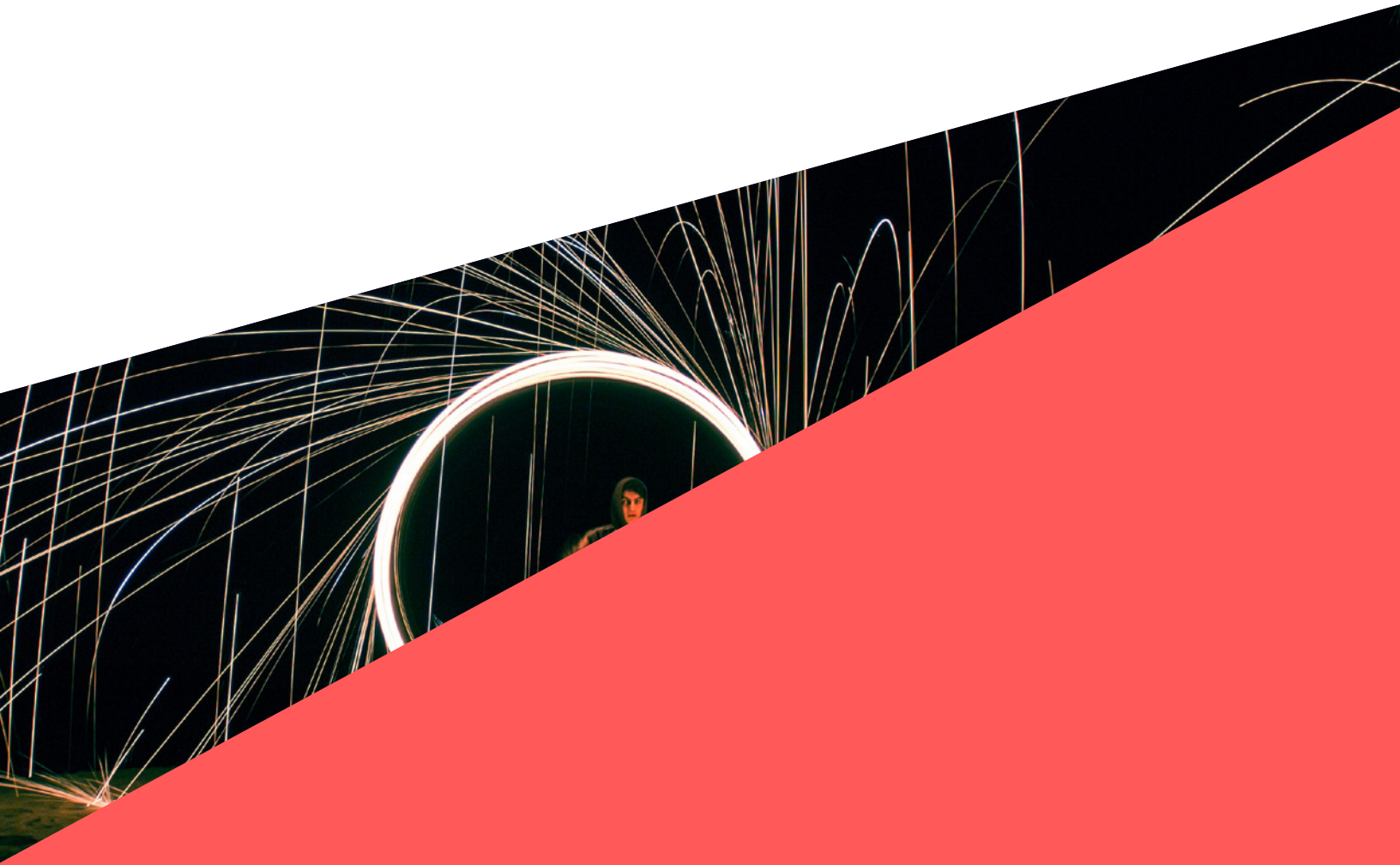
How To:
Drive Serendipitous Discovery Recommendation Engines

Successfully Predict Content Users Will Like (Even If They Don't Know They'll Like It)



A GUIDEBOOK BY DATAIKU

www.dataiku.com

TABLE OF CONTENTS

About the Guidebook

This guide is intended to provide a high-level overview of recommendation engines, how they're built, and how they can be used to improve businesses across industries, from media to e-commerce and more. By the end, readers should have an understanding of:

The breadth of different systems that use recommendation engines to improve user experience and the types of recommendation engines that exist.

