MOVAStore

Aim: Increase clickthrough rates (CTR) by delivering more relevant and effective recommendations to users.

SYSTEM DESIGN

Implement a machine learning algorithm whose success is defined by a user's clickthrough on a recommended app (ie recommender system). The algorithm will take some input parameters based on the user and the app, score all other apps based on how likely a clickthrough is, and return a list of highest-scoring apps as recommendations.

Input Parameters

In recommender systems there are generally two main categories - content-based and collaborative filtering systems. Different parameters are used for each system and the overall recommender system would use a combination of both systems.

Content-based filtering: uses information about the user/items to make predictions
Collaborative filtering: uses past interactions between users and items to make predictions

Content-based Filtering Parameters	Collaborative Filtering Parameters
- **App keywords, description, tags, category, location**: To recommend similar apps - **App screenshots, logo**: Image processing can be used to recommend apps with a similar aesthetic - **App's number of downloads**: a more popular app should be more highly recommended - **App's number of clickthroughs**: an app with more clickthroughs (ie successes) should be more highly recommended - **Single user's past downloads, ratings and comments**: Indicates user engagement, recommend similar apps	- **All users' past downloads, ratings, comments and clickthroughs**: Based on other users interactions with apps, make recommendations (eg users who downloaded this app also liked…), or recommend apps that already have high CTR

Weighting the input parameters

The parameters are not all equally important, and the AI will need to be trained to find the relative importance of each parameter.

https://mlrose.readthedocs.io/en/stable/source/tutorial3.html

Output

The output should be a list of apps sorted by score, highest scoring displayed first. Size of the array can be either decided by score (ie only those scoring above some number get displayed) or predefined (eg top 10).

OTHER IMPROVEMENTS

Additional Input Parameters

Some additional data could be collected in the sign up process to develop a profile of the user (eg age, gender, location or using OAuth to get relevant data) that could improve the success of the algorithm.

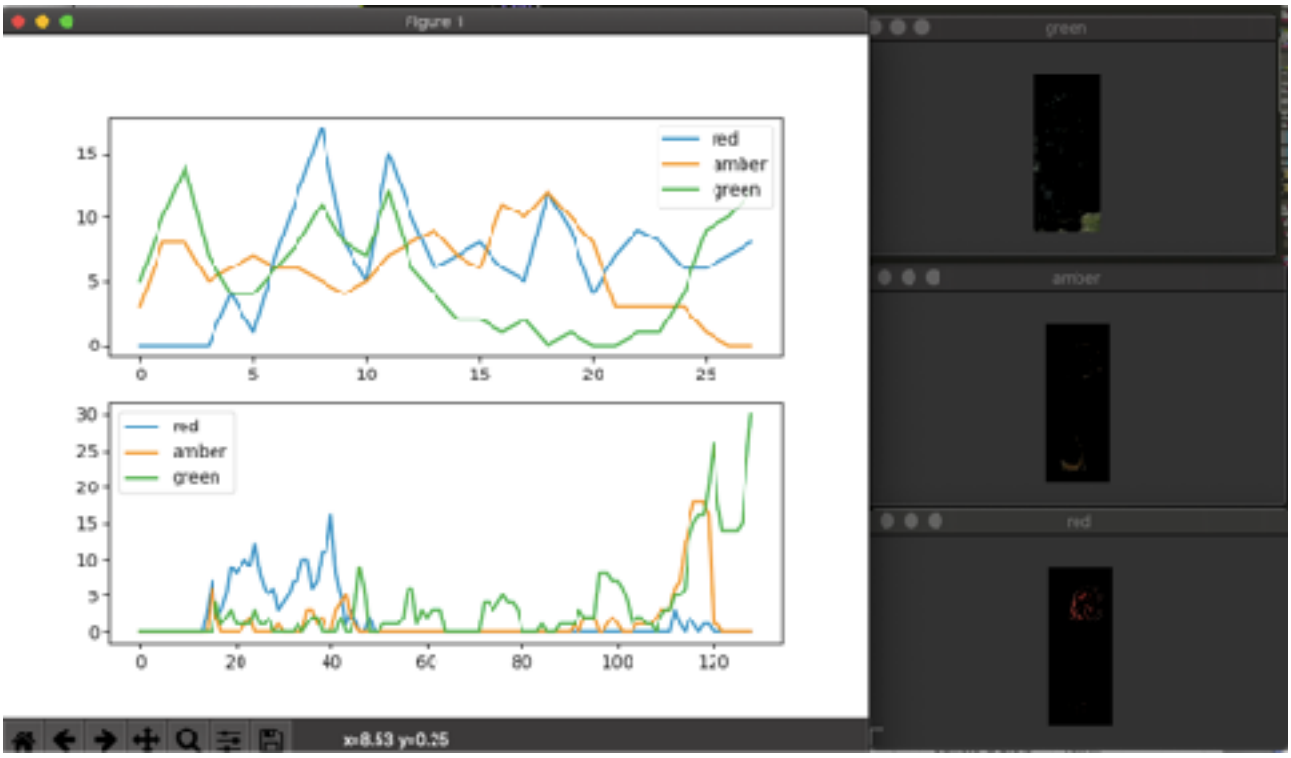
Identifying Traffic Signals

Code can be found in traffic-light.py in this repository.

I wanted to try to do this without using a machine learning library first, so that's what the code is - essentially it takes the image of a closely cropped (vertical) traffic light, applies masks to filter out certain colours of the image, and reads the maximum of the colours and their position.

I tried it on 4 different images of traffic lights (also in the repository), and it fails on one of the images (a green light where the traffic light casing is yellow).

To improve accuracy, I think the next steps would for this particular algorithm would be to optimise the colour ranges for the masks, as well as take into account the overall shapes of the peaks, as well as the position of peaks in the y-direction. Currently the algorithm only takes into account the maximum in the x-direction - but there's a lot more information found in the overall shapes of the graphs than just the position of the maximum. I didn't have time to figure out the best way to parse it.



In the image above: on the right, resulting images after green, amber and red masks are applied to the original cropped image, and on the left, a graph of the intensity of each one in the x direction and y direction respectively. In a more complete version of this algorithm, the highest intensity colour, red, in the x direction would be checked against the increased intensity in the y direction (in the 20-24 region on the lower graph), and would be able to tell with higher probability that the traffic light was red.

Of course this could be run through a machine learning algorithm, which is likely to be more accurate given a large enough data set. If the end goal is autonomous driving, AI is probably more helpful for this because it has to take into account all the different angles a car might “see” a traffic light from, where the algorithm I wrote only takes into account a fairly face-on point of view.

I didn’t have the time or data set to implement this, but if I did, I’d start by following this tutorial https://towardsdatascience.com/train-image-recognition-ai-with-5-lines-of-code-8ed0bdd8d9ba, which uses the Image AI library in Python.

Python Programming Competency

I’d rate myself 3/5 - very comfortable with basics (operations on data structures, recursive functions, class inheritance etc), but I’ve only just started learning about more complex things such as threading/concurrency, Python GIL, unit testing, multiple inheritance etc

I have some experience with pandas, and after the assignment, I’ve some experience with numpy, matplotlib and cv2 now too.