بسم الله الرحمان الرحیم

موضوع پروژه:

سیستم های پیشنهاد دهنده و

Transformer

گردآورنده:

سبحان قنبری

استاد:

آقای علی عالمی متین پور

A Transformer-based recommendation system

معرفی:

این مدل از رفتار توالی کاربران در تماشای فیلم و نمره ای که کاربران به فیلم می دهند با استفاده از دیتاست movielens datasets عمل می کند. همچنین از مشخصات کاربر و ویژگی های فیلم برای پیشبینی امتیاز کاربر به فیلم استفاده می کند. هدف این مدل این است که بتواند رتبه بندی یک فیلم را پیشبینی کند و بتواند به کاربر فیلم های پیشنهادی براساس مواردی مانند فیلم های دیده شده توسط کاربر، نمرات داده شده توسط کاربر، سن شغل و رده سنی کاربر، ژانر مورد علاقه کاربر و ...

دیتاست:

دیتاست Movielens از یک میلیون مجموعه استفاده می کند که شامل یک میلیون رتبه بندی از 6000 کاربر در 4000 فیلم همراه با برخی ویژگی های کاربر است

Setup:

```
import os
import math
from zipfile import ZipFile
from urllib.request import urlretrieve
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import StringLookup
```

از کتابخانه های math,numpy,pandas,tensorflow,keras,zipfile,os,urllib استفاده میشود

```
urlretrieve("http://files.grouplens.org/datasets/movielens/ml-1m.zip",
"movielens.zip")
```

در مرحله اول باید دیتاست را دانلود کنیم و سپس آن را بارگزاری کنیم

```
users = pd.read_csv(
    "ml-1m/users.dat",
    sep="::",
    names=["user_id", "sex", "age_group", "occupation", "zip_code"],
)

ratings = pd.read_csv(
    "ml-1m/ratings.dat",
    sep="::",
    names=["user_id", "movie_id", "rating", "unix_timestamp"],
)

movies = pd.read_csv(
    "ml-1m/movies.dat", sep="::", names=["movie_id", "title", "genres"],
encoding='latin-1'
)
```

ابتدا یک کاربر تعریف بارگزاری کردیم که شامل ایدی، جنسیت، گروه سنی، شغل، زیپ کد هست

بعد از آن یک رتبه بندی بارگزاری کردیم که شامل ایدی کاربر، ایدی فیلم، امتیاز و مدت زمان فیلم است

در مرحله اخر فیلم را تعریف کردیم که ویژگی های ایدی، عنوان، ژانر را دارد

```
users.head()
```

دستور head() مشخصات پنج کاربر اول را نشان می دهد

User_id	sex	Age_group	occupation	zipcode
0	F	1	10	48067
1	M	56	16	70072
2	M	25	15	55117

3	M	45	7	02460
4	M	25	20	55455

همچنین این دستور را می توان برای رده بندی و فیلم استفاده کرد

```
users["user_id"] = users["user_id"].apply(lambda x: f"user_{x}")
users["age_group"] = users["age_group"].apply(lambda x: f"group_{x}")
users["occupation"] = users["occupation"].apply(lambda x: f"occupation_{x}")

movies["movie_id"] = movies["movie_id"].apply(lambda x: f"movie_{x}")

ratings["movie_id"] = ratings["movie_id"].apply(lambda x: f"movie_{x}")
ratings["user_id"] = ratings["user_id"].apply(lambda x: f"user_{x}")
ratings["rating"] = ratings["rating"].apply(lambda x: float(x))
```

با این کد برای کاربرها پیشوند کاربر، برای گروه سنی پیشوند گروه و برای شغل ها پیشوند شغل ایجاد کردیم

