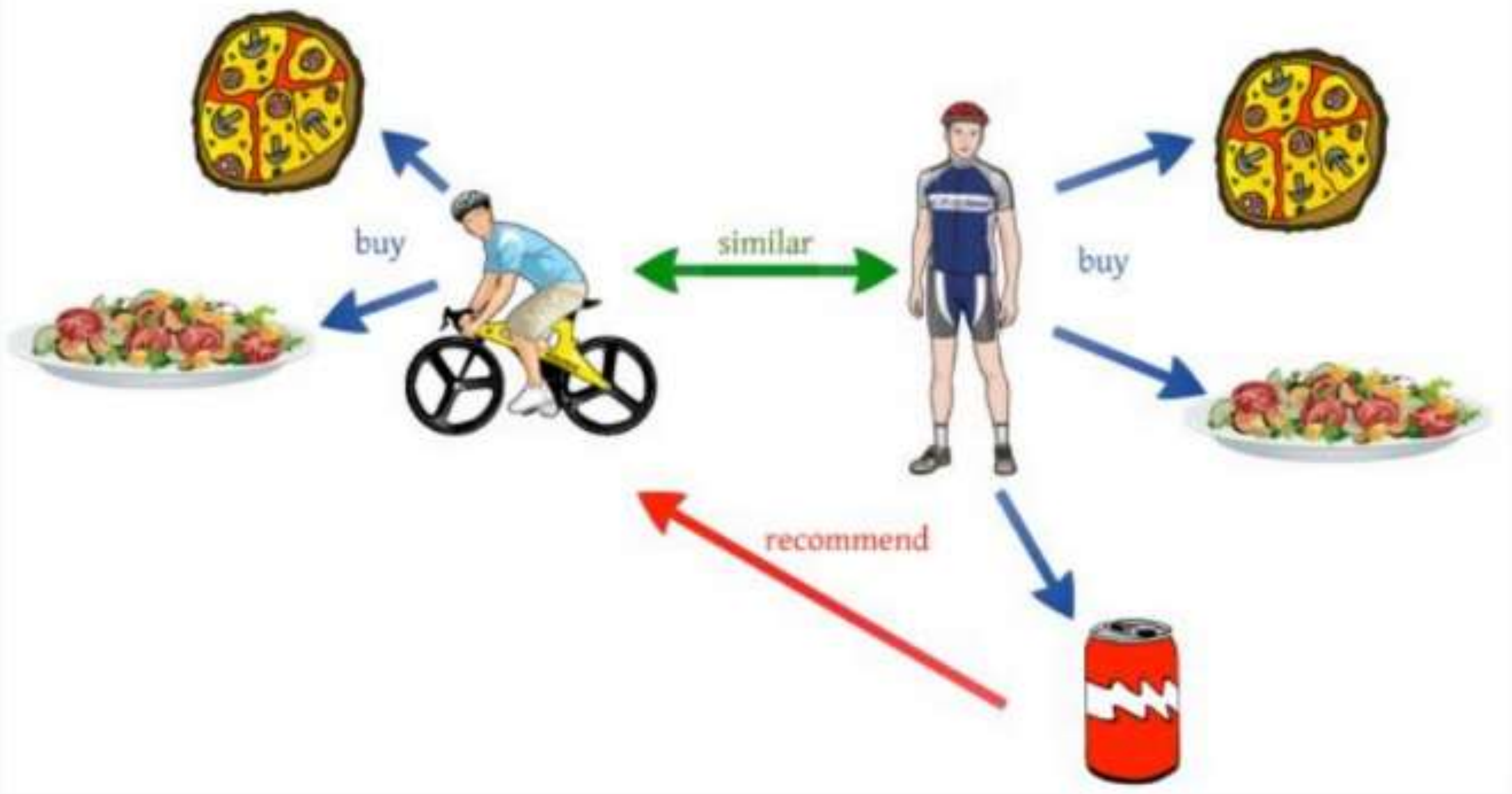
Build a Recommendation System for Peapod

- An application of item-based “Collaborative Filtering” model with Turicreate and Python
- Recommendation technology is used to achieve **GOAL** *to personalize grocery products and offers to customers using purchase data.*
- Below steps will be used to accomplish this goal:
  - Transforming and normalizing data
  - Training Models
  - Evaluating Model Performance
  - Selecting the optimal model

User Based Collaborative Filtering



Product Overview

Suppose a grocery chain release a new mobile app to allow its customers to place orders before they even walk into store.

