H&M Personalized Fashion Recommendations

Lan, Xing Yang

March 21, 2022

1 Introduction

1.1 Context

H&M is one of the largest companies in the clothing and fashion industry with a 21.57 billion dollar market cap. With ongoing current events in the World, online competitors such as SHEIN, the digitization and automation of the shopping process has made collecting data and improving the user experience a critical priority to remain competitive, and H&M's market cap has fallen 24% in the last month. H&M is hosting a Competition on Kaggle for people to predict future clothing purchases using years of real customer data, transaction data, clothing data, and a $50,000 prize pool.

The competition is titled "Personalized Fashion Recommendations" and the goal is to predict clothing that customers will buy in the next 7 days after the submission, where H&M will actually collect the testing dataset that does not exist yet. Participants can submit 12 articles of clothes for each customer that they believe the customer might buy in the 7 day period. Scoring is very expensive When there are thousands of submissions each containing millions of customers. As the scoring function does not penalize incorrect guesses, participants in the competition should submit 12 guesses for each customer.

1.2 Data Overview

There are 4 training data sets. The first dataset is images of all the clothes that are within the scope of the competition. The second dataset is on the customers, providing a customer ID for each customer. The third dataset is on the clothes or interchangeably "articles", providing an article ID and many characteristics of the clothing such as the color and department. The fourth dataset is a transaction history that provides transaction IDs each corresponding to an article ID and a customer ID indicating the customer purchased the article at the price and date also given in the transaction. The customer data set which doesn't have a lot of information in the first place is not very informative with the exception of the postal code column. However one can search through the transactions and match a customer up to the articles of clothing that they have bought and articles of clothing in the articles.of set have a lot of data.

1.3 Parametric and Nonparametric Components

The problem is fundamentally a non-parametric problem; the information given is nonlinear and quantitative. Fields such as natural language processing and image processing, seasonal time series are well within the scope of the model, so there are a few important ideas to cover.

1. The scoring does not have a known distribution like an MSE, and many of the prior probabilities are unexplored.

2. Model is going to be used for an actual company so an optimized ensemble will continue to add data and account for as much information as possible to maximize revenue. Rao-Blackwell theorem shows that for an unbiased estimator, the estimate can be improved given more information. The model for the posterior distribution can be improved with more prior information.

3. The clothing recommendations should be personalized, although you could make a vector with 12 columns, that doesn't account for drastically different variables such as different amounts of clothing between customers and more data about some customers and not others.

4. Strategies solely based on gaming the scoring function such as providing an estimate spread are not in the spirit of the competition find a company's personal interests that conflict with clothing suggestions such as keeping a customer on the website for as long as possible are not in the spirit of research.

5. The problem of whether a customer giving a piece of clothing is Boolean it's not directly a problem. A decision tree or progression answers. rather it's an interaction that requires a model first, to generate observations for another model to work on. there is very little data for each customer, and a ratio like the likelihood ratio test for comparison is not very meaningful in this nonparametric contest because the trooper a Subspace is a tiny subset of all possible clothes.

1.4 What makes a model good?

To get an idea of a good model, one can look at what other companies are doing for recommendation algorithms. When there's millions of dollars of funding the actual model used will try to account for as much real-world information as possible including natural language processing image processing real world fashion trends and drivers such as social media, and time series components. Doing so would make a very robust model optimizing this problem. Yet, at the same time this problem can also be solved very easily with just one hunting coding and K means on the cluster of clothes customer has bought. It is context specific, and models exist relevant only to other similar models, with criterion for parameters only having relative meaning. In 2007, Netflix awarded 1,000,000 USD to an team that made the best Netflix movie recommendation algorithm. The team used "an ensemble method of 107 different algorithmic approaches, blended into a single prediction." But Netflix's algorithm is open source, and SHEIN's algorithm is not.

1.5 Remarks

The goal of the competition is to make the best possible recommendation algorithm possible for H&M. Based off examples of recommendation systems historically accepted, any ensemble with a good chance of winning is well beyond the scope of the class. The idea however is clear. Construct a statistical framework to create a meaningful ensemble of transformations to the data. linear algebra component of this project with a transformation that does not require bias, from a perspective I am trying to build a model that can combine information from different data sets in a meaningful way; qualitative features into numerical company computations for a computer on distributions and assumptions that can cause problems down the line.

