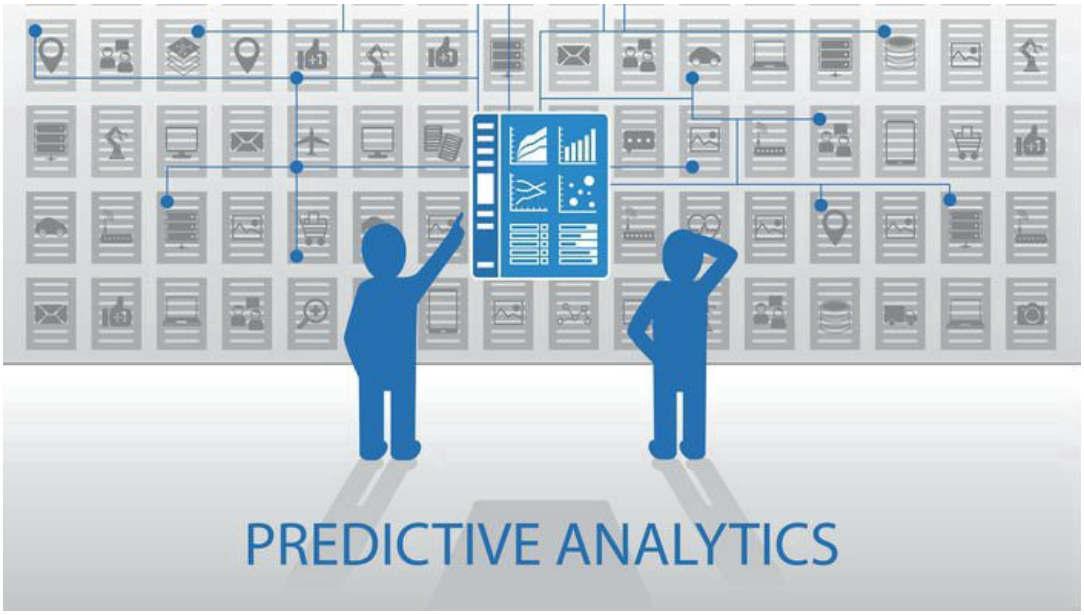
Towards
Data Science

Use Algorithms to Recommend Items to Customers in Python

Ryan Aminollahi
Mar 4, 2018 · 11 min read

Up-selling and cross-selling



Back to my previous use case . X just bought a laptop online. But let's think. Is a laptop the only item on a shopping list? He might need charging stations, monitors, and warranties. He might also be shopping for his middle school kid and need other laptops and electronics. Knowledge of the customer's desires and situations and up-selling cross-selling additional items is easy money for any business.

If we can predict X's shopping list, then we can sell him a lot more. More sales means more profit. You might have heard the sales lingo of up-sells and cross-sells.

> *Up-sell means selling additional items that complement the items purchased.*

So if X purchases a laptop, an up-sell would be cables, a bag, or a warranty.

> *Cross-sell means selling items in other independent categories the customer might be interested in.*

A person buying a high-end laptop is most probably a technical person, and he might be interested in technology-related books or other items like mobile phones.

Now, you might be wondering, what is the upsell cross-sell process?

How do we identify what else the customer is interested in? Well, there are a few

How do we identify what else the customer is interested in? Well, there are a few factors we might want to consider.

First, take a look at the purchased item. Items related to the items purchased are easy up-sells, like cables, bags, and warranty for a laptop. We might also think about the customer's demographics. Typically, customers in the same demographics desire the same kind of things.

Customers like Mr.X might be interested in laptops, desktops, and monitors.

Customers like Mrs.Y who is college-going might be interested in college supplies and fancy cell phone covers. Try to also consider gender. X might be interested in gaming software and accessories. Y might be interested in fashion wear. Sure, these I submit are stereotypical, but it aligns with market research. The next, life events like birthdays, graduation, college, childbirth, etc. trigger significant purchasing behavior.

> *If a business can know or predict such upcoming events, it can proactively offer relevant products and services to their existing customers.*

If X has been buying maternity products for the past couple of months, it would tell us that he has a baby on his way. It is time to offer baby products and packages. Next, seasons and yearly events also trigger significant purchases. Valentines Day, Thanksgiving, and Christmas are hot sales days for a reason.

Similarly, back to school timeframes also trigger new purchases. If Y's semester is starting next June, she's going to need new supplies. It's the right time to offer her back-to-college bundles.

Price and brand preferences are also important considerations. Some customers always look for the lowest priced product or deal. Other go for brand names. Knowing this pattern would help you offer the right product within the category. The up-sell and cross-sell process requires that you understand customer buying behavior and offer the right products and services that appeal to him or her.

We need to collect data relevant to these factors to build models and then use the right models to predict customer needs. This way, we can help our business make easy money.

Find items bought together

Let's put our knowledge of up-selling and cross selling into play. Imagine the following scenario.

