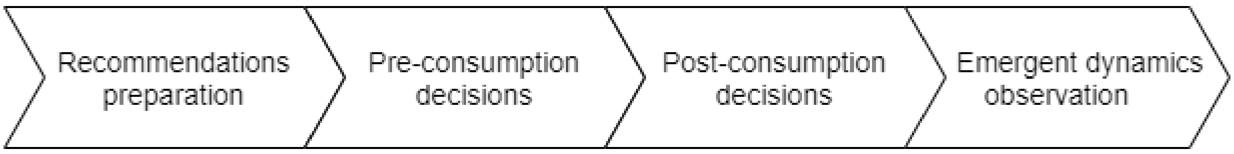
Table of content

Summary

This agent-based model demonstrates the consequences of various recommendation strategies in a multistakeholder platform. Focusing only to satisfy consumers in delivering the recommendations affects other stakeholders' interests, such as the service provider. Similarly, Delivering the recommendations to maximize the profit affects consumers' trust in the service provider. We are interested in understanding this phenomenon deeply and its long-term effects on consumers' trust in the recommendation service provider and the profit gained by the provider when a specific strategy is applied. The model integrates various consumers decisions and experiences sharing via naive social media space. Two types of agents are used in this model:

- Recommendation service provider: Prepares and sends personalized recommendations to the consumers
- Consumer: Receives the recommendations and make further decisions

General model workflow


Recommendations preparation
Pre-consumption decisions
Post-consumption decisions
Emergent dynamics observation

Requirements

We tested the code on a local machine wit Windows 10, Python>=3.7, 16GB, and core i7. The code also was tested on a remote machine with docker, Ubuntu 20.04.2 LTS x86_64, Python docker image, 30GB, and Intel Xeon E5645 (12) @ 2.4.
In windows, it is recommended to install Anaconda last version, which comes with Python 3 and supports scientific packages installation.

The following packages are used in our model, and they are already included in the `requirments.txt`:

- numpy
- matplotlib

- pandas
- scipy
- surprise
- mesa
- pyyaml

Installation

The installation is possible on a local or remote machine. We assume the remote machine has a docker installed. If not docker, you can follow the instructions of local installation.

Local installation

Download and install AnaConda (Indevedual Edition)

Create a virtual environment

```
conda create -n myenv python=3.6
```

Activate the virtual environment

```
conda activate myenv
```

More commands on using virtual environment in Anaconda is available here
Install the required packages by running: \

```
pip install -r requirements.txt
```

If you face errors in insatlling **surprise** package on Windows, run:

```
conda install -c conda-forge scikit-surprise
```

Remote installtion

Before building an image of the simulation code, you need to pull the docker image of Python from here. Use the following code to pull the image:

```
docker pull python
```

On the remote machine, build docker image using the provided `Dockerfile`, change to the root directory and run: `docker build -t <simulation_image_name> .`

Running the model

To run the simulation locally, change directory to src, and run:
`python run.py`
OR
Run the simulation on a docker container, which will create a volume to store the results directory with data generated by the simulation:
`docker run -dit --rm -v ${PWD}/results:/results --name <my_container> <simulation_image_name>`

File structure

The simulation is built in Mesa, an agent-based simulation framework in Python.

```
/
├── data/
│   ├── dataset/                  <- MovieLens dataset
│   │   ├── movies.csv
│   │   └── ratings.csv
│   ├── recdata/                  <- Recommendation algorithm output saved in a
pickle format
│   │   ├── consumers_items_utilities_predictions.p
│   │   ├── consumers_items_utilities_predictions_popular.p
│   │   └── SVDmodel.p
│   └── trust/                    <- Initial data for consumers trust
│       └── beta_initials.p
├── Dockerfile
├── figures/                      <- Figures that show model results
│   ├── modelgeneralflow.png
│   ├── time-consumption_probability.png
│   ├── time-total_profit.png
│   └── time-trust.png
├── README.md
├── requirements.txt
├── results/                      <- Store model results
├── src/
  ├── __init__.py
  ├── config.yml                  <- Model settings
  ├── consumer.py                 <- Contains all propoerties and behaviors of
consumer agent
  ├── mesa_utils/
  │   ├── __init__.py
  │   ├── datacollection.py
  │   └── schedule.py
  ├── model.py                    <- Contains the model class, which manages
agents creation, data sharing, and model output collection
  ├── plots.py                    <- Ploting module for data analysis
  ├── read_config.py
  ├── run.py                      <- Launchs the simulation
  ├── service_provider.py         <- Contains all properties and behaviors of
Service provider agent
```

```
    ├── test.py
    └── utils.py                    <- An auxiliary module for extra helpful
functions
```

Dataset to compute consumers items' utilities

We used Movielens dataset, the small version (1 MB), which contains movie ratings for multiple users, more details. The following shows the content of `ratings.csv`.

userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
1	6	4	964982224
1	47	5	964983815
1	50	5	964982931

The dataset is used to predict consumer items utilities, and to initialize the model.

