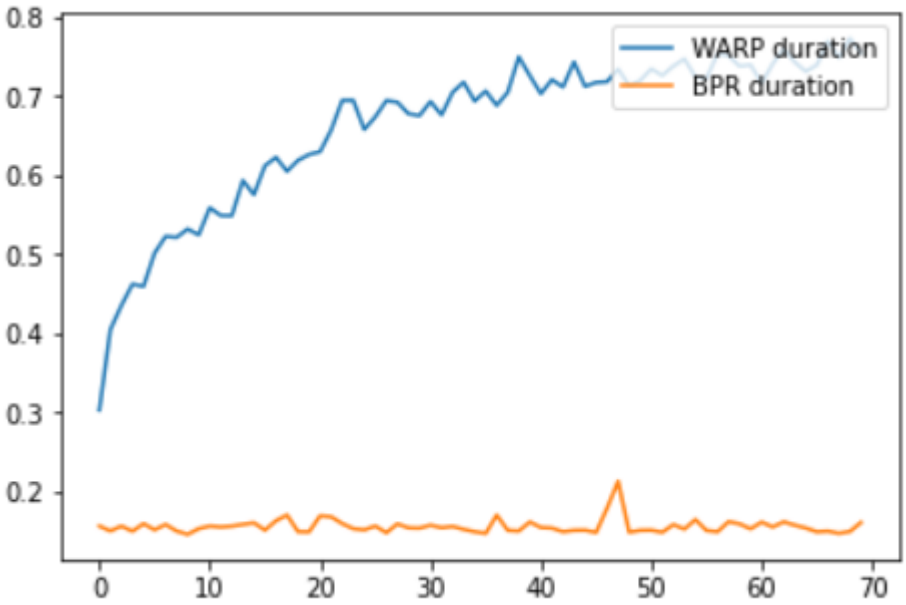
Debido a la velocidad de entendimiento del modelo elegido se utilizará NodeRed para entrenar el modelo una vez al día y así conseguir que los resultados sean lo más ajustados posibles a cada usuario.

Uno de los parámetros a tener en cuenta para la creación del modelo de LightFM es la pérdida o loss entre las que destacan dos funciones BRP y WARP. Aunque el objetivo es lograr el resultado que más se ajuste a los intereses del usuario, se ha hecho un análisis para ver cual de los dos modelos es el más conveniente.

Para realizar este análisis se va a realizar un entrenamiento durante 70 “epochs” o épocas para ver tanto la velocidad como la precisión de entrenamiento de cada uno de ellos.

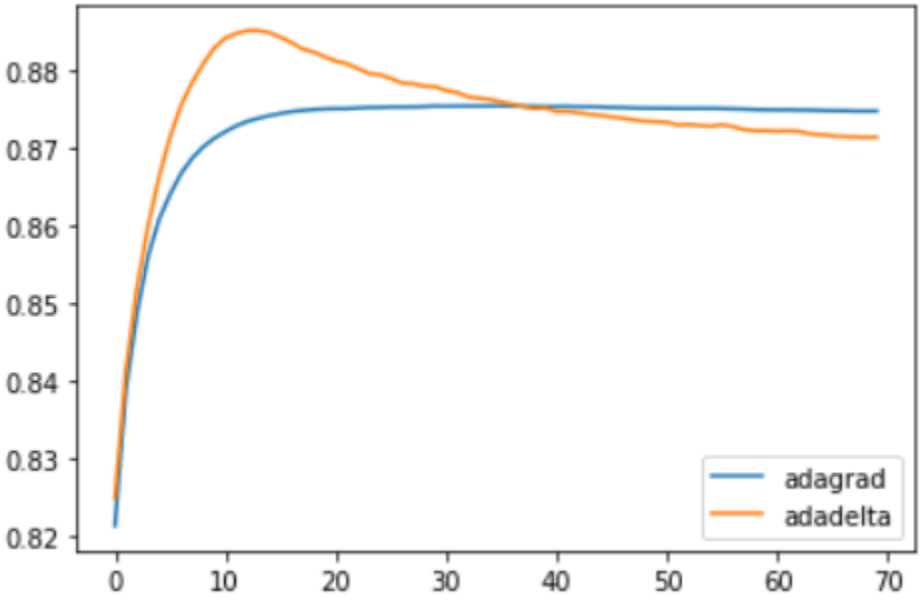


Como se puede observar, el modelo WARP empieza y acaba con una mejor tasa de acierto llegando hasta el 87% mientras que el BPR a pesar de que empieza con un 56% alcanza un 84% al final de las épocas.



