

Instalaciones necesarias

```
In [ ]: # !pip install tensorflow
        # !pip install -q tensorflow-recommenders
        # !pip install -q --upgrade tensorflow-datasets
```

Importación de librerías

```
In [ ]: # Importación de Tensorflow
        from typing import Dict, Text

        import numpy as np
        import tensorflow as tf

        import tensorflow_recommenders as tfrs
```

```
In [ ]: # Importación de pandas para el manejo de dataframes, además de expresiones
        import pandas as pd
        import re
```

Variables generales

Se establecen para facilitar el cambio de variables recurrentes y/o cambiantes de forma sencilla

```
In [ ]: pathToDF = "../../../Inputs/Creados - Proyecto/"
        fileToDF = "dfVentasDefinitivo.csv"

        byColumn = "product_id"
```

Importación de datos

```
In [ ]: df = pd.read_csv(f"{pathToDF}{fileToDF}")
```

```
In [ ]: #Sólo ejecutar una vez, elimina primera columna si la exportación de dicho C
        df.drop(columns=df.columns[0], axis=1, inplace=True)
```

```
In [ ]: df.head(2)
```

```
Out[ ]:
```

	item_id	num_order	created_at	product_i
0	000010d95384a6ba3d57dd870e7b337c	65717498f0771a49497d80f11160093c	2017-09-22 15:46:37	564
1	00001a8fb0bd42b1e16ba731e30cc490	09b538e85ce396ecbb70695f91007830	2018-09-12 21:27:08	2874

2 rows × 26 columns

Depuración de datos

Aplicación de expresión regular que elimina cantidades de los nombres originales de los productos, con el fin de mejorar la legibilidad y posterior reducción.

```
In [ ]: df["name"] = df["name"].apply(lambda x: re.sub("\d+\s*\S*\w+\s*\S*\w", "", x))
```

Creación de dataset de ventas

```
In [ ]: df2 = df[["customer_id", byColumn]]
```

```
In [ ]: df3 = df2.drop_duplicates(subset=['customer_id'])
df3.reset_index(inplace=True)
df3.drop(df3.columns[[0]], axis=1, inplace=True)
df3.reset_index(inplace=True)
df3 = df3[["index", "customer_id"]]

df4 = df2
df4 = pd.merge(df3, df4, how="inner", on=["customer_id"])
df4 = df4[["index", byColumn]]
df4['index'] = df4['index'].astype(str)
df4.rename(columns = {'index': 'customer_id'}, inplace = True)
df4[byColumn] = df4[byColumn].astype("str")

print(df4.shape)
print(df4.dtypes)
df4.head()
```

/tmp/ipykernel_120782/1398491429.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df3.drop(df3.columns[[0]], axis=1, inplace=True)
(810167, 2)
customer_id    object
product_id     object
dtype: object
```

```
Out[ ]:
```

	customer_id	product_id
0	0	5645
1	0	36943
2	0	5645
3	0	8635
4	0	4629

```
In [ ]: # Conversión de dataframe de pandas para uso en Tensorflow
sales = tf.data.Dataset.from_tensor_slices(dict(df4))
```

Comprobación de conversión

```
In [ ]: for row in sales.take(5):
        print(row)
```

```
{'customer_id': <tf.Tensor: shape=(), dtype=string, numpy=b'0'>, 'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'5645'>}
```

```
{'customer_id': <tf.Tensor: shape=(), dtype=string, numpy=b'0'>, 'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'36943'>}
```

```
{'customer_id': <tf.Tensor: shape=(), dtype=string, numpy=b'0'>, 'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'5645'>}
```

```
{'customer_id': <tf.Tensor: shape=(), dtype=string, numpy=b'0'>, 'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'8635'>}
```

```
{'customer_id': <tf.Tensor: shape=(), dtype=string, numpy=b'0'>, 'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'4629'>}
```

Creación de dataset de productos

```
In [ ]: dfProduct = df2[byColumn].unique().tolist()
dfProduct = pd.DataFrame(dfProduct, columns=[byColumn])
dfProduct[byColumn] = dfProduct[byColumn].astype("str")
print(dfProduct.shape)
dfProduct.head(5)
```

