Data Flow Diagrams

[Old]

Duplicate Reviews

minHash   Similarity

Read   Write Read   Write

JSON   CSV   CSV
Reviews   minHash   Duplicate Result

DataStore

Read   Read

Report Generator

Review Count   Rating Distribution   Incentivized Reviews   Graphs

Write

HTML
Web Browser

[New]

Duplicate Reviews

minHash.py   similarity.py   new algorithm

Admin — run local, static command

json → json_to_database.py → SQLite Database

Fake Detection

incentivized.py   anomaly.py → results.html

User — Chooses product to analyze (already in the dataset) → GET message from website → urls.py → views.py

Use Case Examples

Admin

Use Case 1:
Admin downloads a json file from Amazon dataset site.
Admin runs script 'json_to_database.py' in Django using the admin command 'python manage.py json_to_database' and waits for it to finish running.
Admin checks admin django site to verify data was uploaded correctly.

Use Case 2:
Conditions:
There should be data in the 'Review', 'User', and 'Product' tables in the database. See use case 1 for how to.

Admin runs script 'minHash.py' using the admin command 'python manage.py minHash <product asin>' (Where product asin is replaced with the product you wish to analyze).
System executes minHash algorithm on all reviews for the given product asin.
Admin waits for script to finish running and checks to see that the 'minHash' attribute is filled in for the reviews of a given product on the Django admin site (or queried through the terminal).
Admin runs script 'similarity.py' using the admin command 'python manage.py similarity <product asin>'.
System executes cross checks all reviews for similar bigram values and calculates a given duplicate ratio.
Admin waits for script to finish running and checks to see that the 'duplicateRatio' attribute in the Product table is filled in for the given product on the Django admin site.

Users

Use case 1:
Conditions:
User is on the home page of the website.

User clicks on the 'More' tab which reveals links to different subsites.
User clicks on the 'Search Repository' link.
User chooses product to analyze.
Browser sends GET request to backend. urls.py maps a given url to a specific view given the webpage and product asin.
views.py uses the scripts 'incentivized.py' and 'anomaly.py' as detection algorithms to calculate metrics for an overall 'fake score'.
The database is updated and scores are pulled.
The system calls to render an html file, which can use these variables through the use of Django's template API.
An overall score is given depending on the given values.
The system returns the request (POST) and shows the results webpage.