Cattywampus' Solution to the Hut Challenge 2013

Yangfan Zhang (Imperial), Derek Khu (Oxford), Daniel Wong (Cambridge)

Written by Yangfan Zhang
with minor edits by Daniel Wong

High-level Description

- An ensemble of multiple machine learning methods, including:
    a. a random forest classifier that mines features from the orders.
    b. performing random walks on the product-customer graph.
    c. item-based collaborative filtering*
    d. matrix factorisation using LIBMF*
    * *not used in highest-scoring submission.*
- Predictions for each customer
    a. Cold start
        - For new customers with no order history / < 6 predictions made
            - Padded with the top 6 products bought by customers on their first order. [200,392,500,316,47]
    b. use each method to calculate a probability of each product being purchased by this customer
    c. combine the probabilities with optimised weightage
    d. rank the combined probabilities to output the top 6 products

Details

- Random Forest classifier
    - X and Y are extracted from the data
    - **splitXY.py**
        - split criteria - multiple possible
            1. SplitXYByTime - split by time - everything before a specified time used for X, everything afterwards used for Y
            2. SplitXYByPer - split by a specified percentage, e.g. a 75%-25% split for X-Y respectively
        - **X**: a list of lists, with each internal list corresponding to one customer and showing the orders available within the first split. Used to mine data
        - **Y**: a list of lists, with each internal list corresponding to one customer and listing the products that the customer bought within the second split of the data
        - options: can specify filters for minimum orders in X and minimum product count in Y
    - List of prominent features
        - Time since last order
        - Time since first order

        - Number of orders
        - Number of products
        - Average Interval Between Transactions
        - Individual product count - product 200
        - Average orders per transaction
        - Number of transactions
        - Individual product count in the first 12 months - product 200
        - Average interval between product purchase - product 200
        - Country of order
- Random walks on product-customer graph
    - Graph: a bipartite graph of customers and products. The weight of an edge between a customer c and a product p is the quantity of product p bought by customer c.
        - Graph (customer_product_counts.csv)
            1. extracted using an SQL query (can be found in queries.sql)
- Item-based Collaborative Filtering
    - for each item
        - calculate the cosine similarity with other products and sort them by the similarity value
    - for each customer
        - for each product the customer bought
            - ➢ find out the top N most similar products
        - return the 6 products with highest accumulative similarity
    - data file (the partial user-product matrix): interim/products_by_test_customers_cnts.csv

Source File Description

File / Folder	Content
data/	Original train set and generated data files ● Train.csv – given train set ● all_cusomers.csv - set of all customer IDs ● publicChallenge.csv - set of customer IDs in the give test set
db/	SQL queries for data extraction
interim/	Files of orders, product IDs and customer IDs of different sizes
runs/	Parameters used in Makefile, and solution in each run
submitted/	Submitted solutions (to be updated)
Makefile	Master copy of make file

common.py	Methods for file I/O
customer.py	Methods for processing and analysing orders and customer ids
features.py	Methods to create features to train the random forest
filtercustomer.py	Methods to filter customers who have purchased at least N products
itembasedcf.py	Item-based Collaborative Filtering implementation
probas.py	● Input: probabilities from the estimators (e.g. RF - train.py, RW - randomwalks.py, itembasedcf.py, external libMF implementation ● Output: a list of N (=6) products in descending order of likelihood ● Options: can apply weights to tweak relative weightage of methods. Also explored different ways of combination e.g. summation/geometric/harmonic/ranklists
randomwalks.py	Random walks implementation
score.py	Methods to calculate the MAP@6 score of predictions
splitxy.py	● X: a list of lists, with each internal list corresponding to one customer and showing the orders available within the first split. Used to mine data ● Y: a list of lists, with each internal list corresponding to one customer and listing the products that the customer bought within the second split of the data
train.py	Random forest and GBM (Gradient Boosting Machine) implementation ● training ● prediction ● feature prediction ● feature importance analysis