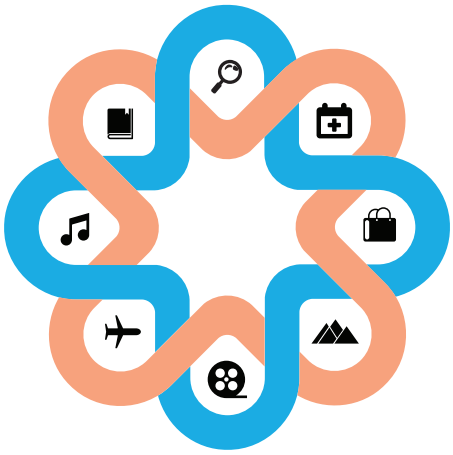
Business questions to evaluate before determining whether building a recommendation engine is the best solution, and if so, considerations before diving in.

How to build a simple recommendation engine as well as tips and guidance for building a more complex system.

For those completely unfamiliar with data science in the context of recommendation engines, this guide will provide a short introduction to the topic and walk through the core aspects.

But on top of that, for those already familiar, the guide includes some code and practical examples for execution.

What and Why?



Recommendation engines - or recommender systems - are built to predict what users might like, especially (though not always) when there are lots of choices available. Recommendation engines can explicitly offer those recommendations to users (e.g., Amazon or Netflix, the classic examples), or they might work behind the scenes to choose which content to surface without giving the user a choice.

Either way, the "why" is clear: they're critical for certain types of businesses because they can expose a user to content they may not have otherwise found or keep a user engaged for longer than they otherwise would have been (read: increase revenue from more sales or due to increased advertising/partners). While building a simple recommendation engine can be quite straightforward, the real challenge is to actually build one that works and where the business sees real uplift and value from its output.

Recommendation engines can be built using a variety of techniques, from simple (e.g., based only on other rated items from the same user) to extremely complex. Complex recommendation engines leverage a variety of different data sources (one challenge is using unstructured data, especially images, as the input) and machine learning (including deep learning) techniques. Recommendation engines are well suited for the world of artificial intelligence (AI) and unsupervised learning; as users continue to consume content and provide more data, these systems can be built to provide better and better recommendations.

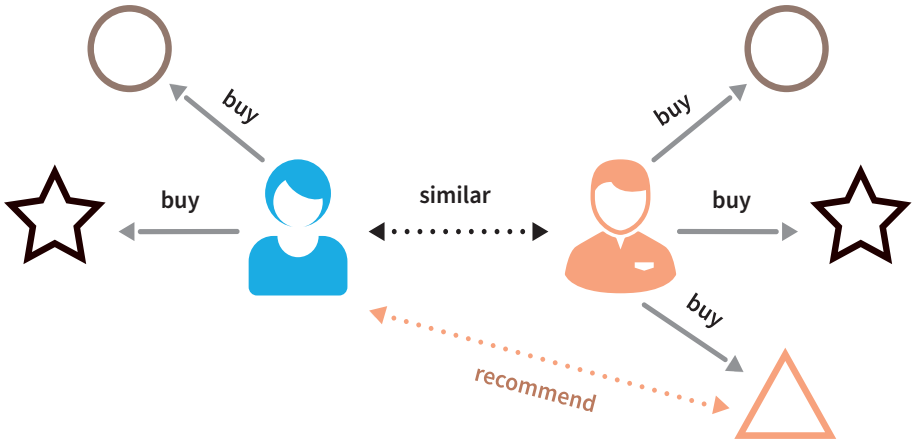
In general, very few recommendation engines are a core product - that is, it's extremely difficult to create a product that is, in and of itself, a recommendation engine. One reason is because recommendation engines require lots and lots of data to work well. Building a recommendation from the ground up as the core product suffers from the cold-start issue: a lack of data from the beginning.

