1 Coupon-Recommender-System for Starbuck's App Users built with Contextual Bandit Ready Data

1.1 Preface

The final capstone solution with the Starbucks data differs in terms of goals and methods, as they were described in the initial proposal. For example, while unsupervised learning was indeed used with and without AWS, the decision, on how to address Starbucks's goals, was finally made by choosing a reinforcement algorithm/method, instead of a classical supervised method. So you will find differences when reading this document, and comparing it to the initial capstone proposal, and you will see that this solution will not always gives a concrete answer to the ML Engineers rubric. Link to proposal: https://review.udacity.com/#!/reviews/3261307. However, the following structure of the text mostly reflects the structure of the ML Engineer rubric although it makes probably not always sense.

In addition, this paper uses the annotation "we" instead of "I". However, this does not mean that I worked with several people on the solution. It is however very uncommon to write so many sentences with I or in passive sentence. Thus, I will stick to "we" when describing, if a certain action was done or thought was put into shape.

1.2 Definition

1.2.a Project Overview

The following case is an example case of the company Starbucks, known for hot- and cold coffee and coffee-like beverages. As described in the proposal, Starbucks is a Fortune 500 company with more than 30.000 stores in more than 80 markets. The data was collected from the Starbucks App.

Given the fact that their products fulfill a basic need, like caffeine-shot and 'good' taste, their products may be interchangeable, and may only stand-out from the crowd by emotional brand aspects and special flavor beverages. Despite the more or less high interchangeability, their special products and basic products have **higher prices**, justified by achieving a community and union-feeling when consuming Starbuck's. However, higher prices induce that not everyone is able to consume their products everyday or several times a day.

Some users will wait or search for coupons, while others consume their beverages independent from that need. Thus, the consumption may, to some extent, **income** driven. Thus the income of every user may be one of the most interesting variables to look at.

Another interesting aspect may be the **coupons** themselves. Sending out coupons to users, does not mean that they will use them, although they may be registered in the Starbucks App. People still underlie the freedom to chose, if they want to redeem coupons and/or have the need to redeem coupons. The underlying dataset reflects exactly this kind of situation. The dummy data set provided by Starbucks consists of users buying behavior **observed over time** in the Starbucks App for **only one product** (showing us that the real problem is much more complex).

Also, users were eligible to receive 10 different coupons for the product, which are further sectioned into a **coupon status** like **received**, **viewed**, **redeemed** and finally acquired the product with the respective **transaction value**. So, the final data set shows historical usage/buying behavior of users, since they joined the app including their interaction with the coupons. The following input variables are given by starbucks, and are free to experiment with, I will use the same description as in the proposal and will add some insights:

profile.json (len max 17000, min 14825, user demographics)

- gender (object): gender, has 4 characteristics (male, female, other, none)
- age (int64): age of users
- id (object): system-id given by the app
- became_member_on (int64): value, when the user joined the app (format: yearmonthday)
- income (float64): yearly income of users

portfolio.json (len 10, coupon characteristics)

- id (string): id_code of the coupon
- reward (int64): reward for the user in terms of savings
- difficulty(int64): A scale of 0-10 showing (probably; I do not know) the amount of dollars to spend so the coupon can be redeemed
- duration(int64): how long it takes to take to redeem the coupon
- offer_type(string): name of the coupon, e.g. bogo (buy one get one free)
- channel (string): through which channels the user could possibly addressed with that coupon (e.g. email, social, web etc.)

transcript.json (len 306534, events per user tracked)

- person (string): system id of the user
- event (string): event taken by the user (transaction, offer received or viewed etc.)
- value (string): value earned for Starbucks through the user
- time (int64): time until the next action/event was taken (time has no units, it is unclear, if t is days or hours, however)

1.2.b Problem Statement

Since there was no real problem defined for the Starbucks data, the solution could be basically anything to tackle. However, one overall goal of Starbucks will always be: how to get the highest/best value out of every user. One way to address this issue is to look at:

1.) Starbucks's actions (A)

Starbucks's actions in the dataset are relatively clear. Since there is no price visible they changed. The only stimuli available is a set of different coupons (A), where each user may receive a coupon (a) out of this set, at a specific point of time.

2.) Users characteristics and behavior (C).

The other source of course reflects attributes of the users and their behavior. Thus, how many income they have, how old they are, and if the looked at a coupon or also redeemed it.

1.2.c Project goals

Project goals defined in the initial proposal changed a little bit, but in its core, the goals are more or less the same. But now, given the information of the Problem Statement, we can label every goal depending on their 'data origin':

1. Characteristics of individuals:

- Do groups of individuals actually exists, how can we separate them? **(C)**
- Are there specific characteristics per group or do we face more or less equal groups? **(C)**

2. Maximizing the revenue for individuals:

- If groups exist, is there are way to identify different values of groups for Starbucks? **(A)**
- Can we identify, which group of individuals should get a specific coupon to spend more? **(A)**
- Does a group redeem an incentive at all, and how can we probably encourage them **(A)**

Summing up: It seems a valuable task to somehow group users, identify the characteristics and their historical behavior and give Starbucks from this information some insights on, what they could have done better with their coupons.

1.2.d Metrics

We will use the following metrices overall in the project: - R^2, MAE for checking the quality of imputing missing continuous variables - Accuracy for checking the quality of imputing missing category/binary variables - α-significance-levels for permutation-tests - redeem-probability of a user for a next best action/coupon Most of the metrics, we mentioned, in the proposal we won't use, because the main task is no longer a pure ML-problem, it is now a Reinforcement Problem. What this means and how actions and characteristics fit in, we will explain later.