The Greek alphabet

User_id	Sex	Age_group	Occupation	Zip_code
User_1	F	$group\_1$	Occupation_10	48067
User_2	M	$group\_56$	Occupation_16	70072
User_3	M	$group\_25$	Occupation_15	55117
User_4	M	$group\_45$	Occupation_7	02460
User_5	M	$group\_25$	Occupation_20	55455

```
genres = [
    "Action",
    "Adventure",
    "Animation",
    "Children's",
    "Comedy",
    "Crime",
    "Documentary",
    "Drama",
    "Fantasy",
    "Film-Noir",
    "Horror",
    "Musical",
    "Mystery",
    "Romance",
    "Sci-Fi",
```

```
    "Thriller",
    "War",
    "Western",
]

for genre in genres:
    movies[genre] = movies["genres"].apply(
        lambda values: int(genre in values.split("|"))
```

ویژگی مربوط به ژانرها را فراخواندیم و با استفاده از دستور for برای هر فیلم ژانر مربوط به خودش را با دستور apply قرار دادیم و در اخر با دستور split() آن هارا از هم با ("|") جدا کردیم.

```
ratings_group = ratings.sort_values(by=["unix_timestamp"]).groupby("user_id")

ratings_data = pd.DataFrame(
    data={
        "user_id": list(ratings_group.groups.keys()),
        "movie_ids": list(ratings_group.movie_id.apply(list)),
        "ratings": list(ratings_group.rating.apply(list)),
        "timestamps": list(ratings_group.unix_timestamp.apply(list)),
    }
)
```

در این بخش کد دیتافریمی ساخته می شود که به کاربرها لیستی از فیلم با ایدی آن ها داده می شودکه آن ها را براساس timestamp مرتب سازی کرده است و این مرتب سازی از کوچک به بزرگ است. مراحل مرتب سازی ضروری است زیرا سیستم می تواند ارتباط بن وابستگی زمانی و کاربر-فیلم را درک کند همچنین گروهبندی کردن داده باعث سرعت بخشیدن به پردازش می شود

```
sequence_length = 4
step_size = 2


def create_sequences(values, window_size, step_size):
    sequences = []
    start_index = 0
    while True:
        end_index = start_index + window_size
        seq = values[start_index:end_index]
        if len(seq) < window_size:
            seq = values[-window_size:]
            if len(seq) == window_size:
                sequences.append(seq)
            break
        sequences.append(seq)
        start_index += step_size
    return sequences


ratings_data.movie_ids = ratings_data.movie_ids.apply(
    lambda ids: create_sequences(ids, sequence_length, step_size)
)

ratings_data.ratings = ratings_data.ratings.apply(
    lambda ids: create_sequences(ids, sequence_length, step_size)
)

del ratings_data["timestamps"]
```

در اینجا کدی قرار دارد که به صورت چهارتایی و به فاصله دوتا دسته بندی انجام می شود که می توان از آن برای دسته بندی امتیاز فیلم و فیلم ها استفاده کرد و در اخر قسمت زمانبندی را از جدول حذف می کنیم

با دسته بندی چهارتایی به فاصله دوتا 498623 توالی ایجاد می شود

```
ratings_data_movies = ratings_data[["user_id", "movie_ids"]].explode(
    "movie_ids", ignore_index=True
)
ratings_data_rating = ratings_data[["ratings"]].explode("ratings",
ignore_index=True)
ratings_data_transformed = pd.concat([ratings_data_movies, ratings_data_rating],
axis=1)
ratings_data_transformed = ratings_data_transformed.join(
    users.set_index("user_id"), on="user_id"
)
ratings_data_transformed.movie_ids = ratings_data_transformed.movie_ids.apply(
    lambda x: ",".join(x)
)
ratings_data_transformed.ratings = ratings_data_transformed.ratings.apply(
    lambda x: ",".join([str(v) for v in x])
)

del ratings_data_transformed["zip_code"]

ratings_data_transformed.rename(
    columns={"movie_ids": "sequence_movie_ids", "ratings": "sequence_ratings"},
    inplace=True,
)
```

در ابتدا یک DataFrame جدید ایجاد می کنیم فقط شامل ستون هایی از ایدی کاربر و ایدی فیلم است سپس ستون ایدی فیلم را گسترش می دهیم به طوری که هر ردیف شامل یک فیلم باشد. همچنین یک DataFrame جدید دیگر از رتبه بندی ها ایجاد میکنیم و آن را گسترش می دهیم بعد از آن دو دیتافریم ایجاد شده را در ستون ها به هم متصل می کنیم و یک دیتافریم جدید ایجاد میکنیم