Configuration file

Before running the simulation, make sure you set up desired values for model parameters in the `config.yml`. The configuration file is in yaml format. The file has two sections: the first section is model_input, which contains input data to the model; the second section includes model parameters. The table below gives a brief description of each parameter.

Parameter	Description	Value
time	Number of timesteps	1000
iterations	Number of runs of each simulation setting	5
expectation_threshold_quantile	Minimum expectation threshold for consumer. We use a quantile of the predicted ratings per consumer	0.7
strategy	Recommendation strategy to be used by the service provider	Five strategies will be ran by seperately Mesa BatchRunner
recommendation_length	Number of items to be recommended	10
error	Drawn from normal distribution, used to compute the actual consumer utility of a consumed item	mu=0,sd=0.3

Parameter	Description	Value
social_media_on	1:social media exists,0: social media does not exist	1
update_expectation_threshold	Time interval of updating the consumer expectation threshold	100
trust_update_param	Parameter used to update trust after consumption, we use the distance between the actual item utility and consumer expectation threshold	"euclidean"
feedback_likelihood	Probability of consumer submitting feedback to the service provider	0.1
dropout_threshold	Minimum trust threshold for consumer to leave the platform. This value is multiplied by the consumer' initial trust	0.8

Note: The simulation may take a long time based on the predefined time and iterations in the `config.yml`. To run a light simulation, choose smaller values for time and iterations.

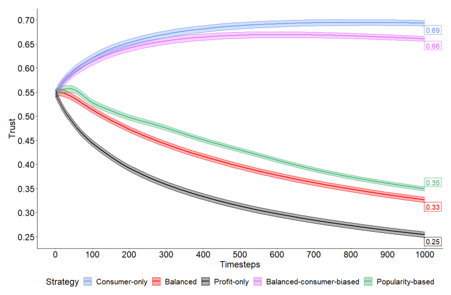
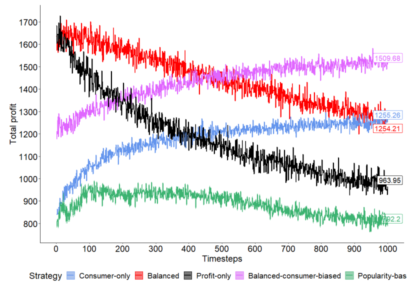
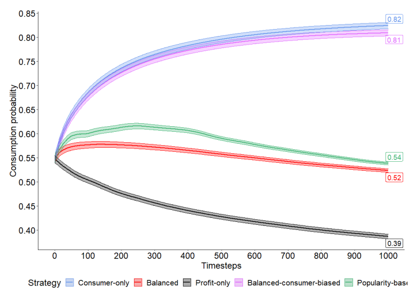
Results

Each execution of the model generates a unique folder inside the results folder. The collected data from the simulation contains various CSV files, scenarios.json, and png plots. The combination of sensitive parameters with different specified values are saved in the `scenarios.json` file.

We test five different scenarios while varying recommendation strategies and fixing other model variables.

The following results are taken from simulating 1000 timesteps and repeat the simulation three times. The simulation comprises one service provider agent and 610 consumer agents, and consumers can share their experiences on social media. We observe the following:

- Consumer trust in the service provider
- Service provider total profit
- Consumption probability

Consumers trust over time	**Total profit over time**	**Consumption probability over time**
		

Trust
Timesteps
Strategy
Consumer-only
Balanced
Profit-only
Balanced-consumer-biased
Popularity-based

Total profit
Timesteps
Strategy
Consumer-only
Balanced
Profit-only
Balanced-consumer-biased
Popularity-bas

Consumption probability
Timesteps
Strategy
Consumer-only
Balanced
Profit-only
Balanced-consumer-biased
Popularity-base