There is an **OPPORTUNITY** for the app *to show recommendations*: When a customer first taps on the "ORDER" page, company may recommend top 10 items to be added to their basket. e.g. Fresh vegetables, Salad, Chips, Meat, Cookies, Bread, Honey and so on.

Problem Statement

In this data challenge, we are building collaborative filtering models for recommending product items. The steps below aim to recommend users their top 10 items to place into their basket. The final output will be a csv file in the output folder, and a function that searches for a recommendation list based on a speficied user:

Input: user - customer ID

Output: ranked list of items (product IDs), that the user is most likely to want to put in empty "basket"

Implementation

1. Import modules

- **pandas** and **numpy** for data manipulation
- **turicreate** for performing model selection and evaluation
- **sklearn** for splitting the data into train and test set

2. Load data

Two datasets are used in this exercise, which can be found in *content* folder:

- *recommend_1.csv* consisting of a list of 1000 customer IDs to recommend as output
- *tran_data.csv* consisting of user transactions

3. Data preparation

- Our goal here is to break down each list of items in the **products** column into rows and count the number of products bought by a user

3.1. Create data with user, item, and target field

- This table will be an input for our modeling later
- In this case, our user is **customerId**, **productId**, and **purchase_count**

3.2. Create dummy

- Dummy for marking whether a customer bought that item or not.
- If one buys an item, then ***purchase_dummy*** are marked as 1
- Why create a dummy instead of normalizing it, you may ask?
    - Normalizing the *purchase count*, say by each user, would not work because customers may have different buying frequency don't have the same taste
    - However, we can normalize items by ***purchase frequency*** across all users, which is done in section 3.3. below.

3.3. Normalize item values across users

- To do this, we normalize **purchase frequency** of each item across users by first creating a **user-item matrix** as follows

4. Split train and test set

- Splitting the data into training and testing sets is an important part of evaluating predictive modeling, in this case a collaborative filtering model. Typically, we use a larger portion of the data for training and a smaller portion for testing.
- We use 80:20 ratio for our train-test set size.
- Our training portion will be used to develop a predictive model, while the other to evaluate the model's performance.
- Now that we have three datasets with **purchase counts**, **purchase dummy**, and **scaled purchase counts**, we would like to split each.

5. Baseline Model

Before running a more complicated approach such as collaborative filtering, we would like to use a baseline model to compare and evaluate models. Since baseline typically uses a very simple approach, techniques used beyond this approach should be chosen if they show relatively better accuracy and complexity.

5.1. Using a Popularity model as a baseline

- The popularity model takes the most popular items for recommendation. These items are products with the highest number of sells across customers.
- We use *turicreate* library for running and evaluating both baseline and collaborative filtering models below
- Training data is used for model selection

Using purchase counts **Using purchase dummy** **Using normalized purchase count**

Notes

- Once we created the model, we predicted the recommendation items using scores by popularity. As you can tell for each model results above, the rows show the first 30 records from 1000 users with 10 recommendations. These 30 records include 3 users and their recommended items, along with score and descending ranks.
- In the result, although different models have different recommendation list, each user is recommended the same list of 10 items. This is because popularity is calculated by taking the most popular items across all users.
- In a grouping example below, products 132, 248, 34, and 37 are the most popular (best-selling) across customers. Using their purchase counts divided by the number of customers, we see that these products are at least bought 3 times on average in the training set of transactions (same as the first popularity measure on purchase_count variable)

6. Collaborative Filtering Model

- In collaborative filtering, we would recommend items based on how similar users purchase items. For instance, if customer 1 and customer 2 bought similar items, e.g. 1 bought X, Y, Z and 2 bought X, Y, we would recommend an item Z to customer 2.
- To define similarity across users, we use the following steps:

  1. Create a **user-item matrix**, where index values represent unique customer IDs and column values represent unique product IDs
  2. Create an **item-to-item similarity matrix**. The idea is to calculate how similar a product is to another product. There are a number of ways of calculating this. In steps 6.1 and 6.2, we use cosine and pearson similarity measure, respectively.
     - To calculate similarity between products X and Y, look at all customers who have rated both these items. For example, both X and Y have been rated by customers 1 and 2.
     - We then create two item-vectors, v1 for item X and v2 for item Y, in the user-space of (1, 2) and then find the cosine or pearson angle/distance between these vectors. A zero angle or overlapping vectors with cosine value of 1 means total similarity (or per user, across all items, there is same rating) and an angle of 90 degree would mean cosine of 0 or no similarity.
  3. For each customer, we then **predict his likelihood to buy a product** (or his purchase counts) for products that he had not bought.
     - For our example, we will calculate rating for user 2 in the case of item Z (target item). To calculate this we weigh the just-calculated similarity-measure between the target item and other items that customer has already bought. The weighing factor is the purchase counts given by the user to items already bought by him.
     - We then scale this weighted sum with the sum of similarity-measures so that the calculated rating remains within a predefined limits. Thus, the predicted rating for item Z for user 2 would be calculated using similarity measures.
- We can use **turicreate** library to capture different measures like using **cosine** and **pearson** distance, and evaluate the best model.