1.3 Analysis

1.3.a Data Exploration, Exploratory Visualization, and Data Preprocessing

Data file: 01_Starbucks_Data_Preparation.ipynb

Files used in Notebook: profile.json, portfolio.json, transcript.json

Files created in Notebook: starbucks_df_per_coupon.csv , starbucks_df_per_person.csv

1.) From the 17.000 individuals in **profile.json**, we have gender data on 14.825 individuals, and we can see it is skewed toward males

	gender	amount	percentage
0	F	6129	41.342327
1	M	8484	57.227656
2	O	212	1.430017

2.) Looking at the channels in **portfolio.json**, through which the individuals are addressed, these are dominated by email. Since we have 10 different coupons, that means every coupon will be sent by email, while only a little bit more than half of the coupons (6) may be acquired over social. Yet, all coupons are more or less available in all channels, which gives a channel variable not that much variation.

	channels	amount	percentage
0	email	10	30.303030
1	mobile	9	27.272727
2	social	6	18.181818
3	web	8	24.242424

3.) When looking at the reward, difficulty, and duration of every coupon in regard to all channels, we can see that informational coupons, have the lowest duration, yet zero difficulty and reward. It seems they are just what their name says, just informational, nothing to achieve nothing to redeem. Yet, bogo seems the most worthwile for all users compared to the difficulty. The reward equals always the difficulty. Discount seems to be less interesting, sometimes the difficulty is 2 to 5 times higher than the reward.Essential is, that every sort of coupon (informational, bogo, discount), is represented at least once through every channel. As told previously, we have 10 different coupons.

	id	reward	difficulty	duration	offer_type	email	social	web	mobile
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	discount	1	0	1	0
1	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	discount	1	1	1	1
2	2906b810c7d4411798c6938adc9daaa5	2	10	7	discount	1	0	1	1
3	3f207df678b143eea3cee63160fa8bed	0	0	4	informational	1	0	1	1
4	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	bogo	1	1	1	1
5	5a8bc65990b245e5a138643cd4eb9837	0	0	3	informational	1	1	0	1
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	bogo	1	0	1	1
7	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	bogo	1	1	0	1
8	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	bogo	1	1	1	1
9	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	discount	1	1	1	1

4.) Looking at all the events, done by users in the **transcript.json**, we can see that less than half of the offers, that were sent to users were actually completed. Telling us, that there are users, who are not really interested at the coupons, but transaction happens anyway. Yet, you can see that there is a structure on events beginning with received, viewed, completed, and transactions (which may also happen anyway). It seems that these actions follow some sort of funnel. If we stick to this sort of definition, we can check, if users fulfilled a whole customer journey. We name that variable **amount_of_completed_user_paths.** And this variable may help us later. There are other variables we have finally engineered, but they are of less importance for the documentation. Although the algorithm will use them. However, the income is indeed a variable we have to investigate further, as previously mentioned.

	event	amount	percentage
0	offer completed	33579	10.954413
1	offer received	76277	24.883700
2	offer viewed	57725	18.831516
3	transaction	138953	45.330371

Data file: 02_Starbucks_Data_Imputation_and_Statistical Testing_per_person.ipynb

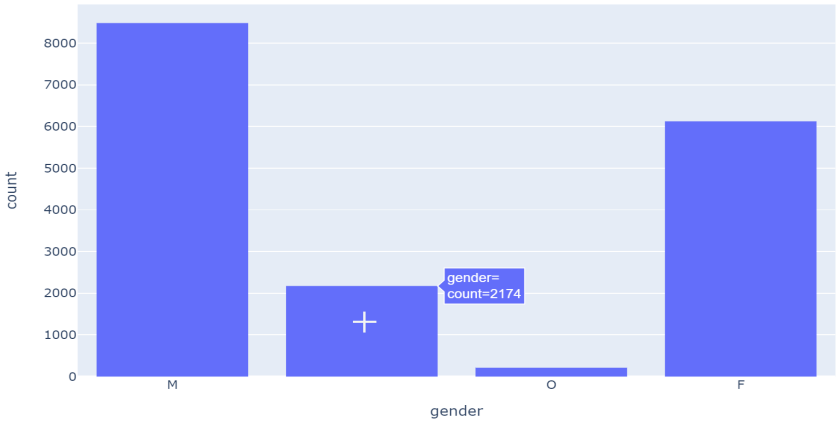
Files used in Notebook: starbucks_df_per_person.csv **Files created in Notebook:** starbucks_df_per_id.csv, starbucks_imputed.csv