Mr. X walks into his favorite departmental store and is shocked to find that the shelves have been rearranged. The products are not where they usually are. X asked, why did they do that? The answer is something we're going to see in this use case. The departmental store has analyzed shopper buying patterns and arranged the products such that the products bought together are stocked and displayed near each other.

This is the reason why you see milk and bread together, or beer and chips together, even though one requires refrigeration and the other one doesn't. The goal of this use case is to find items frequently bought together. We are focused here on those that are

bought in a single transaction, not items bought by the same customer across time. The customer is immaterial here. By finding such items it is possible for us to stock them together, create product bundles, or offer them as this item is usually bought together with on our website.

The data for this use case is going to be sale transactions. Each sale is a record with a list of items bought in the transaction. The algorithm we use here is the association rules mining algorithm, or more properly called, marked basket analysis. Association rules mining are put rules that specify the dependency of one item onto another. It would be like if you bought a laptop, then carry cases are bought together 60% of the time.

> *We will use sale transaction data to run this algorithm and generate rules that shows the percentage of times that things are done together. As you would notice, this action is not customer-specific so it does not require real-time action based on the customer. Using this use case we decide on which items to stock together, which products to group together in a catalog, or which products to show on the site for any main product in our website.*

In our online computer store scenario, how do you think this is going to translate? Would you group similar items into the same categories, or would you provide them as sidebar recommendations? You can do both, based on how your marketing folks think. The key is to identify related items and this use case helps you do so.

Create customer group preferences

Let take a look at the scenario in which we can group similar customers into advertising buckets. Imagine that our old friend X has just signed up as a new customer with our business and has bought a laptop. Great, but what other general product categories will he be interested in? Can we sell him books? Will he be interested in music CD's? There are a couple of ways to attack the issue. I would like to suggest the possibility of identifying other customers who are like X.

We can then use data from these customers to recommend products to X. Think about it. X is a new customer. He really has no history with our business and we don't know much about him. How about, we can try to fit him into a sort of pre-created customer groups based on certain personal attributes. Then, based on common buying preferences for that group, we can recommend different products to X.

> *There are two sets of data required for this use case. First is the demographics of the customer. Then, We need the sale transactions involving these customers and various products that we sell.*

So what does the algorithm look like? First, based on customer attributes, we try to group similar customers. We would use clustering algorithms like ***K-Means*** or ***K-***

Nearest-Neighbors to create these groups. Try to create an optimal number of groups, say three to ten.

Next, with these customer attributes as future variables, and the groups as target variables, build a classification model that can predict the group based on these attributes.

> *A general tree classification would suffice. Based on past transaction history, build a list of products that a customer in each customer group would be interested in. This can be a simple count based algorithm, or a more sophisticated one like association rules mining.*

Essentially, We come up with an affinity score between the customer group and the product. So what is the call to action here?

Build models that group customers, and then build rules that associate these groups of customers to products. Know when a new customer comes in, we can use the customer classification model to predict increal time the group to which the customer belongs. Once we have the group, we can then recommend the products the group most bought through a similar customer's bought recommendations.

This is a very broad algorithm in the sense that each group of customers might be interested in a hundred or more items if the catalog is very large.

We can alternatively compute the affinity between the customer groups and product categories, and recommend the customers to visit a specific category rather than recommend a single product.

User-item affinity and recommendations

Generally, for business purposes, customers are grouped based on various attributes. For example, X might be grouped by a bank as a middle age executive based on his age, or as a gold customer based on his high account balance. Y might be grouped as a young student based on her occupation and age. In the same way, products are also grouped based on the type or use. Like a laptop might be grouped as electronics based on the type, or a backpack might be grouped as college supplies based on its use.

Businesses try to build recommendation engines based on these groups.

> *A new age of recommendation engines have become very popular today and they are called collaborative filtering recommendation engines. Collaborative filtering works on just three pieces of data. A user or a customer, an item, and an affinity score between the user and the item.*

For example, in Amazon, the user is the buyer, the item is the product, and the affinity is whether the user purchase this item.

This affinity score would be a binary score whether the user bought the item or not.

In Netflix, the user is the viewer, the item is the video, and the affinity score is the

In Netflix, the user is the viewer, the item is the video, and the affinity score is the combination of ratings, minutes viewed, and the number of views for that user.

The goal of collaborative filtering is to group items based on users who bought them, or group users based on items they bought. Users who bought the same item or similar items bought by the same user or similar, the individual attributes of the users or the items does not matter.

So milk may end up being similar to soap. X might end up being similar to Y. The goal of this use case is to find products that are similar to each other based on the customers who bought them. Then when a new customer buys a product, we can recommend other items with strong affinity scores to this item, to the new customer. The data for this use case is going to be affinity data with the three elements. The user, item, and the score. We can get deep and get creative if you are to come up with the score.