An instructor once said that there are a lot of machine learning models these days that claim To not use any assumptions at all, but if you look into the code go have a Gaussian distribution in there. Again and again reoccurring issues of unseen bias show up in statistical processes. "Probability calculations are hard. There may not be one right way to do something. It is easy to violate some hidden or unknown assumption" a Harvard Astrophysicist says in his 2020 critique on a 29 page statistical paper. A team wrote the paper over the course of a month but started off with incorrect application of the binomial distribution, making the same mistake seen in 1977 case of Castaneda v. Partida (DeGroot, 2011, p.278 & p.293).

2 Statistical Prelude

2.1 Prior and Posterior Probabilities

The probability space of a customer is not every article at H&M, but only the articles they have seen. This along with many factors occurring give the articles they have bought at some point in time a posterior distribution on many unknown priors.

The articles a customer has bought can not directly be split into training and testing, then bootstrapped for a model. The conditional probability of buying a piece of clothing is conditional on the clothing the customer already has, so the my model's customer representation is repeatedly calculated.

2.2 KMeans and OneHot Encoding

KMeans reduces in cluster variance it can be considered as a sample of a sample, such that for each cluster maximum likelihood estimation would have less variance on a global scale. see means therefore will be implemented in this model at the very beginning on customers to reduce variance. This is based on the intuition that men and women prefer different clothes and people over the age of 50 and people under the age of 25 also tend to prefer different clothes. The intuition is that reducing in cluster variance is best done at the beginning if it is to be done at all so that it doesn't create next effects later on.

2.3 Forward-Modeling

For each customer a different version of the customer calculated for the generation of their likelihood function or interchangeably, scoring function. When the scoring function is calculated for articles that are omitted in that version of the customer, the response variable is naturally True.

The intuition behind this is if one already has a pair of white leggings already they would probably want to buy a pair of different colored leggings as opposed to white leggings. While such a method is by no means completely free of bias, using joint likelihoods is a common introduction of bias on the assumption of some Gaussian distribution of clothing purchases. That would be similar to performing linear regression on the count vectored matrix of a corpus of text.

3 The Datasets

3.1 Articles

The "Articles" dataset provides all the details of the article, given the "article_id", including the type of product, the color, the product group and other characteristics of the item. The comprehensive list of the names and information of the variables will be provided at the Appendix section.

3.2 Customers

The customers dataset contains mostly low dimensional variables on the customer, except for age and postal code. The variables are,

1. customer_id : an ID number for referencing customer
2. FN : "missed" or 1
3. Active : "missed" or 1
4. club_member_status: if the customer is an H&M club member
5. fashion_news_frequency: How frequently the customer receives a fashion newsletter from H&M
6. age: the age of the customer
7. postal_code: the customer's postal code

3.3 Transactions

A transaction, or a single row of the "Transactions" dataset is a row of form,

"t_dat", "customer_id", "article_id", "price"

which give the time of the transaction, the customer ID, the article ID, and the price of the transaction on an normalized scale.

4 Proposed Method: Representation

4.1 Discuss an Existing Method: Alternating Least Squares

Firstly I want to cover a powerful existing method. Alternating Least Squares, (ALS) is an existing machine learning method used in recommender systems for movies and TV shows. The role of this specific model in a technique is to aggregate existing data of the user (customer in our case) with the data that it is considering recommending; it is one model amongst many in an ensemble. There are two main reasons I do not simply adapt existing code and slap it on to this project, as one may be able to infer from the motivations section, and the earlier discussion of prior and posterior probabilities.

First, it would no longer be a project but just busy work, and second, if it was really that simple then H&M would probably not be hosting this competition. People will watch the same movies again, and watch remastered versions of old movies, and will happily watch movies that are about as similar to each other as they can legally be within a platform. This applies to food and restaurants and so on. While people will watch the 3 original Spider man movies back to back and have an 8 Harry Potter Movie marathon, intuitively I think the most avid consumers of online clothing differ in consumption from consumers of movies and TikToks. While large consumption of movies does not necessitate diversity in the movies watched; nobody is trying to buy the same top 3 times in different colors. ALS generates an interaction score using matrix factorization. While it is a great model, it has scalability issues and makes ensembling more difficult later on. So here is an outline of the model I am proposing and prototyping.

4.2 Vector Representation

One-hot encoding a row of the Article dataset converts the row corresponding to that article to an array that is with the length corresponding to the number of total feature. The clothing then for each column or category you would get the different categorical values that are possible as the features.