5.) We now look at the next notebook and continue exploration on a different scale. We see that we have NaN data, that only effects 6 rows. We simply delete these rows , and continue with 16994 rows of data from individuals.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 31 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              17000 non-null  object
 1   time                            17000 non-null  int32
 2   value                           17000 non-null  float64
 3   gender                          17000 non-null  object
 4   age                             17000 non-null  int32
 5   became_member_on                17000 non-null  int32
 6   income                          14825 non-null  float64
 7   reward                          17000 non-null  float64
 8   offer_received                  17000 non-null  int32
 9   offer_viewed                    17000 non-null  int32
 10  offer_completed                 17000 non-null  int32
 11  transaction                     17000 non-null  int32
 12  offer_views_missed              17000 non-null  int32
 13  informational                   17000 non-null  int32
 14  bogo                            17000 non-null  int32
 15  discount                        17000 non-null  int32
 16  total_reward_completed          17000 non-null  float64
 17  total_duration_completed        17000 non-null  float64
 18  total_difficulty_completed      17000 non-null  float64
 19  difficulty_offered              16994 non-null  float64
 20  duration_offered                16994 non-null  float64
 21  last_time_offer_received        17000 non-null  int32
 22  last_time_offer_completed       17000 non-null  int32
 23  first_time_offer_received       17000 non-null  int32
 24  first_time_offer_completed      17000 non-null  bool
 25  mean_time_between_actions       17000 non-null  float64
 26  amount_of_completed_user_paths  17000 non-null  int32
 27  email                           17000 non-null  float64
 28  social                          17000 non-null  float64
 29  web                             17000 non-null  float64
 30  mobile                          17000 non-null  float64
dtypes: bool(1), float64(13), int32(15), object(2)
memory usage: 2.9+ MB
```

6.) Now we look at the previously mentioned information on income data for our individuals. Looking at income we can see, that nearly every gender is in every income range. However, in our data set, there are more females having much more income (right side of histogram), although we have much more males in the data set. Looking at the diagram, we also see a high descent in income around 75 k, maybe this highlights the mean or the median of the income. Even if not, it seems that there is a break in income range, between a high amount of users having low, and a low amount of users having high income. This is a very intersting thing and we will look, if this difference is significant (practical relevance is already given by this high numbers or so to speak the magnitude: Practical vs. Statistical Significance - Statistics By Jim)

Using an approximate permutation test (Permutation Test: Visual Explanation (jwilber.me)) we can see that income below the mean or the median and above are significantly different from each other. Given the scale of the income, this also implies, that effect sizes are present (6.4 - Practical Significance | STAT 200 (psu.edu), and that this variable may have a certain relevance (**see also Notebook: 02_Starbucks_Data_Imputation_and_Statistical Testing_per_person.ipynb, cell number: 19 and 20**). We will also see in a later stage (but it is a good thing to already mention it), that the rows, which have missing information on gender, also have missing information on income, and have messed up age data (e.g. age above 90 years, e.g. 118 or so) and also missing gender data. So it seems, that we have to do some sort of imputation, if we do not want to drop more than 2000 rows of individual personel data. As a reminder look at the data for gender:



7.) We now face at the decision, how we want to fill the missing/wrong data in our over 2000 rows. We now that gender, age and income are missing in the same rows. One solution, that could be helpful, would be a simple machine-learning approach to forecast missing data. We use the ExtraTreesRegressor/ExtraTreesClassifier with a RandomizedSearchCV in a sklearn pipeline to find a good combination of hyperparameters for our missing gender, age and income. (For gender, we make a binary encoded variable with the labelbinarizer, which is a way of forecasting a mutliclass target without doing a one vs all binary forecast.

Hindering Data Leakage

What is essential is, that it depends on the case with which variable we want to forecast, and that the should not forecast missing variables with already forecasted values. Thus, if we forecast age, income or gender, we use only a fraction of the train data and forecast every variable for itself, without giving the testdata information about the already forecasted values. Otherwise this would induce some sort of data leakage from the first forecast,

8.) Looking at our results we get the following R^2 values for our missing or broken rows:

- income: r^2 train: 0.988, r^2 test: 0.690 (higher degree of overfitting)

- gender: accuracy train: 0.823, accuracy test: 0.632, (some degree of overfitting)

- age: r^2 train: 0.236, r^2 test: 0.139, (which seems worse, but still better than the mean age) The results of imputing show the following:

- Income seems broadly distributed. The overall count column inreases to 400 and looking at the bars, everywhere is a raise to be recognized:

- Imputing gender leads to overweight male much more, since we do not know the ground truth for gender behind Starbucks user data (overweight toward male or female or equally distributed). We do not know if we made it worse or not. Could be that we added 2101 males with low income and 60 females with high income. As long as we do not know, if this is ground truth, we leave it like that **(We do not know the full ground truth because we have only 1 product here!!!)**