6.1. Cosine similarity

- Similarity is the cosine of the angle between the 2 vectors of the item vectors of A and B
- It is defined by the following formula

$$similarity = cos(\theta) = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- Closer the vectors, smaller will be the angle and larger the cosine

6.2. Pearson similarity

- Similarity is the pearson coefficient between the two vectors.
- It is defined by the following formula

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Notes

- In collaborative filtering above, we used two approaches: cosine and pearson distance. We also got to apply them to three training datasets with normal counts, dummy, or normalized counts of items purchase.
- We can see that the recommendations are different for each user. This suggests that personalization does exist.
- But how good is this model compared to the baseline, and to each other? We need some means of evaluating a recommendation engine. Lets focus on that in the next section.

7. Model Evaluation

For evaluating recommendation engines, we can use the concept of precision-recall.

- **RMSE (Root Mean Squared Errors)**
  - Measures the error of predicted values
  - Lesser the RMSE value, better the recommendations
- **Recall**
  - What percentage of products that a user buys are actually recommended?
  - If a customer buys 5 products and the recommendation decided to show 3 of them, then the recall is 0.6
- **Precision**
  - Out of all the recommended items, how many the user actually liked?
  - If 5 products were recommended to the customer out of which he buys 4 of them, then precision is 0.8
- **Why are both recall and precision important?**
  - Consider a case where we recommend all products, so our customers will surely cover the items that they liked and bought. In this case, we have 100% recall! Does this mean our model is good?
  - We have to consider precision. If we recommend 300 items but user likes and buys only 3 of them, then precision is 0.1%! This very low precision indicates that the model is not great, despite their excellent recall.
  - So our aim has to be optimizing both recall and precision (to be close to 1 as possible).

Lets compare all the models we have built based on precision-recall characteristics:

8.1. Evaluation summary

Based on RMSE

	Popularity Model	Cosine Similarity	Pearson Similarity
Purchase counts	Overall RMSE: 1.08923813046	Overall RMSE: 1.91864796826	Overall RMSE: 1.08667784593
Purchase dummy	Overall RMSE: 0.0	Overall RMSE: 0.96946812563	Overall RMSE: 1.0
Scaled counts	Overall RMSE: 0.135299735079	Overall RMSE: 0.163527228819	Overall RMSE: 0.134997902908

Popularity Model

Purchase counts

cutoff	mean_precision	mean_recall
1	0.00123376152116	0.000830974671602
2	0.000725742071268	0.000901820921416
3	0.00246752304231	0.00408089060386
4	0.00292111183685	0.00603594544395
5	0.00592205530155	0.0164145278207
6	0.00620509470934	0.0204656687595
7	0.00567115589976	0.021693035026
8	0.00531606067204	0.0230749688867
9	0.0050560030965	0.0247878929707
10	0.00513825386458	0.0285992798323

Purchase dummy

cutoff	mean_precision	mean_recall
1	0.0058568329718	0.00276248697073
2	0.0056399132321	0.00505454242873
3	0.0056158110388	0.00824877167502
4	0.00553145336226	0.0109233054338
5	0.00550976138829	0.0135297635389
6	0.00551940226561	0.016320707531
7	0.00545398202665	0.0185065182255
8	0.00573933477946	0.022191836964
9	0.0058809351651	0.026297473434
10	0.00608821402748	0.0307860687144

Scaled counts

cutoff	mean_precision	mean_recall
1	0.00231800072438	0.00138114209827
2	0.00242665700833	0.00285541815422
3	0.00241458408789	0.00449974991808
4	0.00202825063383	0.0049343750539
5	0.00188337558855	0.00558290044217
6	0.00205239647471	0.00729432314961
7	0.0021213845915	0.00899056847136
8	0.00218218036943	0.0102255864213
9	0.00235824379251	0.0125032290974
10	0.00236146323796	0.0139557738965