به طور کلی، این کد یک مجموعه داده رتبه بندی خام را می‌گیرد، آن را به دنباله‌ای از شناسه‌های فیلم و رتبه بندی‌های مربوط به هر کاربر تبدیل می‌کند و آن را برای استفاده در ساخت یک سیستم توصیه آماده می‌کند.

```
random_selection = np.random.rand(len(ratings_data_transformed.index)) <= 0.85
train_data = ratings_data_transformed[random_selection]
test_data = ratings_data_transformed[~random_selection]

train_data.to_csv("train_data.csv", index=False, sep="|", header=False)
test_data.to_csv("test_data.csv", index=False, sep="|", header=False)
```

برای دیتاها دو دسته ایجاد می کنیم. برای دسته اول داده های اموزشی یا train data را ایجاد می کنیم و 85% داده هارا برای اموزش به آن اختصاص می دهیمو 15% باقی مانده را به داده های تست(test_data) اختصاص می دهیم و در نهایت در فایل با فرمت csv ذخیره می کنیم.

```
CSV_HEADER = list(ratings_data_transformed.columns)

CATEGORICAL_FEATURES_WITH_VOCABULARY = {
    "user_id": list(users.user_id.unique()),
    "movie_id": list(movies.movie_id.unique()),
    "sex": list(users.sex.unique()),
    "age_group": list(users.age_group.unique()),
    "occupation": list(users.occupation.unique()),
}

USER_FEATURES = ["sex", "age_group", "occupation"]

MOVIE_FEATURES = ["genres"]
```

از دیتافریمی که ایجاد کردیم نام ستون هارا برمیداریم و درون یک دیتافریم جدید میگذاریم و درون آن ها مقادیر غیرتکراری می گذاریم هدف از اینکار دسته بندی کردن است

```python
def get_dataset_from_csv(csv_file_path, shuffle=False, batch_size=128):
    def process(features):
        movie_ids_string = features["sequence_movie_ids"]
        sequence_movie_ids = tf.strings.split(movie_ids_string, ",").to_tensor()

        # The last movie id in the sequence is the target movie.
        features["target_movie_id"] = sequence_movie_ids[:, -1]
        features["sequence_movie_ids"] = sequence_movie_ids[:, :-1]

        ratings_string = features["sequence_ratings"]
        sequence_ratings = tf.strings.to_number(
            tf.strings.split(ratings_string, ","), tf.dtypes.float32
        ).to_tensor()

        # The last rating in the sequence is the target for the model to predict.
        target = sequence_ratings[:, -1]
        features["sequence_ratings"] = sequence_ratings[:, :-1]

        return features, target

    dataset = tf.data.experimental.make_csv_dataset(
        csv_file_path,
        batch_size=batch_size,
        column_names=CSV_HEADER,
        num_epochs=1,
        header=False,
        field_delim="|",
        shuffle=shuffle,
    ).map(process)

    return dataset
```

این کد تابع get_dataset_from_csv را تعریف می کند که یک فایل CSV حاوی داده های رتبه بندی فیلم را بارگیری می کند و یک شی TensorFlow Dataset را برمی گرداند. انتظار می رود فایل CSV دارای فیلدهای user_id,sequence_movie_ids,sequence_rating باشد که با | از هم جدا شده اند هر ردیف نشان دهنده یک کاربر و ترتیب رتبه بندی فیلم‌ها است.

فیلد sequence_movie_ids فهرستی از شناسه های فیلم است که با کاما از هم جدا شده‌اند و فیلد sequence_ratings فهرستی از رتبه بندی‌های متناظر با کاما است.

تابع process در get_dataset_from_csv تعریف شده است و برای پیش پردازش ویژگی ها و مقادیر هدف مجموعه داده استفاده می شود. به طور خاص، رشته های sequence_movie_ids و sequence_ratings را به تانسورهای مجزا تقسیم می‌کند و آخرین مورد از هر یک، مقدار هدف برای پیش بینی مدل است. ویژگی های به دست آمده و تانسورهای هدف توسط process برگردانده می شوند.