```
count     2174
unique       2
top          M
freq      2101
Name: gender, dtype: object
```

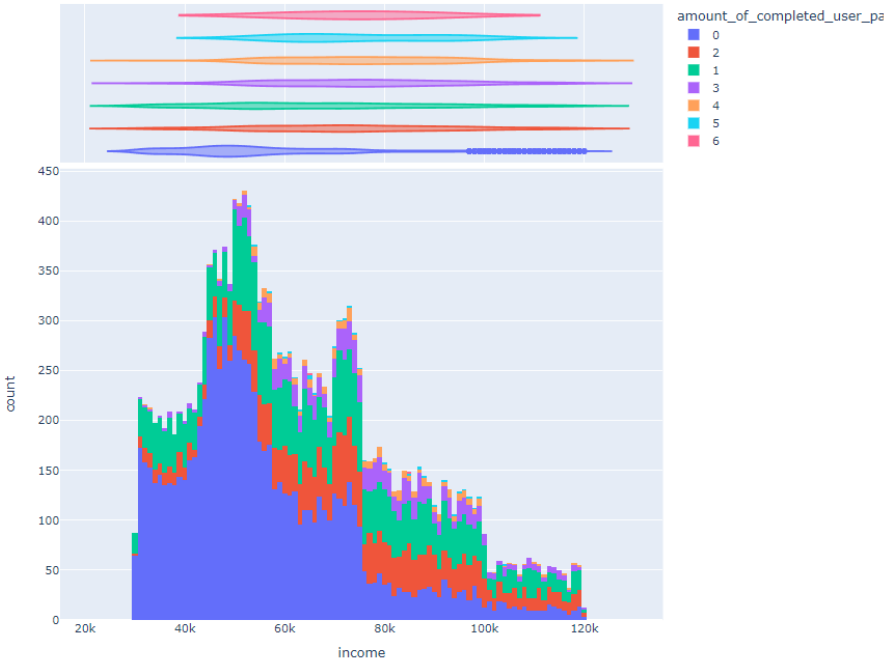
- For age we have no specific insight, but we decided to set an upper bound, where we believe the age of a current subject is not above 90. So everything above 90 seems less likely, we let the algorithm forecast these values again.

Data file: 03_Manual_User_Clustering_and_Goal_Derivation.ipynb

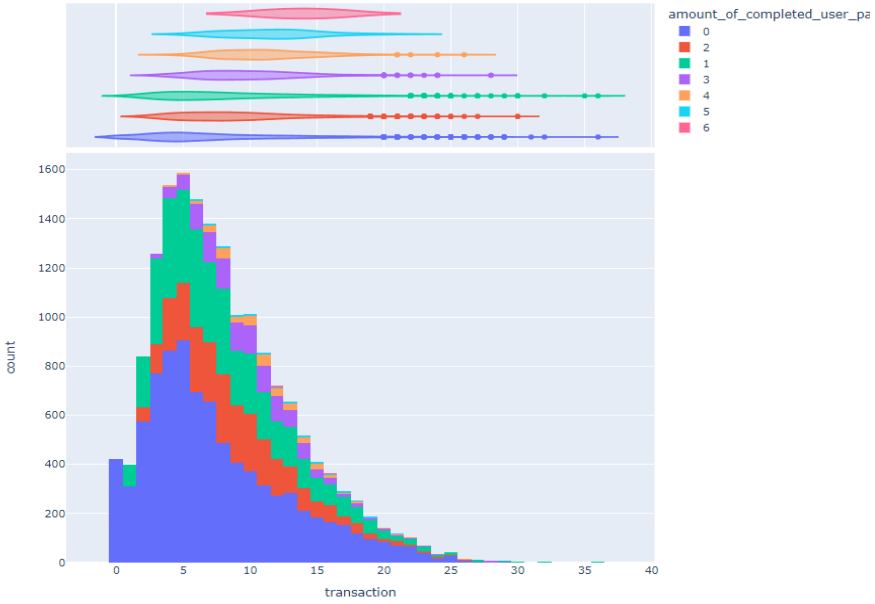
Files used in Notebook: starbucks_imputed.csv

9.) We now have a more or less cleaned, imputed and prepared data set for a deeper look. Lets look at our goals we defined initially, that we wanted to have separate groups, if possible. Since Starbucks focuses on user data, identfying users behaviorial structures in data mess is an obvious goal. One approach, we highlighted earlier, is the focus on a customer journey or an user funnel. Surprisingly, we already engineered a variable, which reflects the amount of complete paths per user. If we split user data among this variable, and look at the income, we see the first interesting insights:

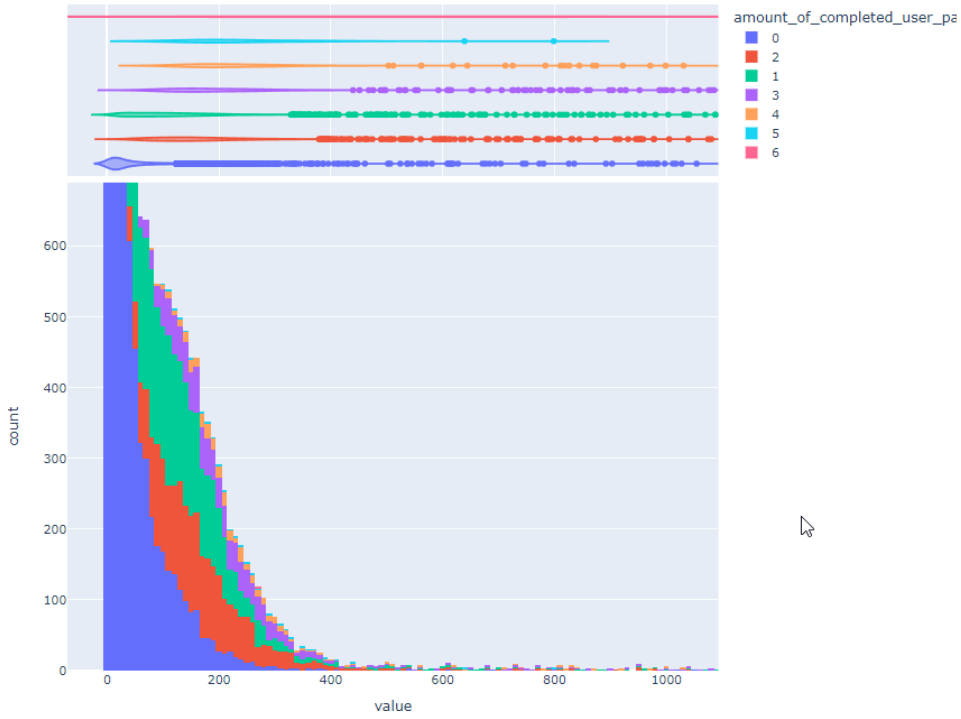
- Users, who have no complete user path, have the lowest income, as can be seen from the belly of the blue distribution. Users, who have a higher income, tend to complete more user journeys, when looking at the right side of the histogram.



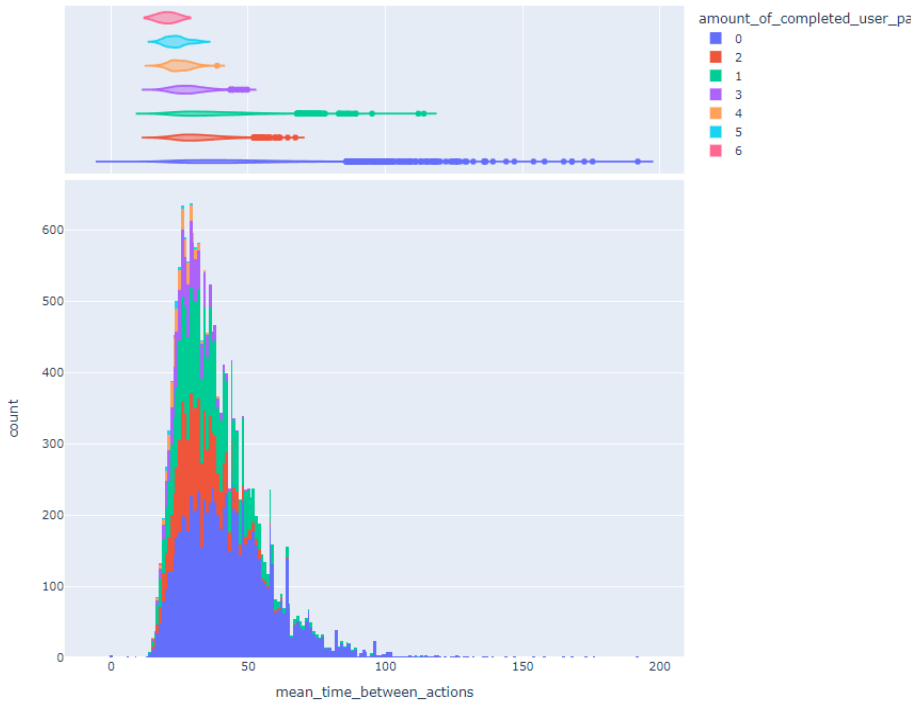
- This does also account for the amount of transactions



- As well as for the value of users. We count more higher value for users which have at least more than 1 (> 1) fulfilled customer journey (picture shows a zoomed view, look at notebook for more and further details)



- Interestingly users, with more completed user paths, are faster, when it comes to make decisions (in whatever unit of time the data is ;-)



1.3.b Algorithms and Techniques Used in Detail

1.3.b.1 Supervised Methods for Imputing Missing Values

10.) We have already outlined a few things we have done in the Data Preparation / Imputing /Cleaning Part. Since the Rubric demands it, we highlight at this part the usage of our supervised methods for this: We used the ExtraTreesClassifier and the ExtraTressRegressor to impute missing variables (Extra Trees because we want to see, if a hardcore random algorithm is better than a pure random fill in (accuracy == 0.5) or a mean fill ($R^2 = 0$): sklearn.ensemble.ExtraTreesClassifier — scikit-learn 1.0.2 documentation

sklearn.ensemble.ExtraTreesRegressor — scikit-learn 1.0.2 documentation

1.3.b.2 Statistical Tests to look for a group effect in one main driver variable

11.) We also made a approximative permutation test to see, if there is a difference between high and low income, as we thought this is a main driver of user characteristics, as higher incomes result in more completed user journeys, which require more money to spend (Fulfilled user journeys for just informational user journeys are more or less not useful for starbucks, if we do not account for something like brand effects and only pure generated value):

http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/ (Example 1)

1.3.b.3 Unsupervised Methods to show skills learned for AWS and to show alternatives for the upcoming Reinforcement Task

12.) What we not have discussed until now, is the use of an unsupervised cluster algorithm. But what for? We have already developed a variable that shows groups, like **amount of completed user paths**. We could already make all kind of groups with this variable. What do we get out of it? Well, we use this as a showcase, for showing off AWS skills as this is still an ML Engineer Course. Also, we would like to highlight alternative versions for the upcoming reinforcement task, which needs clustered data to train own. Thus, we decided to show two different cluster approaches in two different ways:

12. a.) Clusters used from AWS: Kmeans from sagemaker: http://sagemaker.readthedocs.io/en/stable/algorithms/kmeans.html

12. b.) Clusters used from Sklearn: AffinityPropagation https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

While kmeans may be already known to many users. AffinityClustering tries to find clusters, without giving any hints, how many clusters there are. Yet, the method needs parameters to be tuned, which reflect the amount of clusters found. *Keep in mind:* There is no real clustering without giving some information upfront to look for, although some unsupervised methods state it. Nevertheless, we still use the AffinityPropagation, since it was the only interesting algorithm, which uses a fit and predict method beside kmeans.

Although DBSCAN and HBSCAN would have been much more interesting as alternative clustering methods instead of AffinyPropagation, we were not able to let it run it, as it would have needed a self written predict function that only returns labels from the fit method of DBSCAN/HDBSCAN. But, maybe in a revised version on my github.

1.3.b.4 Reinforcement Task: Contextual Bandits for recommending a user within a specific group the next best Starbucks coupon

13.) The final main task, we have not yet talked about, will be a reinforcement task. That means, that we choose an algorithm, which learns from past data of users, and based on user's past decisions tries to recommend users coupons with increased probability f**or another fulfilled user path**. For this, approach we look for two or three groups of users. Which groups of users this will be, we explain in the Methodology section. As mentioned in the metrics section, our goal is to get a redeem probability for every coupon, a user could get, telling Starbuck's, which coupon they should send next to a specific user in a specific group.

For this we will refer to the library of vowpal wabbit in Python. A reinforcement library in python which is specifically designed for this problem. The workthrough of the page also explains what the difference is between multi-armed bandits and contextual bandits. The main goal therefore is, that we are using historical data of Starbucks Sended Coupons (Actions), and Users Characteristics (Contextual information) to show us a final array of redeem probabilities for every coupon in Starbucks repertoire. Contextual Bandits — VowpalWabbit latest documentation

For more info on multi-armed bandits vs contextual bandits see here:

https://stanford.edu/~ashlearn/RLForFinanceBook/MultiArmedBandits.pdf

1.3.c Benchmark Possibilities

1.3.c.1 Benchmark for supervised-based filling of missing values In *supervised ML tasks*, a Benchmark would be a baseline model, reflecting the mean of a target kpi, which is basically an R^2 of zero. Or in terms of a classification task, an accurarcy of 50 % thus a straight line dividing a roc coordinate system into two identical triangles. We have this benchmark for imputing our missing values, and already discussed it and used the techniques.