Cosine Similarity

cutoff	mean_precision	mean_recall
1	0.12148922273	0.0694807681634
2	0.0956165178895	0.108182417676
3	0.0805573699107	0.13518831688
4	0.0705058422237	0.156716976309
5	0.0635750054431	0.17557553816
6	0.0580109828967	0.191151386924
7	0.0533316744943	0.204719560898
8	0.0495500399158	0.216954753788
9	0.0465926409754	0.228342715884
10	0.043798534001	0.238004762798

cutoff	mean_precision	mean_recall
1	0.123499638467	0.0706187619924
2	0.0973246565437	0.108528700411
3	0.0821884791516	0.135716600484
4	0.0718365871294	0.156882097516
5	0.0642082429501	0.174289229229
6	0.0580380814654	0.18846258633
7	0.0528044623489	0.199022952172
8	0.0489696312364	0.210135377426
9	0.0457941672692	0.220233807758
10	0.0432393347795	0.230170848603

cutoff	mean_precision	mean_recall
1	0.0634552698298	0.0358238575307
2	0.04835204636	0.0530246419602
3	0.0421103464928	0.0687608357266
4	0.0367439333575	0.0783888188439
5	0.0329011227816	0.087049886467
6	0.0304720511892	0.0966614658812
7	0.0285714285714	0.105192212651
8	0.0271278522275	0.113957393809
9	0.0255302024226	0.12007390324
10	0.0242883013401	0.126077589786

Pearson Similarity

cutoff	mean_precision	mean_recall
1	0.00123376152116	0.000830974671602
2	0.000725742071268	0.000901820921416
3	0.00241914023756	0.00399017284495
4	0.00299368604398	0.00617141729725
5	0.00592205530155	0.0164145278207
6	0.0062413818129	0.0206833913809
7	0.00569189138752	0.0217898006355
8	0.00534327599971	0.0231862493377
9	0.00512051350283	0.0251048003418
10	0.00516002612671	0.0286379860761

cutoff	mean_precision	mean_recall
1	0.0058568329718	0.00276248697073
2	0.0056399132321	0.00505454242873
3	0.0056158110388	0.00824877167502
4	0.00553145336226	0.0109233054338
5	0.00550976138829	0.0135297635389
6	0.00551940226561	0.016320707531
7	0.00545398202665	0.0185065182255
8	0.00573933477946	0.022191836964
9	0.0058809351651	0.026297473434
10	0.00608821402748	0.0307860687144

cutoff	mean_precision	mean_recall
1	0.00231800072438	0.00138114209827
2	0.00242665700833	0.00285541815422
3	0.00241458408789	0.00449974991808
4	0.00204636001449	0.00497059381522
5	0.00188337558855	0.00558290044217
6	0.00206446939515	0.00731846899049
7	0.00218347389662	0.00919822270291
8	0.00233611010503	0.0112039816428
9	0.002326049338	0.0123764634328
10	0.00239043824701	0.0141006489418

Based on Precision and Recall

Notes

- **Popularity vs. Collaborative Filtering:** We can see that the collaborative filtering algorithms work better than popularity model for purchase counts. Indeed, popularity model doesn't give any personalizations as it only gives the same list of recommended items to every user.
- **Precision and recall:** Looking at the summary above, we see that the precision and recall for Purchase Dummy > Normalized Purchase Counts.
- **RMSE:** Since RMSE is higher using pearson distance than cosine, we would choose model the smaller mean squared errors, which in this case would be cosine. Therefore, we select the Cosine similarity on Purchase Dummy approach as our final model.

9. Final Output

- In this step, we would like to manipulate format for recommendation output to one we can export to csv, and also a function that will return recommendation list given a customer ID.
- We need to first rerun the model using the whole dataset, as we came to a final model using train data and evaluated with test set.

9.1. CSV output file

```
df_rec = recom.to_dataframe()
print(df_rec.shape)
df_rec.head()
```

```
(10000, 4)
```

Summary

In this exercise, we were able to traverse a step-by-step process for making recommendations to customers. We used Collaborative Filtering approaches with cosine and pearson measure and compare the models with our baseline popularity model. We also prepared three sets of data that include regular buying count, buying dummy, as well as normalized purchase frequency as our target variable. Using RMSE, precision and recall, we evaluated our models and observed the impact of personalization. Finally, we selected the Cosine approach in dummy purchase data.