Rather, recommendation engines are more commonly a layer on top of a core product that helps users discover additional items in your catalogue (whether those items are pieces of content or physical products). This guidebook will focus on recommendation engines in the context of this more common use case.

There are two primary types of recommendation engines, each with different sub-types. Depending on goals, audience, the platform, and what you're recommending, these different approaches can be employed individually, though generally the best results come from using them in combination:

Collaborative Filtering

It primarily makes recommendations based on inputs or actions from other people (rather than only the user for whom a recommendation is being made). Variations on this type of recommendation engine includes:



The Association Strategy

A specific type of User Similarity Strategy, otherwise known as "Users who looked at X also looked at Y."

Implementing this type of recommendation engine is a matter of looking at purchasing sequences, or purchasing groups, and showing similar content.

Useful for capturing recommendations related to naturally complementary content (e.g., recommending a tea infuser for someone who is purchasing loose tea) as well as at a certain point in the life of the user (e.g., after a purchase).

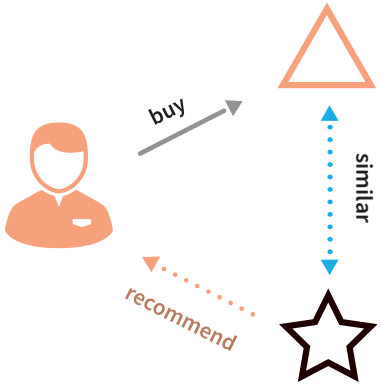
The User Similarity Strategy

This strategy involves creating user groups by comparing users' activities and providing recommendations that are popular among other members of the group.

This strategy is useful on sites with a strong but versatile audience to quickly provide recommendations for a user on which little information is available.

Content-Based

Content-based systems make recommendations based on the user's purchase or consumption history and generally becomes more accurate the more actions (inputs) the user takes. More specific types of content-based recommendation engines include:



The Content Similarity Strategy

The most basic type of content-based recommendation engine, this strategy involves recommending content that is close based on its metadata. This approach makes sense for catalogs with a lot of rich metadata and where traffic is low compared to the number of products in the catalog.

The Latent Factor Modeling Strategy

Going one step further than The Content Similarity Strategy, the crux of this strategy is inferring individuals' inherent interests by assuming that previous choices are indicative of certain tastes or hobbies. Where the previous strategy is based on explicit, manually filled catalog metadata, this strategy hinges on discovering implicit relationships.

This is done by using the history of users' larger interactions (e.g., movie watched, item purchased, etc.) to «learn» these tastes.

The Topic Modeling Strategy

This is a variant of the Latent Factor Modeling Strategy whereby instead of considering users' larger actions, one would infer interests by analyzing unstructured text to detect particular topics of interest.

Particularly interesting for use cases with rich but unstructured textual information (e.g., news articles).

There is also one final strategy that is neither collaborative filtering nor content-based but that can be useful for specific types of businesses or products/content:

The Popular Content Promotion Strategy

This involves highlighting product recommendations based on the product's intrinsic features that may make it interesting to a wide audience: price, feature, popularity, etc. This strategy can also take into account the freshness or age of the content and thus enable using the most «trendy» content for recommendations. This is often used in cases where new content is the majority.

How?

Building a successful and robust recommendation engine can be relatively straightforward if you're following the basic steps to growing from raw data to a prediction.[1] That being said, there are some particularities to consider when it comes to recommendation engines that often go overlooked and that, for the most efficient process and best predictions, are worth introducing (or reiterating).

This section will walk through the seven fundamental steps to completing a data project in the context of building a recommendation engine, covering the basics plus some more advanced topics in the shaded boxes.