1.3.c.2 No Benchmark for our unsupervised and our reinforcement task

In *unsupervised ML tasks*, we do not know the ground truth of "how many" clusters there really are. Our data is unlabeled in such a way, that the solution could be basically anything, there is no reality to compare with. A big cluster, 4 to 5 minor clusters or two mediocre clusters, everything might be possible. Yet, we could give found solutions a 'quality stamp' e.g. by the elbow method Elbow Method — Yellowbrick v1.4 documentation (scikit-yb.org) or by the cvi: A new clustering validity index for arbitrary shape of clusters – ScienceDirect. Yet, a minimized sum of squared errors or a quality index also do not guarantee that this is the real world we are reflecting with our solution. There is nothing to compare to. Our clusters regardless which approach we use, base on theory and gut feeling.

In general, for *reinforcement tasks*, benchmarks means something different: rss2018 (stanford.edu). Yet here the rubric refers to a baseline model. While *it would be* possible to use, the most bad algorithm and set a time benchmark for solving our next best action coupon problem, we want to stick in line with the ML cosmos we have here.

Unfortunately, there is no real ground truth to compare to also! But not, because there is generally none in terms of reinforcement learning/recommender systems! Although, we always know, which coupons a user took in the past and how much it costs Starbucks and how much value the user generated), we will never know, if our reinforcement task recommended better coupons to users, than the recommendation engine used in the past. Why is that so?

Because the baseline model is something we still have to built in the future! It yet does not exist., Given a control group, just like in A/B Splittesting, we would know, if a recommendation engine would be better. And if our reco engine, trained on users with more fulfilled user paths, really seduces users, to redeem more coupons, than for example, just a random reco of coupons. And since A/B Splittesting is something that borrows from bayesian statistics, it is no wonder that a bandit approach with such a control group for a reco system, also borrows from that theory. A contextual bandit algorithm updates its reward distribution every time an action from Starbucks (sending out a coupon) ended in a fulfilled user path. If not, probabilities over time for specific coupons will change. Awesome isn't it? If you want more on why there is no benchmark group go to chapter 1.5.

1.4 Methodology

The following rubric's section will deviate from the ML Engineer Rubric. For our final solution, it is not needed to do an AWS clustering, as we already knew we want to use the "amount_of_fulfilled_user_paths" to look at. Nevertheless, we will highlight the aws results for a final show off of our cloud sagemaker skills, we achieved during the course. In addition, Implementation and Refinement section switched places.

1.4.a Data Preprocessing

1.4.a.1.a Processing for (builtin) Unsupervised clustering in AWS and short results overview

Data file: **04_AWS_Clustering_Unsupervised.ipynb**

Files used in Notebook: **starbucks_imputed.csv**

Files created in Notebook: **starbucks_imputed_scaled.csv**

In this file we are using the imputed data from the previous section (starbucks_imputed.csv) to look for clusters and similarities between users automatically. We want to see, if **a builtin cluster algorithm of aws tends** toward our general idea by clustering users in terms of fulfilled user paths. But before doing a clustering in AWS, we scale the data, as our distances between our features will be distorted by features with very large numbers. This would result in features which seem to be essential for clustering although they are not. After achieving a scaled data set, we use the data to feed it to the kmeans algorithm of aws. The results indicate, that three cluster solution (also tried a two cluster solution in the notebook for reasons) seem to make sense in terms of the user-funnel previously highlighted. We get clusters, which in general make sense. We have a cluster with nearly no fulfilled user paths (a few outliers remaining, cluster 1). A cluster with a lot of fulfilled user paths (cluster 2), and a cluster 3 where it seems, people are on the edge of 'wandering' to cluster number two.

Cluster 1

count	5625.000000
mean	0.087822
std	0.284940
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	2.000000

Cluster 2

count	4883.000000
mean	1.937948
std	1.132026
min	0.000000
25%	1.000000
50%	2.000000
75%	3.000000
max	6.000000

Cluster 3

count	6486.000000
mean	0.733734
std	0.771421
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	3.000000

1.4.a.1.b Processing for (external) Unsupervised clustering in AWS and short results overview

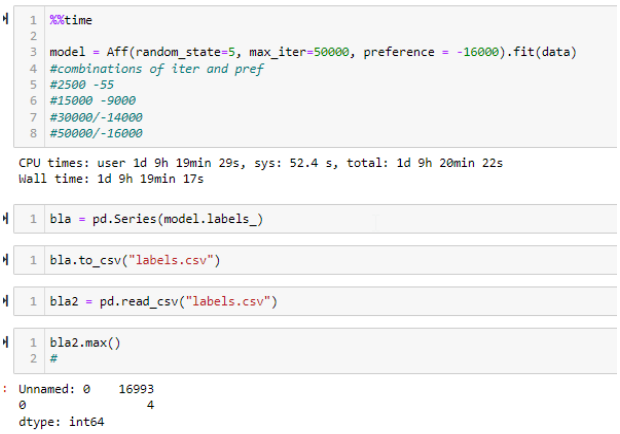
Data file: 05_AWS_External_Clustering_Sklearn_Affinity.ipynb

Files used in Notebook:

s3://sagemaker-us-east-1-729296914025/data/starbucks_imputed_scaled.csv

In this file, we do basically the same as before, but with an external algorithm by sklearn: We were limited by algorithms how have a fit and predict method. Thus, better solutions like HDBSCAN or DBSCAn, which have only a fit method, seem to be out of scope. But why?, To be honest, I didn't manage to tell AWS to return fit labels in its internale predict function, maybe I should have tried to overwrite it? Well maybe this is an interesting story for a revised version of this task, after earning the degreee.

Further, we achieved no solution with 3 clusters. Training with Affinity Algorithms creates more than 500 clusters in 3 minutes. Yet, a solution having at least 5 clusters (beginning with cluster number 0) takes more than a day. Thus we book this notebook here under "earned skills in AWS to use external methods and a training script".



1.4.a.2 Processing for Reinforcement Learning with Contextual Bandits

In this Notebook we map only our previous imputed values to our historical user data, which we need for our final task.

Data file: 06_Map_imputed_data_to_coupon_data.ipynb

Files used in Notebook: starbucks_df_per_coupon.csv, starbucks_df_per_id.csv, starbucks_imputed.csv

Files created in Notebook: starbucks_df_per_coupon_imputed.csv

1.4.b Implementation (final implementation of algorithms/tasks, no intermediate results)

Data file: 07a_Final_Model_Contextual_Bandits_with_Vowpal_Wabbit_cover_algorithm.ipynb 07b_Final_Model_Contextual_Bandits_with_Vowpal_Wabbit_bagging_algorithm.ipynb

Files used in Notebook: starbucks_df_per_coupon_imputed.csv a) Implementation example for supervised ExtreTreesRegressor (For Classifier it is the same, except we calculate the score on accuracy and predict labels instead of label probabilities):