```
(19787, 1)
```

```
Out[ ]:
```

	product_id
0	5645
1	28743
2	68986
3	9692
4	81921

```
In [ ]: # Conversión de dataframe de pandas para uso en Tensorflow
products = tf.data.Dataset.from_tensor_slices(dict(dfProduct))
```

Comprobación de conversión

```
In [ ]: for row in products.take(3):
        print(row)
```

```
{'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'5645'>}
{'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'28743'>}
{'product_id': <tf.Tensor: shape=(), dtype=string, numpy=b'68986'>}
```

Aplicación de TensorFlow

Se define el alcance de las variables

```
In [ ]: sales = sales.map(lambda x: {
        byColumn: x[byColumn],
        "customer_id": x["customer_id"],
    })
products = products.map(lambda x: x[byColumn])
```

```
In [ ]: sales
```

```
Out[ ]: <MapDataset element_spec={'product_id': TensorSpec(shape=(), dtype=tf.string, name=None), 'customer_id': TensorSpec(shape=(), dtype=tf.string, name=None)}>
```

```
In [ ]: sales_vocabulary = tf.keras.layers.StringLookup(mask_token=None)
sales_vocabulary.adapt(sales.map(lambda x: x["customer_id"]))

products_vocabulary = tf.keras.layers.StringLookup(mask_token=None)
products_vocabulary.adapt(products)
```

```
In [ ]: class RecomendatorModel(tfrs.Model):

    def __init__(
        self,
        sales_model: tf.keras.Model,
        products_model: tf.keras.Model,
        task: tfrs.tasks.Retrieval):
        super().__init__()

        self.sales_model = sales_model
        self.products_model = products_model

        self.task = task

    def compute_loss(self, features: Dict[Text, tf.Tensor], training=False) ->

        sales_embeddings = self.sales_model(features["customer_id"])
        products_embeddings = self.products_model(features[byColumn])

        return self.task(sales_embeddings, products_embeddings)
```

```
In [ ]: # Define capas modelos de ventas y productos
sales_model = tf.keras.Sequential([
    sales_vocabulary,
    tf.keras.layers.Embedding(sales_vocabulary.vocabulary_size(), 64)
])

products_model = tf.keras.Sequential([
    products_vocabulary,
    tf.keras.layers.Embedding(products_vocabulary.vocabulary_size(), 64)
])

# Define los objetivos
task = tf.rs.tasks.Retrieval(metrics=tf.rs.metrics.FactorizedTopK(
    products.batch(128).map(products_model)
))
```

```
In [ ]: # Creación del modelo
model = RecomendatorModel(sales_model, products_model, task)
model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))

# Entrenamiento para tres 'epochs'
model.fit(ratings.batch(4096), epochs=3)
```

```
Epoch 1/3
198/198 [=====] - 355s 2s/step - factorized_top_k/
top_1_categorical_accuracy: 2.2341e-04 - factorized_top_k/top_5_categorical
_accuracy: 0.0018 - factorized_top_k/top_10_categorical_accuracy: 0.0036 -
factorized_top_k/top_50_categorical_accuracy: 0.0154 - factorized_top_k/top
_100_categorical_accuracy: 0.0270 - loss: 34014.5863 - regularization_loss:
0.0000e+00 - total_loss: 34014.5863
Epoch 2/3
198/198 [=====] - 404s 2s/step - factorized_top_k/
top_1_categorical_accuracy: 0.0418 - factorized_top_k/top_5_categorical_acc
uracy: 0.1447 - factorized_top_k/top_10_categorical_accuracy: 0.1779 - fact
orized_top_k/top_50_categorical_accuracy: 0.2639 - factorized_top_k/top_100
_categorical_accuracy: 0.3080 - loss: 29515.8075 - regularization_loss: 0.0
000e+00 - total_loss: 29515.8075
Epoch 3/3
198/198 [=====] - 405s 2s/step - factorized_top_k/
top_1_categorical_accuracy: 0.0917 - factorized_top_k/top_5_categorical_acc
uracy: 0.2419 - factorized_top_k/top_10_categorical_accuracy: 0.2969 - fact
orized_top_k/top_50_categorical_accuracy: 0.4274 - factorized_top_k/top_100
_categorical_accuracy: 0.4890 - loss: 23852.8232 - regularization_loss: 0.0
000e+00 - total_loss: 23852.8232
```

