Mobile App made with flutter

Features :

- Fetch popular movies from Tmdb API
- Show movies description
- Add user rated movies
- Run TensorFlow lite model to get recommendations

Classes and Functions

1. MovieAPIProvider Class

This is used to fetch data from Tmdb API over a http response with API key provided .

Variable description:

_apiKey: Stores the private key to authenticate Tmdb API .

_baseUrl: The http URL of the Tmdb API, need to append API key with required parameters .

response: Stores the respone given by the http client on passing the _baseUrl with appended extra attributes.

Function description:

`fetchMovieList()`: gets a Json response of popular movie list .

`fetchMovieDetail()`: gets a Json response of movie details consisting of movie description , poster link and Imdb rating given movie id .

`fetchSimilarMovies()`: gets a Json response of movie that are similar to given movie Id and decodes it into an *ItemModel.*

Code Snippet :

```
class MovieApiProvider {
  http.Client client = http.Client();
  final _apiKey = '9c9576f8c2e86949a3220fcc32ae2fb6';
  final _baseUrl = "http://api.themoviedb.org/3/movie";

  Future<ItemModel> fetchMovieList() async {
    final response =
        await client.get(Uri.parse('$_baseUrl/popular?api_key=$_apiKey'));
    print(response.body.toString());
    if (response.statusCode == 200) {
      print("Inside 200 status code");
      return ItemModel.fromJson(json.decode(response.body));
    } else {
      print("Status code : ${response.statusCode}");
      throw Exception('Failed to load movies list');
    }
  }

  Future<MovieDetail> fetchMovieDetail(int movieId) async {
    final response =
        await client.get(Uri.parse("$_baseUrl/$movieId?api_key=$_apiKey"));
    print(response.body.toString());
    if (response.statusCode == 200) {
      return MovieDetail.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to retrieve Movie Detail');
    }
  }

  Future<ItemModel> fetchSimilarMovies(int movieId) async {
    final response = await client
.get(Uri.parse("$_baseUrl/$movieId/similar?api_key=$_apiKey"));
    print(response.body.toString());

    if (response.statusCode == 200) {
      return ItemModel.fromJson(json.decode(response.body));
    } else {
      throw Exception('Error retrieving similar movies details');
    }
  }
}
```

2. MovieDetail Class

This class describes the structure of movie details based on the Json structure response from the API which consists of movie name , description , Imdb rating ,poster path .

Variable description:

_adult: This variable is a Boolean value to show if the movie is R rated or not.

_backdropPath: this string variable stores the poster id which is used to get Image from the network

_genres: this stores the types of movies such as comedy, action, etc with their ids .

_homepage: Stores the official website URL of the movie.

_id: The identification number of the movie is stored in this variable .

Function description:

`fromJson()`: converts Json response of movie details into the variables of the class object.

`toJson()`: creates a Json variable from the class object variables

Code Snippet :

```
class MovieDetail {
  bool _adult;
  String _backdropPath;
  List<Genres> _genres;
  String _homepage;
  int _id;

  MovieDetail(
      {bool adult,
      String backdropPath,
      List<Genres> genres,
      String homepage,
      int id,
      }) {
    this._adult = adult;
    this._backdropPath = backdropPath;
    this._genres = genres;
    this._homepage = homepage;
    this._id = id;
  }

  bool get adult => _adult;
  set adult(bool adult) => _adult = adult;
  String get backdropPath => _backdropPath;
  set backdropPath(String backdropPath) => _backdropPath = backdropPath;

  List<Genres> get genres => _genres;
  set genres(List<Genres> genres) => _genres = genres;
  String get homepage => _homepage;
  set homepage(String homepage) => _homepage = homepage;
  int get id => _id;
  set id(int id) => _id = id;


  MovieDetail.fromJson(Map<String, dynamic> json) {
    _adult = json['adult'];
    _backdropPath = json['backdrop_path'];
    if (json['genres'] != null) {
      _genres = new List<Genres>();
      json['genres'].forEach((v) {
        _genres.add(new Genres.fromJson(v));
      });
    }
    _homepage = json['homepage'];
    _id = json['id'];
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['adult'] = this._adult;
    data['backdrop_path'] = this._backdropPath;
    if (this._genres != null) {
      data['genres'] = this._genres.map((v) => v.toJson()).toList();
    }
    data['homepage'] = this._homepage;
    data['id'] = this._id;
    return data;
  }
}
```

3. ItemModel Class

This is used to define the structure of Json response of the similar movies from the API

Variable description:

_pages: the current page number of the fetched results

_total_results: the total number of movies which are similar to given movie.