Teniendo en cuenta la velocidad que se necesita para entrenar el modelo es evidente que el modelo BPR es mucho más rápido que el WARP pero como la diferencia es ínfima y el objetivo es la precisión el modelo WARP elegido.

Una vez seleccionado el tipo de pérdida que se va a utilizar es imprescindible seleccionar qué programa de ritmo de aprendizaje se va a utilizar. para esta librería existen dos tipos: adagrad y adadelta.



Cómo se puede ver aunque el modelo adadelta produzca mejores resultados al principio del entrenamiento, decae a través del tiempo y obtiene peores resultados al final de las épocas respecto al modelo adadelta. Es por esto que se ha decidido utilizar el modelo adagrad ya que proporciona una mayor firmeza y una precisión que se mantiene a través del tiempo.

LightFM

LightFM es un modelo de recomendación híbrido de representación latente. El modelo aprende incrustaciones (representaciones latentes en un espacio de alta dimensión) para usuarios y elementos de una manera que codifica las preferencias del usuario sobre los elementos. Cuando se multiplican, estas representaciones producen puntuaciones para cada elemento de un usuario determinado; es más probable que los elementos con una puntuación alta resulten interesantes para el usuario.

Las representaciones del usuario y del artículo se expresan en términos de representaciones de sus características: se estima una incrustación para cada característica, y estas características luego se suman para llegar a las representaciones de los usuarios y los artículos. Por ejemplo, si el artista 'Barricada' se describe con las siguientes características: ‘rock urbano, 'Pamplona' y 'Barricada', entonces su “embeding”se realizará tomando las incrustaciones de las características y agregándolas juntos. Lo mismo se aplica a las funciones de usuario.

Las incrustaciones se aprenden a través de métodos de descenso de gradiente estocástico.

Parámetros

no_components: la dimensionalidad de las incrustaciones latentes de características.
k: para el entrenamiento de k-OS, el k-ésimo ejemplo positivo se seleccionará de los n ejemplos positivos muestreados para cada usuario.
n: para el entrenamiento de k-OS, número máximo de positivos muestreados para cada actualización.
learning_schedule : uno entre ('adagrad', 'adadelta').
loss: uno de ('logistic', 'bpr', 'warp', 'warp-kos'): la función de pérdida.
learning_rate: tasa de aprendizaje inicial para el programa de aprendizaje de adagrad.
rho: coeficiente de promedio móvil para el programa de aprendizaje adadelta.
epsilon: parámetro de acondicionamiento para el programa de aprendizaje de adadelta.
item_alpha:Penalización L2 en las características del artículo. Consejo: establecer este número demasiado alto puede ralentizar el entrenamiento. Una buena forma de comprobarlo es si los pesos finales de las incrustaciones resultaron ser en su mayoría cero. La misma idea se aplica al parámetro user_alpha.
user_alpha: penalización L2 en las funciones del usuario.
max_sampled: número máximo de muestras negativas utilizadas durante el ajuste WARP. Requiere mucho muestreo para encontrar tripletes negativos para usuarios que ya están bien representados por el modelo; esto puede provocar tiempos de entrenamiento muy largos y un sobreajuste. Establecer esto en un número más alto generalmente conducirá a tiempos de entrenamiento más largos, pero en algunos casos puede mejorar la precisión.
random_state: la semilla del generador de números pseudoaleatorios que se utilizará al mezclar los datos y al inicializar los parámetros.

Variables

item_embeddings: contiene los vectores latentes estimados para las características del elemento. La entrada [i, j] -ésima da el valor del j-ésimo componente para la característica del i-ésimo elemento. En el caso más simple, en el que la matriz de características del elemento es una matriz de identidad, la i-ésima fila representará el vector latente del i-ésimo elemento.

user_embeddings: contiene los vectores latentes estimados para las características del usuario. La entrada [i, j] -ésima da el valor del j-ésimo componente para la i-ésima característica de usuario. En el caso más simple donde la matriz de características del usuario es una matriz de identidad, la i-ésima fila representará el i-ésimo vector latente de usuario.
item_biases: contiene los sesgos de item_features.
user_biases: contiene los sesgos de user_features.