```
# we fit a model on the full data
X_train, X_test, y_train, y_test = train_test_split(starbucks_income_gender_full_x, starbucks_income_full_y, random_stat

steps = [('extra', ExtraTreesRegressor())]
pipeline = Pipeline(steps)

params = {
    "extra__max_depth": [4, 8, 16, 24, 32],
    "extra__min_samples_split": [2, 4, 6, 8, 10, 12, 16, 24, 32],
    "extra__min_samples_leaf": [2, 4, 6, 8, 10, 12, 24, 32]
}

random_search = RandomizedSearchCV(pipeline, params, cv = 5, n_iter=50, random_state=0, verbose=5, scoring="neg_mean_abs
random_search.fit(X_train, y_train)
```

```
#ExtraTreesRegressor Imputing income with best params solution
extraReg = ExtraTreesRegressor(
    min_samples_split=random_search.best_params_["extra__min_samples_split"],
    min_samples_leaf=random_search.best_params_["extra__min_samples_leaf"],
    max_depth=random_search.best_params_["extra__max_depth"])

extraReg.fit(X_train, y_train)

# train_r2_score partial data
train_r2 = r2_score(np.array(y_train).reshape(-1, 1), extraReg.predict(X_train))
print(train_r2)

#test r2_score partial_data
test_r2 = r2_score(y_test, extraReg.predict(X_test))
print(test_r2)
```

```
0.9878056663999246
0.6889990783958915
```

```
#Have a look at our train r2 with full data (retrain on full data after tuning)
extraReg.fit(starbucks_income_gender_full_x, starbucks_income_full_y)
train_r2 = r2_score(starbucks_income_full_y, extraReg.predict(starbucks_income_gender_full_x))
train_r2
```