سپس تابع make_csv_dataset برای ایجاد یک شی TensorFlow Dataset از فایل CSV استفاده می شود. آرگومان column_names نام فیلدها را در فایل CSV مشخص می کند و header=False نشان می دهد که فایل ردیف سرصفحه ندارد. آرگومان field_delim جداکننده مورد استفاده برای جداسازی فیلدهای فایل را مشخص می کند. آرگومان num_epochs روی 1 تنظیم شده است تا نشان دهد که مجموعه داده فقط باید یک بار تکرار شود.

در نهایت، متد map روی مجموعه داده فراخوانی می شود تا تابع process را به هر عنصر اعمال کند و مجموعه داده از پیش پردازش شده را برمی گرداند. آرگومان shuffle تعیین می‌کند که آیا مجموعه داده ها باید قبل از تکرار دوباره مخلوط شوند یا خیر، و batch_size اندازه دسته‌هایی را مشخص می‌کند که مجموعه داده باید به آن‌ها تقسیم شود.

به طور خلاصه، این کد تابعی را تعریف می کند که یک فایل CSV از داده های رتبه بندی فیلم را بارگیری می کند و آن را در یک شی TensorFlow Dataset پیش پردازش می کند، که می تواند برای آموزش مدل های یادگیری ماشین استفاده شود.

Create model inputs:

```
def create_model_inputs():
    return {
        "user_id": layers.Input(name="user_id", shape=(1,), dtype=tf.string),
        "sequence_movie_ids": layers.Input(
            name="sequence_movie_ids", shape=(sequence_length - 1,),
dtype=tf.string
        ),
        "target_movie_id": layers.Input(
            name="target_movie_id", shape=(1,), dtype=tf.string
        ),
        "sequence_ratings": layers.Input(
            name="sequence_ratings", shape=(sequence_length - 1,),
dtype=tf.float32
        ),
        "sex": layers.Input(name="sex", shape=(1,), dtype=tf.string),
        "age_group": layers.Input(name="age_group", shape=(1,), dtype=tf.string),
        "occupation": layers.Input(name="occupation", shape=(1,),
dtype=tf.string),
    }
```

این کد تابعی به نام create_model_inputs() تعریف می کند که فرهنگ لغت تانسورهای ورودی را برای یک مدل یادگیری ماشینی برمی گرداند.

ورودی ها عبارتند از:

user_id: تانسور رشته ای شکل (1،) که نشان دهنده شناسه کاربر است.

sequence_movie_ids: یک تانسور رشته ای شکل (طول_توالی - 1،) که نشان دهنده دنباله شناسه های فیلم است که کاربر در گذشته رتبه بندی کرده است (به استثنای فیلم مورد نظر).

target_movie_id: تانسور رشته ای شکل (1،) که نشان دهنده شناسه فیلمی است که مدل سعی می کند امتیاز آن را پیش بینی کند.

sequence_ratings: یک تانسور شناور شکل (sequence_length - 1) که نشان دهنده دنباله رتبه هایی است که کاربر به فیلم ها در sequence_movie_ids داده است.

جنسیت: تانسور رشته ای شکل (1) که جنسیت کاربر را نشان می دهد.

سن_گروه: تانسور رشته ای شکل (1،) که نشان دهنده گروه سنی کاربر است.

occupation: تانسور رشته ای شکل (1،) که نشان دهنده شغل کاربر است.

هر تانسور ورودی با استفاده از تابع layers.Input از TensorFlow Keras API ایجاد می‌شود، که یک تانسور مکان نگهدار ایجاد می‌کند که می‌تواند به عنوان ورودی مدل Keras استفاده شود. آرگومان های name، shape و dtype برای تعیین نام، شکل و نوع داده هر تانسور ورودی استفاده می شود.