Funciones utilizadas

A continuación se detallarán todas las funciones empleadas durante la creación y el despliegue del agente.

Ajustado del modelo

Para ajustar el modelo se van a utilizar dos funciones:

Fit: Se ajusta al modelo.
Fit partial:Se ajusta al modelo. A diferencia del ajuste, las llamadas repetidas a este método harán que el entrenamiento se reanude desde el estado actual del modelo.

Construcción de conjuntos de datos

A continuación se detallarán el procedimiento seguido para construir el conjuto de datos para el modelo.

Dataset

Es una herramienta para construir interacciones y matrices de características, que se encarga del mapeo entre los identificadores de usuario / elemento y los nombres de las características y los índices de características internas.

Para crear un conjunto de datos. Se llama a fit o a fit-partial, proporcionando identificadores de usuario / artículo y nombres de funciones que se van a utilizar en el modelo. Esto creará asignaciones internas que traducen los identificadores y los nombres de las funciones a índices internos utilizados por el modelo LightFM.

Build Interactions es un iterable de (ID de usuario, ID de artículo) o (ID de usuario, ID de artículo, peso) para construir una matriz de interacciones y pesos.

Build_user / item_features con iterables de (user / item id, [features]) o (user / item id, {feature: feature weight}) para construir matrices de características.

Para agregar nuevas características o ID de usuario / artículo, se llama a fit partial

Parámetros:
user identity features: crea una función única para cada usuario además de otras funciones. Si es verdadero (predeterminado), se asignará un vector latente para cada usuario. Este es un valor predeterminado razonable para la mayoría de las aplicaciones, pero debe establecerse en falso si hay muy pocos datos para cada usuario.
item identity features: crea una característica única para cada elemento además de otras características. Si es verdadero (predeterminado), se asignará un vector latente para cada elemento. Este es un valor predeterminado razonable para la mayoría de las aplicaciones, pero debe establecerse en falso si hay muy pocos datos para cada elemento.

Build item features

Construye una matriz de características del artículo a partir de un iterable del formulario (identificación del artículo, [lista de nombres de características]) o (identificación del artículo, {nombre de la característica: peso de la característica}).

Devoluciones
Matriz de características: matriz de características de los elementos.

Tipo de retorno
Matriz de CSR (número de elementos, número de funciones)

Build user features

Construye una matriz de características de usuario a partir de un iterable del formulario (identificación de usuario, [lista de nombres de características]) o (identificación de usuario, {nombre de característica: peso de característica}).

Devoluciones
Matriz de funciones: matriz de funciones de usuario.

Tipo de retorno
Matriz de RSE (número de usuarios, número de funciones)

Fit

Ajuste la identificación de usuario / artículo y las asignaciones de nombre de función.
Llamar a Fit por segunda vez restablecerá las asignaciones existentes.

Fit partial

Ajuste la identificación de usuario / artículo y las asignaciones de nombre de función.
Llamar a Fit por segunda vez agregará nuevas entradas a las asignaciones existentes.

Validación cruzada

Funciones de división de conjuntos de datos.

Random train test split

Interacciones divididas aleatoriamente entre entrenamiento y prueba. Esta función toma un conjunto de interacción y lo divide en dos conjuntos separados, un conjunto de entrenamiento y un conjunto de prueba.

Parámetros
interacciones: las interacciones que se van a dividir.
test percentage : la fracción de interacciones que se colocará en el conjunto de prueba.

random_state : el estado aleatorio utilizado para la reproducción aleatoria.

Devoluciones:
Matriz COO: Una tupla de (datos de entrenamiento, datos de prueba)

Evaluación del modelo

Módulo que contiene funciones de evaluación adecuadas para juzgar el rendimiento de un modelo LightFM ajustado.

Precisión en k

Mida la precisión en la métrica k para un modelo: la fracción de positivos conocidos en las primeras k posiciones de la lista clasificada de resultados. Una puntuación perfecta es 1.0.