```
: 0.987572170568045
```

b) Implementation for unsupervised kmeans:

```
from sagemaker import KMeans
kmeans_customers = KMeans(role=role,
                          instance_count=1,
                          instance_type='ml.c4.xlarge',
                          output_path=output_path_cluster, # specified, above
                          k=2,
                          epochs=20,
                          sagemaker_session=sagemaker_session)
```

c) Implementation for unsupervised AffinityPropagation (which also uses the train.py):

```
# your import and estimator code, here
Affinity = SKLearn(
    entry_point="train.py",
    source_dir="train_scripts",
    framework_version='0.23-1',
    role = role,
    train_instance_count=1,
    train_instance_type="ml.c5.2xlarge",
    sagemaker_session=sagemaker_session,
    hyperparameters = {'max_iter': 20000, 'random_state': 5, 'preference': -1500}
)
#other than 0
```

train.py for Affinity:

```
# Here we set up an argument parser to easily access the parameters
parser.add_argument('--max_iter', type=int, default=400)
parser.add_argument('--convergence_iter', type=int, default=15)
parser.add_argument('--random_state', type=int, default=2)
parser.add_argument('--damping', type=float, default=0.5)
parser.add_argument('--preference', type=float, default=-1.0)
#0 is behavior < 0.23 scikit previously random_state 0 was hard coded

# SageMaker parameters, like the directories for output data and saving models; set automatically
parser.add_argument('--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
parser.add_argument('--data-dir', type=str, default=os.environ['SM_CHANNEL_TRAIN'])

# args holds all passed-in arguments
args = parser.parse_args()

# Read in csv training file and fit data
data_dir = args.data_dir
data = pd.read_csv(os.path.join(data_dir, "starbucks_imputed_scaled.csv"), header=None, names=None, skiprows=1)
 # delete unwanted shit from AWS
col = "Unnamed: 0"
if col in data.columns:
    data.drop(columns=[col], inplace=True)

model = AffinityPropagation()
model.fit(data)
#sagemaker_session.upload_data(path=f"{data_dir}/{pd.to_pickle(model, 'model_file.pkl')}")

# Save the trained model
joblib.dump(model, os.path.join(args.model_dir, "model.joblib"))
```

d) Implementation for Reinforcement Learning

Implementation of a reinforcement task needs historical data, where **actions** by Starbucks are the sent coupons, the **costs** for Starbucks are the savings by customers (which the algorithm tries to minimize), and **probabilities** that a user redeems a coupon, where calculated upfront, by looking at past data and the amount of fulfilled user paths in relation to the overall broadcasted coupons. **Context** information are the remaining features. For the concrete wordings look at the documentation, mentioned earlier:

```
vw = vowpalwabbit.Workspace("--cb_explore 10 --cover 3", quiet=True)
```

```
choices = pd.DataFrame()
for j in range(len(test_data)):
    #logically we do want to insert all the stupid columns at once
    test_example = "| " + " ".join([str(row) for row in test_data.iloc[j, :].values])
    choice = vw.predict(test_example)
    choices = choices.append(pd.DataFrame(choice).T)
```

```
for i in range(len(train_data)):
    action = train_data["action"].iloc[i]
    cost = train_data["cost"].iloc[i]
    probability = train_data["probability"].iloc[i]
    # Construct the example in the required vw format.
    #print(learn_example)
    learn_example = \
        str(action) + ":" + \
        str(cost) + ":" + \
        str(probability) + " | " + \
        ' '.join([str(row) for row in train_data.iloc[i, 3:].values])
    vw.learn(learn_example)
```

The Tuning for the Reinforcement Task only consists of choosing an appropriate algorithm: https://github.com/VowpalWabbit/vowpal_wabbit/wiki/Contextual-Bandit-algorithms The most sophisticated algorithm is a cover algorithm, which trains several policies (other name for models in the context of bandits), and tries to get a set of prediction probabilities, which is a very suited algorithm for our 10 coupons vector.

1.4.c Refinement for Choosing Contextual Bandit Algorithms

Although there could be made a refinement, on which clustering solution we would use for our contextual bandit problem, we only focus on refinement for the reinforcement task. Using a bagging algorithm instead of a cover algorithm, does not change the recommended coupons! That is very essential. **Both algorithms recommend coupons nr. 2, 4, and 9 to all users to increase probability of a fulfilled user path!** Additionally, the bagging algorithm 'rules out bad decisions' resulting in a fixed probability set of good actions for users, while other coupon probabilities are zeroed out.

However, that leads to the fact that the bagging algorithm does not reflects changing probas over time in context as it seems, and just gives the same proba for every coupon. That means that there is no variation over time. For example, in a certain point of time it could be more likely that coupon nr. 9 while be redeemed with a much higher probability than coupon number 2. But for the bagging algorithm, at every point of time, if a coupon is selected by the algorithm, its redeeming probability will never change.

The cover algorithm on the other hand, does have changing probas per point of time. In our notebooks:

- **07a_Final_Model_Contextual_Bandits_with_Vowpal_Wabbit_cover_algorithm.ipynb**
- **07b_Final_Model_Contextual_Bandits_with_Vowpal_Wabbit_bagging_algorithm.ipynb**

we see the whole differences, yet it is sufficient to look at the following visual results. We can see for the cover-algorithm that coupon nr. 4 increases in redeeming probability over time, for one user (see in chapter 1.4.c.1 the proba table)

1.4.c.1 Cover-Algorithm Implementation and Refinement

(where choices 0-9 reflect the coupons, and the rows are the historical user_data of all users) (the unique values show the different probabilities existent in the data set)

```
vw = vowpalwabbit.Workspace("--cb_explore 10 --cover 3", quiet=True)
```

```
1 unique, counts = np.unique(choices, return_counts=True)
2 unique
```

```
array([0.00340305, 0.10846428, 0.10846429, 0.21692859, 0.27116072,
       0.27116075, 0.37962502, 0.43385717, 0.5423215 , 0.59655362,
       0.59655368, 0.75925004])
```