It can a simple one or zero, or it can be a compound score on the scale of zero to ten. The algorithm used would be item based recommendation through collaborative filtering. The algorithm then creates a recommendation database where we provide an item as input and it comes up with a similar list of items. The call to action is to use the affinity score to build the recommendation database in the backing and keep it refreshed from time to time based on new data.

When a new website user comes in and use a specific product, we can query those database in real time and provide the user with the list of similar products like users who bought this product also bought. This use case is a very popular one, and it is used in popular websites like Amazon and Netflix. how to implement this use case in Python.

How implement Recommend items in Python

we are going to see an implementation of the use case for building recommendations. So when a customer comes to our website and has bought a product, we want to recommend him products that similar people bought. So in this exercise, we're going to use data about which customers bought which product and, based on that, build an item to item affinity score, and then use it for recommendation.

This is the data file.

	A	B	C
1	UserId	ItemId	
2	1001	5001	
3	1001	5002	
4	1001	5005	
5	1002	5003	
6	1002	5004	
7	1002	5005	
8	1003	5001	
9	1003	5002	
10	1004	5001	
11	1004	5004	
12	1004	5005	

13	1005	5002	
14	1005	5004	
15			
16			
17			

So data file is very simple, which user bought which item. So we have user IDs and item IDs. So users 1,001 seem to have bought 5,001, 5,002, and 5,005, and so on, pretty simple, straightforward data file. Moving on to the notebook, the notebook is the Recommendations notebook that we just talked about.

What we're going to do here is, first, we're going to load the file and take a look at it. Now we have to build an affinity score between items based on the users who bought them Now there are a number of out-of-the-box open source as well as commercial collaborative filtering libraries that can build this affinity score.

```
import pandas as pd

userItemData = pd.read_csv('ratings.csv')
userItemData.head()
```

We are going to take every item and find the affinity of that item to other items, and the way I'm going to do it is by finding out how many customers have bought both these products. The more number of customers who buy both the products, the higher is going to be the affinity score. So that is how the affinity score I am going to be computing here, and after that you can see that I am printing out the affinity score between each items. Item one to item two has a high affinity score of .4, whereas 5,001 to 5,003, there is no affinity at all.

```
#Get list of unique items
itemList=list(set(userItemData["ItemId"].tolist()))

#Get count of users
userCount=len(set(userItemData["ItemId"].tolist()))

#Create an empty data frame to store item affinity scores for items.
itemAffinity= pd.DataFrame(columns=('item1', 'item2', 'score'))
rowCount=0

#For each item in the list, compare with other items.
for ind1 in range(len(itemList)):

    #Get list of users who bought this item 1.
    item1Users = userItemData[userItemData.ItemId==itemList[ind1]]["userId"].tolist()
    #print("Item 1 ", item1Users)

    #Get item 2 - items that are not item 1 or those that are not analyzed already.
    for ind2 in range(ind1, len(itemList)):

        if ( ind1 == ind2):
            continue

        #Get list of users who bought item 2
        item2Users=userItemData[userItemData.ItemId==itemList[ind2]]["userId"].tolist()
        #print("Item 2",item2Users)
```

```
        #Find score. Find the common list of users and divide it by the total users.
        commonUsers= len(set(item1Users).intersection(set(item2Users)))
        score=commonUsers / userCount

        #Add a score for item 1, item 2
        itemAffinity.loc[rowCount] = [itemList[ind1],itemList[ind2],score]
        rowCount +=1
        #Add a score for item2, item 1. The same score would apply irrespective of the sequence.
        itemAffinity.loc[rowCount] = [itemList[ind2],itemList[ind1],score]
        rowCount +=1

#Check final result
itemAffinity.head()
```

	item1	item2	score
0	5001.0	5002.0	0.4
1	5002.0	5001.0	0.4
2	5001.0	5003.0	0.0
3	5003.0	5001.0	0.0
4	5001.0	5004.0	0.2

So this is the list that come up with the list of affinity scores. So how do we do recommendations? Let us say the customer bought a product 5,001. We want to find what items want to recommend to him. And for that, We are going to go back to this table, go to all the records that are item one in the first column, and get the list of all the items two and their scores. And we can do that in descending order. And those items that you see here is what I want to recommend.

```
searchItem=5001
recoList=itemAffinity[itemAffinity.item1==searchItem]\
        [["item2","score"]]\
        .sort_values("score", ascending=[0])

print("Recommendations for item 5001\n", recoList)
```

Recommendations for item 5001

	item2	score
0	5002.0	0.4
6	5005.0	0.4
4	5004.0	0.2
2	5003.0	0.0

So for 5,001, we see that 5,002 and 5,005 has a score of .4, of 5,004 has .2, and 5,003 has zero. So what we can do is have a threshold that we are going to only recommend those items whose score is above a certain point.

Let’s say if we have that threshold as .25, then we would recommend the products 5,002 and 5,005 to the customer