Auc score

Mida la métrica ROC AUC para un modelo: la probabilidad de que un ejemplo positivo elegido al azar tenga una puntuación más alta que un ejemplo negativo elegido al azar. Una puntuación perfecta es 1.0.

Desarrollo del agente

Estos son los pasos a seguir para construir el agente.

Construyendo las asignaciones de ID

Lo primero que debemos hacer es crear un mapeo entre el usuario y los identificadores de elementos desde nuestros datos de entrada a los índices que nuestro modelo usará internamente.

Hacemos esto porque LightFM trabaja con identificadores de usuario y artículo que son números enteros consecutivos no negativos. La clase Dataset nos permite crear un mapeo entre los ID que usamos en nuestros sistemas y los índices consecutivos preferidos por el modelo.

Para hacer esto, creamos un conjunto de datos y llamamos a su fitmétodo. El primer argumento es iterable de todos los identificadores de usuario en nuestros datos, y el segundo es un iterable de todos los identificadores de elementos. En este caso, usamos expresiones generadoras para iterar perezosamente sobre nuestros datos y generar identificadores de usuario y artículo:

```
dataset = Dataset()
dataset.fit((x['userID'] for x in ratings), (x['artistID'] for x in
ratings))
num_users, num_items = dataset.interactions_shape()
```

Esta llamada asignará una identificación numérica interna a cada usuario e identificación de elemento que pasamos. Estos serán contiguos (de 0 a la cantidad de usuarios y elementos que tengamos), y también determinarán las dimensiones del modelo de LightFM resultante.

A continuación, se usará fit partial para agregar algunas asignaciones de características de elementos:

```
dataset.fit_partial(items=(x['id'] for x in dict_list),users=(x['id'] for
x in user_list),
                    item_features=i1,user_features=users)
```

Antes de crear el modelo hay que crear las matrices de relaciones para los usuarios y los artistas. Para ello utilizaremos las funciones que proporciona LightFM:

```
item_features = dataset.build_item_features(((x['id'], [x['genre']]) for x
in dict_list))
user_features = dataset.build_user_features(((x['id'], [x['friendId']])
for x in users))
```

Construyendo la matriz de interacciones

Una vez creado el mapeo, construimos la matriz de interacción:

```
(interactions, weights) = dataset.build_interactions(((x['userID'],
x['artistID']) for x in ratings))
```

Antes de crear el modelo, tenemos que dividir los datos en train y test. Necesitamos que tengan la misma dimensión así que usaremos la función proporcionada por la librería

```
train, test = random_train_test_split(interactions, test_percentage=0.2,
random_state=42)
```

Construyendo un modelo

Por último, se crea el modelo de LightFM

```
NUM_THREADS = 16
NUM_COMPONENTS = 30
NUM_EPOCHS = 70
ITEM_ALPHA = 1e-6
model = LightFM(loss='warp',
                item_alpha=ITEM_ALPHA,learning_schedule='adagrad',
                no_components=NUM_COMPONENTS)

model = model.fit(train, epochs=NUM_EPOCHS, num_threads=NUM_THREADS)
model.fit(train, item_features=item_features,user_features=user_features)
train_auc = auc_score(model,
                      train,
```

```
                    item_features=item_features,
                    num_threads=NUM_THREADS).mean()
```

Predicción

Para predecir las recomendaciones para un usuario, basta con introducir el id y los item labels.

```
def sample_recommendation(model, train, test,
item_labels,data,item_labels_csv, user_ids):
   for user_id in user_ids:
       # artistas que ya les gustan
       ids_artistas = data['artistID'][data['userID'] == user_id]
       known_positives =
item_labels_csv['name'][item_labels_csv['id'].isin(ids_artistas)]
       # artistas recomendados
       scores = model.predict(user_id, np.arange(item_labels.size))
       # artistas ordenados de mayor a menor
       top_items = item_labels[np.argsort(-scores)]
   return top_items
```