```
1 choices
```

	0	1	2	3	4	5	6	7	8	9
0	0.003403	0.003403	0.216929	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.542322
0	0.003403	0.003403	0.216929	0.003403	0.542322	0.003403	0.003403	0.003403	0.003403	0.216929
0	0.003403	0.003403	0.216929	0.003403	0.542322	0.003403	0.003403	0.003403	0.003403	0.216929
0	0.003403	0.003403	0.108464	0.003403	0.596554	0.003403	0.003403	0.003403	0.003403	0.271161
0	0.003403	0.003403	0.542322	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.216929
...	...	...	...	...	...	...	...	...	...	...
0	0.003403	0.003403	0.216929	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.542322
0	0.003403	0.003403	0.216929	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.542322
0	0.003403	0.003403	0.759250	0.003403	0.108464	0.003403	0.003403	0.003403	0.003403	0.108464
0	0.003403	0.003403	0.542322	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.216929
0	0.003403	0.003403	0.542322	0.003403	0.216929	0.003403	0.003403	0.003403	0.003403	0.216929

16523 rows × 10 columns

1.4.c.1 Bagging-Algorithm Implementation and Refinement

```python
vw = vowpalwabbit.Workspace("--cb_explore 10 --bag 5", quiet=True)
```

```python
unique, counts = np.unique(choices, return_counts=True)
unique
```

```
array([0.        , 0.33333334])
```

```python
choices
```

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
...	...	...	...	...	...	...	...	...	...	...
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333
0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0	0.0	0.0	0.333333

16523 rows × 10 columns

1.5 Results (no comparison made to a benchmark)

1.5.a Justification

To finally justify, if our contextual bandit approach solved the problem, we are unfortunately missing certain parts to our puzzle, we can not really provide, but only describe. However, we have at least the justification of our doing by looking at our equal cluster characteristics for train and test, which will be explained at the end of this list.

1.5.a.1 Not justifiable

- **missing benchmark group**: like in classical A/B Splittests, or to be precise **we miss a treatment group to test this implementation of a coupon recommendation**. So to speak, we only have one group, the baseline group that created costs and transaction values based on the coupon reco decisions Starbucks already took. That brings us to our next problem:
- **alternative transaction value:** we only know the real transaction value, because the users in our baseline-group already took some decisions in the past, all we developed was an alternative recommender system of coupons, from which we have no insights, if it would bring us more transaction value for Starbucks, so logically, if you have no treatment group, you have no alternative transaction value
- **sum of costs:** The costs were the rewards for the user, and we only know the costs of our baseline group (68182). Alternative cost scenarios for our algorithm allow only for dumb cost scenarios, as it would consists of taking the min (33046) or the max costs (165320) induced by one of the three coupons recommended.
- **Random guess comparison:** Also, we would need a comparison against a pure random reco of coupons. It is possible though that our reco system for coupons is, in terms of costs and transaction value, worse than a pure random reco. That would be already an advanced benchmark

1.5.a.2 Justifiable

- **Equality of groups:** Normally, in A/B Splittest we have equal groups /group characterstics to be sure any change in user behavior is due to the stimuli, just like in a controlled experiment. In this context the clusters of train and test have to be equal at least in user characteristics, to see if we had send out our coupons to equal groups, being sure that the higher transaction value in the future would be addressable through the coupons (and they should have probably assigned randomly which is not true, but a random distribution should lead to more or less equal groups). The only variables we have for this are, gender, age and income. Looking at our data, we can see that at least from the descriptives. Normally, this would need deeper tests, but we justify our solution at least by cluster equality.

Cluster Train(left) and Cluster Test (right) for features gender and age

	gender	age	gender	age
count	15464.000000	15464.000000	16523.000000	16523.000000
mean	0.389033	53.186498	0.388247	53.237124
std	0.510865	15.723041	0.514433	15.528882
min	0.000000	18.000000	0.000000	18.000000
25%	0.000000	45.000000	0.000000	45.000000
50%	0.000000	53.000000	0.000000	53.000000
75%	1.000000	64.000000	1.000000	64.000000
max	2.000000	90.000000	2.000000	90.000000

Cluster Train(left) and Cluster Test (right) for feature income

	income	income
count	15464.000000	16523.000000
mean	63132.465019	63572.085477
std	20978.670982	20749.020536
min	30000.000000	30000.000000
25%	47264.305556	48000.000000
50%	59000.000000	60000.000000
75%	76000.000000	76000.000000
max	120000.000000	120000.000000

Thus, there is no Justification/Benchmark of our recommender system of coupons, regardless which cluster solution we would have taken (kmeans, affinity or the amount of fulfilled user paths). Because in the final contextual bandit task, there is no treatment group available. ***Yet, the system works and could be only justified in real time, for example, if it would be better than a random recommendation of coupons.***

1.5.b Conclusion and Model Evaluation (no Validation)

In this final project we showed, how starbucks could **group** (manually, kmeans or affinity) its users and tries to develop a coupon recommender system based on a contextual bandit-algorithm, where the **actions** are the coupons by starbucks, the **costs** are the rewards for users by redeeming the coupon, and the user characteristics and behavior where the **context** data.

For the final recommender system there is not much to say. The recommender system chooses two discounts and one bogo coupons for its users ,and no informational coupons so far. Although the informational coupons have the lowest cost, they do not generate value for starbucks (at least in our data table). The discount and bogo coupons are the folllowing;

	id	reward	difficulty	duration	offer_type
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	discount
1	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	discount
2	2906b810c7d4411798c6938adc9daaa5	2	10	7	discount
3	3f207df678b143eea3cee63160fa8bed	0	0	4	informational
4	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	bogo
5	5a8bc65990b245e5a138643cd4eb9837	0	0	3	informational
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	bogo
7	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	bogo
8	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	bogo
9	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	discount

We see that the algorithm chose two discounts with the lowest costs (user reward), which it would recommend to users in the test cluster (remember: users, in the train cluster hat several fulfilled user paths, users in the test cluster only one fulfilled user path). Yet, it chose one bogo coupon which had the highest cost and difficulty, but users redeem it very fast.

So it seems that our algorithm tries to find a balance between high and low cost. Interestingly it took only coupons which had high, but not the highest difficulty, yet the two dicount coupons had the highest difficult/cost ratio ($10/2 = 5$). Future revision of this approach should test the recommender system under real-life conditions to check if transaction values are really higher than the benchmark group and higher than a simple random recommendation of coupons.