1 Understand the Business

Extremely simple and critical but often overlooked, the first step in building a recommendation engine is defining the goals and parameters of the project. This will most definitely involve discussions between and input from both the data team as well as business teams (which might be product managers, operations teams, even partnership or advertising teams, depending on your product).

Here are some specific topics to consider to understand the business need more deeply and kickstart the discussion between these teams:

What Is the End Goal of the Project?

Is the idea to build a recommendation engine to directly increase sales overall?
Achieve a higher average basket size?
Reduce browsing time and make a purchase happen faster?
Reduce the long tail of unconsumed content? Increase user engagement time with your product?

It's important to be clear about the ultimate objective so that both business and data teams building the product have the same goals.

Is a Recommendation Rystem Really Necessary?

This is perhaps an obvious question, but since they can be expensive to build and maintain, it's worth asking.
Can the business achieve its end goal by driving discovery via a static set of content instead
(like staff/editor picks or most popular content)?

With these two most important questions out of the way and everyone aligned, you might dive into more specifics, such as:

At What Point Will Recommendations Occur?

If recommendations make sense in multiple places (i.e., on a home screen upon first visiting the app or site as well as after purchasing or consuming content), will the same system be used in both places, or are the parameters and needs distinct for each?

What Data Is Available on Which to Base Recommendations?

At the time of recommendation, approximately what percentage of users are logged in (in which case there may be much more data available) vs. anonymous (which could complicate things for building the recommendation engine)?

Are There Product Changes That Must Be Made First?

If the team wants to build the recommendation engine using more robust data, are there product changes that must be made first to identify users earlier (i.e., invite them to log in sooner), and if so, are they reasonable changes from a business perspective?

Should All Content or Products Be Treated Equally?

That is, are there particular products or pieces of content that the business team wants to (or has to) promote aside from organic recommendations?

How Can (or Should) Users With Similar Tastes Be Segmented?

In other words, if employing the User Similarity Strategy, how will you decide what makes users similar?

2 Get Your Data

The best recommendation engines use lots and lots of data. So when it comes to rounding up data to use for your recommendation engines, in general, the more the better. This can be difficult if users are unknown when you're trying to make a recommendation for them - i.e., they're not logged in or, even more challenging, they're brand new.

If you have a business where most users are unknown, you may need to rely on external data sources or general data not explicitly tied to preferences, like demographics, browsing history, etc.

When it comes to user preferences, there are two kinds of feedback: explicit and implicit.

Explicit user feedback is anything that requires user effort, like leaving a review/rating or initiating a complaint or product return (often from customer relationship management, CRM, data).

By contrast, implicit user feedback is information that can be gathered about a user's preferences without them actually specifying those preferences. For example, past purchase history, time spent looking at certain offers, products, or content, data from social networks, etc.

Good recommendation engines usually employ a combination of these types of feedback since there are advantages and disadvantages to each.

Explicit feedback can be very clear - a user has literally stated their preferences, likes, or dislikes. But by the same token, it's inherently biased; a user doesn't know what he doesn't know (in other words, he might like something but has never tried it and therefore wouldn't list it as a preferences or interact with that type of item or content normally).

By contrast, implicit feedback is the opposite - it can reveal preferences that a user didn't - or wouldn't - otherwise admit to in a profile (or perhaps their profile information is stale). On the other hand, implicit feedback can be more complicated to interpret; just because a user spent time on a given item doesn't mean that (s)he likes it, so it's best to rely on a combination of implicit signals to determine preference.

Go Further

Given all the different approaches, styles of recommendation engines, and types of businesses that might leverage these systems, it would be impossible to cover every type and use case combination. So throughout these Go Further sections, we'll walk through one particular example, a simple e-commerce website that wants to employ user-based collaborative filtering (The User Similarity Strategy).

In the code throughout this guidebook, we will always assume that we have one generic dataset to start with, a visit dataset with schema:

user_id | product_id | visit

We will work in SQL (Postgres dialect).