FastAPI

FastAPI es un marco web moderno y rápido (de alto rendimiento) para crear API con Python 3.6+ basado en sugerencias de tipo Python estándar.

```
#!pip install fastapi uvicorn

# 1. Importar librerias
import uvicorn
from fastapi import FastAPI

# 2. Crear el objeto app
app = FastAPI()
# 3. Indexar la ruta,se abre automaticamente http://127.0.0.1:8000
@app.get('/')
def index():
    return {'message': 'Hola'}

# 4. Recibir un parametro
@app.get('/{name}')
def get_name(name: str):
    return {'message': f'Hello, {name}'}

# 5. Ejecutar la api con uvicorn
if __name__ == '__main__':
    uvicorn.run(app, host='127.0.0.1', port=8000)
```

Para nuestro programa se han utilizado de la siguiente manera:

```
@app.post('/mostrarRecomendaciones')
def recomendaciones1(recomend: Recomend):
   resultados=sample_recommendation(model, train, test,
item_labels,data,item_labels_csv, [485])
   lista = list()
   for x in resultados[:12]:
       i = item_labels_csv[item_labels_csv['name'] == x]
       y = i.to_string(header=False,index=False).split('\n')
       lista.append(y)
   x =
'{"name":"'+str(recomend.name)+'","results":["'+str(lista[0])+'","'+str(li
sta[1])+'","'+str(lista[2])+'","'+str(lista[3])+'","'+str(lista[4])+'","'+
str(lista[5])+'","'+str(lista[6])+'","'+str(lista[7])+'","'+str(lista[8])+
'","'+str(lista[9])+'","'+str(lista[10])+'","'+str(lista[11])+'"]}'
   y = json.loads(x)
   return y
```

Como se puede ver se recibe un objeto llamado recomend que llega en formato json y se transforma a un objeto python. Para que esto funcione hay que crear el siguiente objeto.

```
class Search(BaseModel):
   name: str
   city: str
   date: str
   gender: str
   budget: str
```

Apéndice

Red neuronal

Primero, es necesario realizar un procesamiento previo para codificar usuarios y películas como índices enteros.

```
user_ids = data["userID"].unique().tolist()
user2user_encoded = {x: i for i, x in enumerate(user_ids)}
userencoded2user = {i: x for i, x in enumerate(user_ids)}
artist_ids = data["artistID"].unique().tolist()
artist2artist_encoded = {x: i for i, x in enumerate(artist_ids)}
artist_encoded2artist = {i: x for i, x in enumerate(artist_ids)}
data["user"] = data["userID"].map(user2user_encoded)
data["artist"] = data["artistID"].map(artist2artist_encoded)

num_users = len(user2user_encoded)
num_artists = len(artist_encoded2artist)
data["weight"] = data["weight"].values.astype(np.float32)

min_rating = min(data["weight"])
```

```
max_rating = max(data["weight"])

print(
    "Number of users: {}, Number of artists: {}, Min rating: {}, Max rating:
{}".format(
        num_users, num_artists, min_rating, max_rating
    )
)
```

Después, hay que preparar datos de formación y validarlos

```
data = data.sample(frac=1, random_state=42)
x = data[["user", "artist"]].values
y = data["weight"].apply(lambda x: (x - min_rating) / (max_rating -
min_rating)).values
train_indices = int(0.9 * data.shape[0])
x_train, x_val, y_train, y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:],
```

A continuación, se crea el modelo. Incorporamos tanto usuarios como artistas en vectores de 50 dimensiones.
El modelo calcula una puntuación de coincidencia entre las incrustaciones del usuario y el artistas a través de un producto escalar y agrega un sesgo por artistas y por usuario. La puntuación de coincidencia se escala al [0, 1] intervalo a través de un sigmoide (ya que nuestras calificaciones se normalizan a este rango).

```
EMBEDDING_SIZE = 50
class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_artists, embedding_size, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_artists = num_artists
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.artist_embedding = layers.Embedding(
            num_artists,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.artist_bias = layers.Embedding(num_artists, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
```