4.3 Customer Representation

Intuitively people are likely to buy some thing similar to what they already have; people know that people follow fashion trends and one clothes that match their style. given that assumption One can naturally conclude that a similarities score can be calculated between a customer and an article of clothing, and it would be reasonable to conclude that given the information at hand, the clothing that the customer has bought, The similarities score is a meaningful function on the clothes they had bought g(X1,n) to create a new vector

Given a training set alarms and testing site of clothes for a customer that can artificially be created, a customers oh wow factor can be implemented as just the sum of the vectors of the clothes that they have bought. Multiplying their vector can use the similarities score between a customer who had only bought the clothes in their testing site at the time and give an assessment of the customers transaction history and the article. This is forward modeling because a similarity score is parameter of the posterior distribution conditional on a prior distribution.

4.4 Similarity Score

After articles have been one hot encoded, the vector for an article of clothing has columns corresponding to the features possible, for the features in each characteristic. Interestingly, the cross product of two encoded articles is number of features the articles have in common. For example, two articles that are both black and both T-shirts, would both be 1 the corresponding feature indexes. This metric is the original similarity score (SS) in my first implementation.

4.5 Similarity Vector

The similarity factor is just the multiplication of two factors for each index, a multi dimensional similarity score, and a similarity vector's absolute value as the similarity score. Therefor the Similarity Vector (SV) for a customer and for an article will have length $N_{features}$

4.6 Category Similarity Vector

This is a similarity Vector except the feature values are added back together for each category. Therefor the Category Similarity Vector (CATSV) for a customer and for an article will have length $N_{categories}$

4.7 Generalized Mathematical Representation and Notation

For all m articles that exist, a $m \times n$ matrix A can be formed by one hot encoding the matrix of artilces (after feature selection).

A: a matrix of all encoded articles

$A[i]$: a row or vector that represents a single article

Given the above representation, here are some properties.

1. Vector of features they have in common (cross product):

$$A[i] \times A[j]$$

2. Number of features in common (item-wise multiplication):

$$A[i] \cdot A[j]$$

3. If for customer C_1,

$$C_1 = \sum_{tr} A[i]$$

where

tr: a sample of the clothes they have bought for training

ts: the clothes they have bought not in tr

Then a row in data matrix $C_1.X$ is as follows,

$C_1.X[i] = C_1 \cdot A[i]$

and the corresponding row in their data matrix $C_1.Y$ is as follows,

$C_1.Y[i] = 1, \forall i \in ts$

$C_1.Y[i] = 0, \forall i \notin tr \cup ts$

5 Overview of the General Ensemble

The ensemble itself, or the entire model, is as follows for a single customer.

1. K-Means or k_NN on customers to reduce within cluster variance.
2. Obtain the data for their articles that can be further processes, such as the popularity of the article, image data, NLP data of the article descriptions, time series/trend data, real world data.
3. Bootstrap or K-fold the customer's cart and calculate the tr customer representation each time, and calculate rows of $C.X$ with articles they will be buying and random articles they never bought.
4. Optionally combine each row of $C.X$ together from a Similarity Vector back to a Category Similarity Vector.
5. Add for each row the corresponding data such as pricing, transaction date, difference in transaction date, and any data from 2. alongside the Category Similarity Vector.
6. Now that the $C.X$ and $C.Y$ contains bootstrapped relevant data for the customer, fit a regressor such as Ridge or neural network such as MLP Regressor on $C.X$ and $C.Y$.
7. Use the fitted regressor to predict scores for clothes the customer has not bought yet, and those will be the scores that the ensemble predicts the customer will buy.

6 Simulation

6.1 Process

The main difficulty of working with the data is that it is extremely large. With 37 million transactions I was unable to open the CSVs upload it. Particularly, methods such as indexing and related functions were extremely slow. I started the simulation study by using the last 50,000 transactions. I created index maps between columns with 1 to 1 correspondence across datasets for quick access, so that corresponding variables could be obtained with a single index.

After encoding the clothes, answering whether the similarity score metric has any meaning as a generated parameter is not difficult at all to test. The score itself provides information if it can discriminate between the case of $SS(Purchased, WillPurchase)$ and $SS(Purchased, NotPurchased)$. This was quickly confirmed early on with examples of randomly paired articles with high similarity scores, which will not be covered in this report.