Encode input features:

متد encode_input_features به صورت زیر کار میکند:

از هردو ویژگی های کاربر وفیلم استفاده می کند.برای ویژگی های کاربر از layers.Embedding استفاده می کند و هر ویژگی را در یک لایه قرار می دهد. سایز این لایه های Embedding برابر با جذر اندازه واژگان ویژگی هاست.

جاسازی های حاصل برای هر ویژگی به هم متصل می شوند تا یک تانسور ورودی واحد را تشکیل دهند.

برای ویژگی های فیلم ها ابتدا هر فیلم را در یک لایه با layeres.Embedding رمزگزاری می کنیم .ابعاد Embedding به جذر تعداد فیلم‌های مجموعه داده تنظیم می‌شود.

علاوه بر این یک بردار multi-hot genres برای هر فیلم با بردار Embedding آن تطبیق داده می شود.

اتصالات انجام شده از یک لایه غیرخطی layers.Dense با اندازه ابعاد قبلی عبور داده می شود

سپس Embeddingهای موضعی تولید شده به توالی های قبلی اضافه می شوند و Embedding ها با رتبه بندی های توالی ضرب می شوند

این متد چند عنصر از دو عنصر را برمی‌گرداند:

encoded_transformer_features و encoded_other_features
encoded_transformer_features ویژگی‌های فیلم کد گذاری شده و ویژگی‌های کاربر هستند که آماده انتقال به معماری ترانسفورماتور هستند. encoded_other_features هر ویژگی اضافی است که در معماری ترانسفورماتور گنجانده نشده است.

به طور کلی، این روش برای آماده سازی ویژگی های ورودی برای معماری ترانسفورماتور برای ایجاد توصیه های دقیق برای فیلم ها ضروری است. با رمزگذاری ویژگی ها به این روش، ترانسفورماتور می تواند هم ترجیحات کاربر و هم ویژگی های فیلم را برای ارائه توصیه های دقیق در نظر بگیرد.

Create a BST model:

این مدل شامل یک لایه multi-header attention و transformer block است.همچنین حاوی لایه های fully connected with batch normalization,LeakyRelu activation,dropout است.

این مدل به گونه ای طراحی شده است که ورودی‌ها را از یک سیستم توصیه با قابلیت گنجاندن ویژگی‌های کاربر و فیلم دریافت کند و در نهایت، یک پیش بینی برای امتیازی که کاربر به یک فیلم می دهد ارائه می دهد.

include_user_features و include_movie_features متغیرهای منطقی هستند که تعیین می‌کنند آیا هر نوع ویژگی ورودی در مدل گنجانده شود یا خیر.

hidden_units لیستی از اعداد صحیح است که تعداد نورون ها را در هر لایه Fully-connected نشان می دهد

dropout_rate عددی بین 0 و 1 است که نشان دهنده نسبت نورون‌هایی است که به طور تصادفی در طول آموزش از بین می روند.

num_heads یک عدد صحیح است که نشان دهنده تعداد attention heads در لایه multi-headed attention است

encode_input_features تابعی است که ویژگی های ورودی را رمزگذاری و پیش پردازش می کند.

Transformer block شامل یک dropout layer، یک residual connection با یک لایه Add و normalization layer است .تابع فعال سازی LeakyReLU دارای شیب منفی 0.3 است .لایه Flatten خروجی Transformer block را قبل از وارد شدن به لایه های Fully-connected صاف می کند .اگر ویژگی‌های دیگر نیز اضافه شود، آنها به ویژگی‌های ترانسفورماتور مسطح اضافه می‌شوند.

لایه های Fully-connected هر کدام از یک لایه Dense، batch normalization، LeakyReLU activation و dropout تشکیل شده‌اند. لایه خروجی یک لایه Dense با یک واحد است که نشان دهنده رتبه پیش بینی شده فیلم است.

در نهایت از این مجموعه کدها برای اموزش استفاده می کنیم. می توان با تغییرات قسمت های مختلف این اموزش را انجام داد همچنین می توان قسمت های جدید به ویژگی ها اضافه کرد برای مثال سال انتشار فیلم, کدپستی ,ژانر جنسی و ...