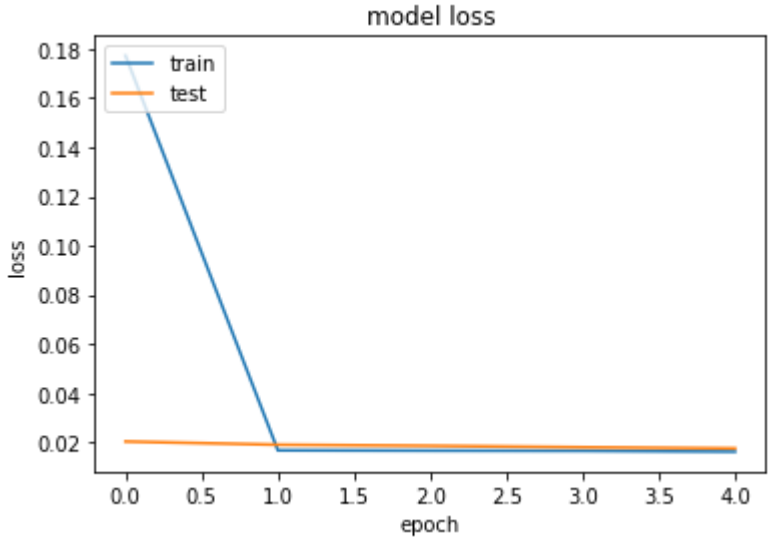
```
        artist_vector = self.artist_embedding(inputs[:, 1])
        artist_bias = self.artist_bias(inputs[:, 1])
        dot_user_artist = tf.tensordot(user_vector, artist_vector, 2)
        x = dot_user_artist + user_bias + artist_bias
        return tf.nn.sigmoid(x)


model = RecommenderNet(num_users, num_artists, EMBEDDING_SIZE)
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
metrics=[keras.metrics.SparseCategoricalAccuracy()],
optimizer=keras.optimizers.Adam(lr=0.001)
)
```

Ahora se entrena el modelo en función de la división de datos

```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=30,
    verbose=1,
    validation_data=(x_val, y_val),
)
```

Se traza pérdida de entrenamiento y validación:



Por último, se calculan las 10 mejores recomendaciones:

```
artist_df = pd.read_csv('artistsWithGenreNew2.dat', sep='\t', skiprows=1,
names=['id','name', 'url', 'pictureURL', 'genre'])
user_id = 785
artists_watched_by_user = data[data.userID == user_id]
artists_not_watched = artist_df[
    ~artist_df["id"].isin(artists_watched_by_user.artistID.values)
]["id"]
artists_not_watched = list(
set(artists_not_watched).intersection(set(artist2artist_encoded.keys())))
```

```
artists_not_watched = [[artist2artist_encoded.get(x)] for x in
artists_not_watched]
user_encoder = user2user_encoded.get(user_id)
user_artist_array = np.hstack(
    ([[user_encoder]] * len(artists_not_watched), artists_not_watched))
ratings = model.predict(user_artist_array).flatten()
top_ratings_indices = ratings.argsort()[-12:][::-1]
recommended_artist_ids = [
    artist_encoded2artist.get(artists_not_watched[x][0]) for x in
top_ratings_indices]
print("Enseñando recomendaciones del usuario: {}".format(user_id))
print("====" * 9)
print("Artistas con alta valoración del usuario")
print("----" * 8)
top_artists_user = (
    artists_watched_by_user.sort_values(by="weight", ascending=False)
    .head(5)
    .artistID.values
)
artist_df_rows = artist_df[artist_df["id"].isin(top_artists_user)]
for row in artist_df_rows.itertuples():
    print(row.name, ":", row.genre)
print("----" * 8)
print("Top 10 artistas recomendados")
print("----" * 8)
recommended_artists = artist_df[artist_df["id"].isin(recommended_artist_ids)]
for row in recommended_artists.itertuples():
    print(row.name, ":", row.genre)
```

```
Enseñando recomendaciones del usuario: 785
====================================
Artistas con alta valoración del usuario
--------------------------------
Cher : pop
David Bowie : rock
Pato Fu : rock
Rita Lee : rock
Ludov : rock
--------------------------------
Top 10 artistas recomendados
--------------------------------
Sarah Brightman : soundtracks
Ezginin GÃ1/4nlÃ1/4ÄŸÃ1/4 : folk
DICKY DIXON LAKE RECORDS : other
Viking Quest : pop
Luzmelt : other
Phillip Boa & The Voodooclub : indie
Iron Jesus : electronic
Brooke Fraser : folk
Dicky Dixon : other
RICHARD DIXON-COMPOSER : gospel
80kidz : electronic
```

```
Gay Fairy Tales : pop
```