```
Out[ ]: <keras.callbacks.History at 0x7f4ce374a920>
```

```
In [ ]: # Uso de fuerza bruta
index = tf.rs.layers.factorized_top_k.BruteForce(model.sales_model)
index.index_from_dataset(
    products.batch(100).map(lambda title: (title, model.products_model(title
```

```
Out[ ]: <tensorflow_recommenders.layers.factorized_top_k.BruteForce at 0x7f4ce374bd
30>
```

Guardado de modelo

```
In [ ]: #model.save_weights("checkpoint")
        #model.save('my_model.tf')
        #model.summary()
```

Chequeo preliminar de resultados

```
In [ ]: numCustomer = "40"
        _, titles = index(np.array([numCustomer]))
        print(f"Top 3 recommendations for User{numCustomer}: {titles[0, :5]}")
```

Top 3 recommendations for User40: [b'8362' b'62785' b'10906' b'88497' b'9537']

```
In [ ]: numCustomer = "1"
        _, titles = index(np.array([numCustomer]))
        print(f"Top 3 recommendations for User{numCustomer}: {titles[0, :5]}")
```

Top 3 recommendations for User1: [b'81348' b'56981' b'9972' b'91125' b'94992']

Depuración de resultados para dataframe final

```
In [ ]: df5=df4
        print(df5.shape)
        df5.head()
```

(810167, 2)

```
Out[ ]:   customer_id  product_id
0           0         5645
1           0        36943
2           0         5645
3           0         8635
4           0         4629
```

```
In [ ]: df5['recomendation1'] = np.nan
        df5['recomendation2'] = np.nan
        df5['recomendation3'] = np.nan
        df5.head()
```

```
Out[ ]:   customer_id  product_id  recomendation1  recomendation2  recomendation3
0           0         5645                NaN                NaN                NaN
1           0        36943                NaN                NaN                NaN
2           0         5645                NaN                NaN                NaN
3           0         8635                NaN                NaN                NaN
4           0         4629                NaN                NaN                NaN
```

A continuación se trasladan las recomendaciones del modelo a un dataframe de Pandas

```
In [ ]: for element in range(df5.shape[0]):
        _, titles = index(np.array([str(element)]))
        result = [element2.decode('utf-8') for element2 in titles.numpy()[0][:3]]
        df5.iloc[element, df5.columns.get_loc('recommendation1')] = result[0]
        df5.iloc[element, df5.columns.get_loc('recommendation2')] = result[1]
        df5.iloc[element, df5.columns.get_loc('recommendation3')] = result[2]
        if element%10000==0:
            print(element)
df5.head()
```

En este punto, se prepara el dataframe de Pandas obtenido para su uso en el Recomendador

```
In [ ]: df6 = df5
```

```
In [ ]: df7 = pd.merge(df6, df2, right_index=True, left_index=True)
df7=df7[["customer_id_y", f"{byColumn}_y", "recommendation1", "recommendation2"]
df7.rename(columns = {'customer_id_y':'customer_id', "name_y": "name"}, inplace=True)
df7.head()
```

```
Out[ ]:
```

	customer_id	product_id_y	recommendation1	recommendation2	recommendation3
0	da5b59745c6a4699dee7684eba901bba	5645	36943	5645	
1	531a918355010bacbe506243a5f05c30	28743	81348	56981	
2	14e6f6400d1c114d509844be3687cb19	68986	72920	95729	
3	872bd419dfb24caf4f996a2cd2b8a9b4	9692	10504	72240	
4	8a1b78fb0503a964a7fb19135d429b78	81921	81921	62707	

```
In [ ]: df8=df7
df8 = df8.groupby(by="customer_id", dropna=False).first().reset_index()

print(df8.shape)
df8.tail()

(113522, 5)
```

```
Out[ ]:
```

	customer_id	product_id_y	recommendation1	recommendation2	recommendation3
113517	ffc9e0a62f07e67ff85803a8b5f30cf	12137	2605	3714	
113518	ffe0497986df50816e428af728f8900	76271	10463	24945	
113519	fffed4187f3b5f17cb58536f7fac8dee	9964	10692	23410	
113520	fff748a7ac35759d9fef57a34fd4a21	8153	27765	134	
113521	fffb88e89a23a34d3d98282bad3889a	33997	10692	23410	

Exportación de resultados

```
In [ ]: df8.to_csv(f"recommendations({byColumn})_V7.csv", index=False)
```