_total_pages: total number of pages available to fetch results .

Function description:

`fromJson()`: converts the Json respone into class object variables.

Code Snippet :

```
class ItemModel {
  int _page;
  int _total_results;
  int _total_pages;
  List<_Result> _results = [];

  ItemModel.fromJson(Map<String, dynamic> parsedJson) {
    print(parsedJson['results'].length);
    _page = parsedJson['page'];
    _total_results = parsedJson['total_results'];
    _total_pages = parsedJson['total_pages'];
    List<_Result> temp = [];
    for (int i = 0; i < parsedJson['results'].length; i++) {
      _Result result = _Result(parsedJson['results'][i]);
      temp.add(result);
    }
    _results = temp;
  }

  List<_Result> get results => _results;

  int get total_pages => _total_pages;

  int get total_results => _total_results;

  int get page => _page;
}
```

Functions used to generate recommendations

1. getId Function

This is used to return the user Id corresponding to the trained dataset based on the user added movie list. It does this by finding closest matching user with minimum root mean square value from the rated movies list.

Parameter description:

`mov:` it stores the user rated movies with their Id (int) and rating (double)

`count:` the maximum numbers of random users to check for similarity.

Variable description:

input0: the user id from the trained dataset

input1: the movie id from the database

output0: predicted rating on the scale of 0 to 5 .

selectedId: the id with minimum root min square value

interpreter: the TensorFlow lite runtime interpreter to run the tflite model.

Code Snippet :

```
double getId(Map<int, double> mov, int count) {
  if (interpreter == null) return -1;
  var input0 = [9.0];
  var input1 = [0.0];
  var output0 = [1].reshape([1, 1]);
  var outputs = {0: output0};
  List<List<double>> inputs;
  double min = double.infinity;
  double temp;
  double selectedId;
  for (int i = 1; i <= count; i++) {
    temp = 0;
    input0 = [Random().nextInt(138493).toDouble() + 1];
    mov.forEach((key, value) {
      input1 = [key.toDouble()];
      inputs = [input0, input1];
      interpreter.runForMultipleInputs(inputs, outputs);
      if (output0[0][0].isNaN) output0[0][0] = -3.5;
      temp += (output0[0][0] + 3.5 - value) * (output0[0][0] + 3.5 - value);
    });
    temp = sqrt(temp / mov.length);
    if (temp < min) {
      min = temp;
      selectedId = input0[0];
    }
  }
  return (selectedId);
}
```

2. output Function

This is used update the recommendations list by getting user Id from getId function and running the interpreter on the unseen movies to get highest rated movies and then add top 70 to the recommendations list.

Parameter description:

`mov:` it stores the user rated movies with their Id (int) and rating (double)

`count:` the maximum numbers of random users to check for similarity.

Variable description:

input0: the user id from the trained dataset

input1: the movie id from the database

output0: predicted rating on the scale of 0 to 5 .

id: the id of the user closest to given

out: the TensorFlow lite runtime interpreter to run the tflite model.

sort: the list of movie Ids with highest rating movies in descending order.

Code Snippet:

```
void output(Map<int, double> mov, int count) async {
  recommendations.clear();
  double id = getId(mov, count);
  if (id == -1) return;
  var input0 = [id];
  var input1 = [0.0];
  var output0 = [1].reshape([1, 1]);
  var outputs = {0: output0};
  List<List<double>> inputs;
  Map<int, double> out = new Map();
  Map<int, String> movi = {};
  movi.addAll(movies);
  mov.forEach((key, value) {
    movi.remove(key);
  });
  movi.forEach((key, value) {
    input1 = [key.toDouble()];
    inputs = [input0, input1];
    if (interpreter != null)
      interpreter.runForMultipleInputs(inputs, outputs);
    if (output0[0][0].isNaN) output0[0][0] = -3.5;
    out[key] = (output0[0][0]);    });
  List<int> sort = out.keys.toList()
    ..sort((k1, k2) => out[k1].compareTo(out[k2]));
  sort = sort.reversed.toList();
  for (int i = 0; i < 70; i++) recommendations.add(sort[i]);
}
```

Creating the TensorFlow model in python

Introduction

We have trained a neural network with TensorFlow on grouplens movies dataset . The code for the training is available at Colab .

Dataset Summary

This dataset (ml-20m) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on March 31, 2015, and updated on October 17, 2016 to update links.csv and add genome-* files.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in six files, `genome-scores.csv`, `genome-tags.csv`, `links.csv`, `movies.csv`, `ratings.csv` and `tags.csv`. More details about the contents and use of all these files follows.

Model training loss graph