7 Results

I think the best way to show the results is to show the clothes photos of what the customer bought, and then what the model predicts the customer will buy. Due to technical reasons I never downloaded the images from the Kaggle Dataset, so I searched them up. This version of the ensemble was done without the K-Means/k-NN beforehand. I had to drop most of the color, description, and item 2 data for the sake of time, but it does consider price.

7.1 Customer's Purchases

(top 8 photos in order) Customer 120 in the model randomly chosen,

7.2 Predictions

(top 8 photos in order) At (alpha = 0.5, rep = 6, falseRep = rep*1.8) wit ridge(alpha = 0.01),

8 Conclusion

I would say there are some major problems with color at the moment, but not color shading, which indicates it is likely an issue with a combination of the color category. Aside from that, the above implementation is pretty much I expect it to be.

It uses arbituary K-Means and ridge regression, and has not gone through parameter optimization. The implementation at the moment is proof of concept of the CATSV model. It has predictive ability towards what a customer might buy, and I might continued to optimize sometime in the future. The model is upscalable, pretty optimized with the index retrieval system of calculating, and can work with an ensemble of other models to give highly relevant clothing suggestions despite the lack of information it has at the moment.

9 References

The references

1. DeGroot, M., Schervish M. (2011). Probability and Statistics. 4th edition.
2. Photoexcitation. “A Critique of Dream Investigation Results”. Dec 21, 2020.

10 Appendix

10.1 Explanations

1. The "binomial" distribution was used in the 1977 case of Castaneda v. Partidais even though the problem is not that of a binomial distribution's, but rather a binomial distribution conditional on a Beta distribution on Dream's actual probability, given an observation (DeGroot, 2011, p.278 & p.293).

10.2 Article Dataset Variables

They are,

article_id - A unique identifier of every article
product_code & prod_name - A unique identifier of every product and its name
product_type & product_type_name - The group of product_code and its name
graphical_appearance_no & graphical_appearance_name :
colour_group_code & colour_group_name : information on color
graphical_appearance_no & graphical_appearance_name : number & name of graphical appearance
perceived_colour_value_id & perceived_colour_value_name : more information on color
perceived_colour_master_id & perceived_colour_master_name : more information on color
department_no & department_name: identifiers for departments
index_code & index_name: info on H&M's own index for the article
index_group_no & index_group_name: info on H&M's own index for the article
section_no & section_name: section information
garment_group_no & garment_group_name: information regarding the clothing
detail_desc: Details

10.3 CATSV model

```
def createCustomers(__customer_transactions):
    Customers = []
    for c_id in customer_transactions:
        Customers.append(Customer(c_id, customer_transactions[c_id]))
    return Customers

class Customer(object):
    def initialize(self):
        self.article_inds = [article_ids.index(transaction) for transaction in self.cart]
        for ind in self.article_inds:
            self.CF += A[ind]

    def __init__(self, customer_id, purchases):
        self.id = customer_id
        self.cart = purchases
        self.article_inds = []
        self.CF = zeros(n_features)
        self.X, self.Y, self.n = [], [], len(self.cart)
        self.initialize()

    def generateObs(self, alpha, rep):
        sample_size = int(self.n * alpha)
        for i in range(rep):
            selected = sample(self.article_inds, sample_size)
            remain = [ind for ind in self.article_inds if ind not in selected]
            CF = sum([A[ind] for ind in selected])
            for rem in remain:
                Xi = getXi(CF, rem)
                self.X.append(Xi)
                self.Y.append([1])
            for ind in [randint(0,n_articles-1) for i in range(rep*2)]:
                if ind not in self.article_inds:
                    Xi = getXi(CF, ind)
                    self.X.append(Xi)
                    self.Y.append([0])

class Model(object):
    def __init__(self):
        self.customers = createCustomers(customer_transactions)
        self.n = len(self.customers)

    def convertToVec(self, customer, ind):
        calculated = list(customer.CF*A[ind])
        calculated = convertToCat(calculated)
        price = article_prices[article_ids[ind]]
        Xi = [price] + calculated
        return Xi

    def fit(self, alpha, rep):
        for i in range(self.n):
            self.customers[i].generateObs(alpha, rep)
        print("Model Has been fitted with", len(model.customers))
```