Bonus

If you're just getting started and want to try your hand building a recommendation engine using available datasets that mimic the type of data you'd probably actually have in the wild, we recommend taking a look at these datasets.[2]

3 Explore and Clean Data

One thing to consider when exploring and cleaning your data for a recommendation engine in particular is changing user tastes. Depending on what you're recommending, older reviews, actions, etc., may not be the most relevant on which to base a recommendation. Consider only looking at features that are more likely to represent the user's current tastes and removing older data that might no longer be relevant or adding a weight factor to give more importance to recent actions compared to older ones.

Go Further

When preparing data to use for a recommendation engine, the first thing to do is some normalization since you'll need it for any of the recommendation scores - normalization will scale every score between 0 and 1 so that it's possible to compare things against each other to understand what is a good recommendation and what's not. As the users and products often follow a long-tail distribution, we will also cut the long tail by filtering on users and products that occur frequently enough (you can see this part below.

```
WHERE b.nb_visit_product > 15 AND b.nb_visit_user > 5
```

The following SQL recipe will produce a table called **visit_normalized** containing:

```
product_id | user_id | nb_visit_user | nb_visit_product| visit_user_normed | visit_product_normed
```

```
SELECT
b.product_id,
b.user_id,
b.nb_visit_user,
b.nb_visit_product,
b.visit/SQRT(SUM(b.visit*b.visit) OVER (PARTITION BY b.user_id)) AS visit_user_normed,
b.visit/SQRT(SUM(b.visit*b.visit) OVER (PARTITION BY b.product_id)) AS visit_product_normed

FROM (
    SELECT *,
      COUNT(*) OVER(PARTITION BY user_id) as nb_visit_user,
      COUNT(*) OVER(PARTITION by product_id) as nb_visit_product
    FROM visit

  ) as b
WHERE b.nb_visit_product > 15 AND b.nb_visit_user > 5
ORDER BY product_id,user_id
```

4 Enrich Data

Datasets for recommendation engines can be challenging to work with because they are commonly high dimensional, but at the same time, it's also common that many of the features don't have any values, which can make clustering and outlier detection difficult.

For example, while this is not how data would usually be stored (more likely is a user | product | visit/buy matrix)you can imagine for illustration purposes how the following type of customer/product matrix for an e-commerce business (where 1=visit/buy, 0=no_visit/no_buy) can get large quickly based on the size of the user base and the amount of content or products available:

	Dress	Shoes	Jacket
Anne	1	1	0
Bob	0	1	1
Celia	1	0	0

Enriching this data involves comparing the products or the users (or both) to one another and determining the similarity between them so that we can then use that information as the basis of the recommendation engine.

Go Further

In this stage, we'll compute a score of similarity between two users. We first present the user-user similarity score, which shows how alike pairs of users are:

The **user_similarity** table will have the following schema: **user_1 | user_2 | similarity_user**

This is what the following query achieves:

```
SELECT c1.user_id AS user_1,
    c2.user_id AS user_2,
    SUM(c1.visit_user_normed*c2.visit_user_normed) AS similarity_user

FROM visit_normalised c1
INNER JOIN visit_normalised c2
ON c1.product_id=c2.product_id

GROUP BY user_1, user_2
ORDER BY user_1, similarity_user DESC
```

As there are usually too many pairs of users to score, so we often restrict this query (cf. the condition **WHERE b.rank <= 10**) by limiting ourselves to the 10 highest user similarity scores per user. The table has then schema **user_1 | user_2 | similarity_user | rank** and our query becomes:

```
SELECT b.*

  FROM(

  SELECT a.*,
    row_number() over(partition by a.user_1 order by a.similarity_user desc) as rank

     FROM user_similarity as a

  WHERE a.user_1 != a.user_2) as b

WHERE b.rank <= 10
```

We can then use this user similarity to score a product for a given user, user_j :

$$score(user\_j, new\_product) = \sum sim(user\_j, user\_i)\delta(user\_i, new\_product)$$

where
δ**(user_i, new_product)**
is equal to 1 if the user i has seen the new product and 0 otherwise

More generally, we could change δ**(user_i, new_product)** by:
The number of times **user_i** has seen the new product.
The time spent by **user_i** on the new product.
How long ago the user visited this product.