Listing 1: Raw Code

```
import os
import re
import random
import statistics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from numpy import unique
from random import randint, sample
from numpy.random import random_sample
from numpy import sqrt, dot, array, diagonal, mean, transpose, eye, diag, zeros
from numpy.linalg import inv, qr
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression, Ridge, Lasso

transactions = pd.read_csv("transactions_train.csv").drop("sales_channel_id", axis = 1)

articles = pd.read_csv("articles.csv")

customers = pd.read_csv("customers.csv").dropna()

def encode(data):
    """Returns a fitted one-hot-encodeder"""
    encoder = OneHotEncoder()
    encoder.fit(data)
    return encoder

def view(vec):
    """
    Takes: a similarity vector
    Returns: a list of characteristics to print
    """
    global features
    chars = []
    for i in range(len(features)):
        if vec[i] == 1:
            chars.append(features[i])
    return ", ".join(chars)

def prune_customers(transactions, n_transactions, min_buy):
    """
    Takes: raw transactions, the amount of desired transactions, and the minimum buy amount
    returns: {customer_ids: list of purchase arrays}
    """
    transactions_new = transactions.iloc[-n_transactions:]
    # get customers corresponding to the transactions:
    unique_customer_ids = unique(transactions_new["customer_id"])
    # some transactions correspond to an NA customer, initializing:
    customers_no_NA = list(array(customers["customer_id"]))
    # initialize the dictionary for speed
    dd = {customer_id:[] for customer_id in unique_customer_ids}
    for row in transactions_new.iterrows():
            dd[row[1][1]].append(row[1][2])  # an array (tr_id, price, date)
    return {customer: dd[customer] for customer in dd if len(dd[customer]) >= min_buy and customer in cu

def prune_customersDF(_customers, _customer_transactions):
    """
    Used to shrink customersDF into only the ones selected
    Returns: a DF
    """
    # uses indices for iloc
    indices = []
    customer_ids = list(_customers["customer_id"])
    IDs = list(_customer_transactions.keys())
    for _id in IDs:
        indices.append(customer_ids.index(_id))
    return _customers.iloc[indices]

# get all the relevant articles
def prune_articlesDF(customer_transactions, raw_articles):
    """ (for the sake of runtime)
    Returns a DF of articles that have been purchased
    """
    raw_article_ids = list(raw_articles["article_id"])
    indices = []
    for cart in customer_transactions:
        for article in customer_transactions[cart]:
```

```
            indices.append(raw_article_ids.index(article))
    indices = list(sorted(list(unique(indices))))
    return raw_articles.iloc[indices]

n_keep = 100000
customer_transactions = prune_customers(transactions, n_keep, 12)
transactions = transactions.iloc[-n_keep-1:]

for i in range(2):
    tr_inds = list(transactions["article_id"])
    pr_inds = list(transactions["price"])
    article_ids = list(articles["article_id"])
    customers = prune_customersDF(customers, customer_transactions)
    articles = prune_articlesDF(customer_transactions, articles)

prices = []
for purchase in article_ids:
    tr_ind = tr_inds.index(purchase)
    prices.append(pr_inds[tr_ind])

drop1 = ["article_id", "product_code", "prod_name", "product_type_no", "graphical_appearance_no"]
drop2 = ["colour_group_code", "perceived_colour_value_id", "perceived_colour_master_id", "department_no"
drop3 = ["index_code", "index_group_no", "section_no"]
drop4 = ["garment_group_no", "detail_desc"]

articles_enc = articles.drop(columns=drop1)
articles_enc = articles_enc.drop(columns=drop2)
articles_enc = articles_enc.drop(columns=drop3)

articles_enc = articles_enc.drop(columns=drop4)

encoder_A = encode(articles_enc)
A = encoder_A.transform(array(articles_enc)).toarray()

features = []
article_prices = {}

for cat in encoder_A.categories_:
    for feature in cat:
        features.append(feature)
for i in range(len(article_ids)):
    article_prices[article_ids[i]] = prices[i]

n_features = len(features)
n_articles = len(article_ids)

print("len(articles_enc) == len(A) and len(articles) == len(A):\n", len(articles_enc) == len(A) and len(

i = 0
category_names = [col for col in articles_enc.columns]
category_lengths = []
for col in articles_enc.columns:
    i += len(list(unique(articles[col])))
    category_lengths.append(i)

def convertToCat(calculated):
    global category_lengths
    n = len(calculated)
    ret = []
    for i in range(len(category_lengths)):
        if i == 0:
            ret.append(sum(calculated[:category_lengths[i]]))
        else:
            ret.append(sum(calculated[category_lengths[i-1]:category_lengths[i]]))
    return ret

def getXi(CF, ind):
    calculated = list(CF*A[ind])
    calculated = convertToCat(calculated)
    price = article_prices[article_ids[ind]]
    Xi = [price] + calculated
```

```
    return Xi

def createCustomers(__customer_transactions):
    Customers = []
    for c_id in customer_transactions:
        Customers.append(Customer(c_id, customer_transactions[c_id]))
    return Customers

class Customer(object):
    # Initialize (): functions like a super().__init__ but instead of inheritance from parent class it ge
    def initialize(self):
        self.article_inds = [article_ids.index(transaction) for transaction in self.cart]
        for ind in self.article_inds:
            self.CF += A[ind]

    def __init__(self, customer_id, purchases):
        self.id = customer_id
        self.cart = purchases  # now: [ids]
        self.article_inds = []  # to index A
        self.n = len(self.cart)
        self.CF = zeros(n_features)
        self.X = [] # list of CF copies
        self.Y = []
        self.initialize()

    def generateObs(self, alpha, rep):  # True and False now
        sample_size = int(self.n * alpha)
        # rep is the amount of times to repeat this bootstrap process
        for i in range(rep):
            # selected indices (inds) of clothes to train on
            selected = sample(self.article_inds, sample_size)
            # clothes not selected, each will be used to calculate a Similarity Vector (a row of
customer's X)
            remain = [ind for ind in self.article_inds if ind not in selected]
            # make a matrix X
            CF = sum([A[ind] for ind in selected])
            # calculate train observations:
            for rem in remain:
                #### not anymore: pick a formula, include price, recombine the features into categories
                Xi = getXi(CF, rem)
                self.X.append(Xi)
                self.Y.append([1])
            # calculate test observations
            for ind in [randint(0,n_articles-1) for i in range(rep*2)]:
                if ind not in self.article_inds:
                    Xi = getXi(CF, ind)
                    self.X.append(Xi)
                    self.Y.append([0])


class Model(object):
    def __init__(self):
        # can use:
        # global customer_transactions
        self.customers = createCustomers(customer_transactions)
        self.n = len(self.customers)

    def convertToVec(self, customer, ind):
        """converts indice of article to formula vec """
        calculated = list(customer.CF*A[ind])
        calculated = convertToCat(calculated)
        price = article_prices[article_ids[ind]]
        Xi = [price] + calculated
        return Xi

    def fit(self, alpha, rep):
        for i in range(self.n):
            self.customers[i].generateObs(alpha, rep)
        print("Model Has been fitted with", len(model.customers))

from timeit import default_timer
```

```
start = default_timer()
model = None
model = Model()
model.fit(0.5, 6)
stop = default_timer()

print(f'Elapsed Time: {stop - start}s customers')

rCInd = randint(0, len(model.customers)-1)
print(f" Customer with indice {rCInd}s shopping cart: \n")
for ind in model.customers[rCInd].article_inds:
    print(f"article ID: 0{article_ids[ind]}")
    print(view(A[ind]), "\n")



def sortDD(pred):
    npred = {k: v for k, v in reversed(sorted(pred.items(), key=lambda item: item[1]))}
    return {key: npred[key] for key in list(npred.keys())[:12]}

def fullPredict(model, n_cust):
    "one one person"
    # choose model
    skM = Ridge(alpha=0.01)
    tty = [item[0] for item in model.customers[n_cust].Y]
    #print(array(model.customers[n_cust].X), array(model.customers[n_cust].Y))
    skM.fit(array(model.customers[n_cust].X), array(model.customers[n_cust].Y))
    Xtest = []
    for i in range(n_articles):
        Xtesti = model.convertToVec(model.customers[n_cust], i)  # person 0
        Xtest.append(Xtesti)
    # use model2
    favorites = {}
    pred = skM.predict(Xtest)
    for i in range(len(pred)):
        if pred[i] > 0.5 and article_ids[i] not in model.customers[n_cust].cart:
            favorites[int(article_ids[i])] = float(pred[i][0])
    return sortDD(favorites)

# Here are the results; based on the first customer in the model class, these are the predicted scores o
print("The Model's Top 12 Predictions for it's 0th customer:\n")
customer_pred = fullPredict(model, rCInd)
for item in customer_pred:
    print(f"article ID: 0{item}, Predicted Score: {customer_pred[item]}")
    print("Features", view(A[article_ids.index(item)]), "\n")
```