This is done using the following SQL script where the score_cf_user table will have the following schema:
user_id | product_id | score_user_product

```
SELECT a.user_1 as user_id,
    b.product_id,
    SUM(a.similarity_user) as score_user_product

FROM user_similarity as a
INNER JOIN visit_normalised as b
ON a.user_2 = b.user_id
GROUP BY user_id,product_id
ORDER BY user_id,score_user_product DESC
```

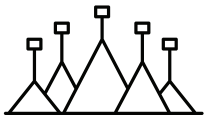
This particular example calculates a user similarity score, but it's also possible to do the same for item-based collaborative filtering or for a content-based system. If calculating more than one score, you will then also need to consider in the end which score you'll use for the final results. Instead of choosing just one, the best solution might be to average the scores to compute a final affinity for each user/product, achieving an optimal combination between these scores.

5 Get Predictive

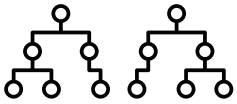
Given the work done in the previous section (comparing the products, users, or both to one another and determining the similarity between them), you could already build a recommendation engine. Just rank those scores by users and you'll have product to recommend.

This strategy doesn't use machine learning or a predictive element, but that's ok! For some use cases, this is sufficient.

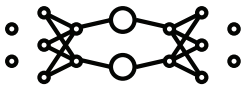
But if you do want to build something more complex, there are lots of subtasks that can be done after users consume recommended content that can be used to further refine the system. There are several ways to leverage the hybrid approach to try for the highest-quality recommendations:



Presenting recommendations from different types of systems together side-by-side.



Maintaining multiple algorithms in parallel where the decision of which algorithm is preferred over another is itself subject to machine learning (e.g., multi-armed bandit).



Using a pure machine learning approach to combine multiple recommendation systems (logistic regression or other weighted regression methods). One specific example would be using a weighted average of two (or more) recommendations using different techniques.

It's also possible that different models will work better in different parts of the product or website. For example, the homepage where the user has yet to take action vs. after the user has clicked or consumed content in some way.

Go Further

The major difficulty in predicting if a user will visit/buy/like a given product is that we only have positive examples from which to learn. Indeed, we always have the visits, buys, and likes, but in many cases, we don't have the counterfactual events (i.e., didn't visit, didn't buy, didn't like). This means that we will have to create those using what we call «negative sampling."

Imagine that you have your visit/buy user data until a given date T (which is probably today or yesterday). You will compute your several affinity scores by taking into account the data since a date T - x days, where x is variable you chose.

Then, you will be able to create a learning set with the data from T - x days until T. During this period of time, you have a set of User/Product couples that are «true» (i.e., the user bought the product during the period). But you don't have negative examples, so you will have to create User/Product couples that are «false» (i.e., the user didn't buy the product during the period). For each couple, your features will be the affinity scores you just computed, and the target will be «true» or «false.» You try to find the best combination to optimise the visits, buys, and likes.

How many «false» couples do I have to create? Well, it's a good question, and there is no real answer to this. It should be enough to have an unbalanced dataset, since in reality, the events visit, buy, or like are actually very unlikely to happen.

How can I create «false» couples ? There are several strategies:

A *Randomly (not a very good one)*

B *Randomly, but only picking users that visited, bought, or liked at least one thing during the period. We don't pick users with only negative examples during that period because otherwise it would be too easy to have a good score, and it would not be very relevant.*

C *Excluding couples that happened in the past (if someone already bought something you don't want to penalize it). We don't take as fake example a true example from the past.*

Now that your learning set is ready, you just have to split into a train and test set to create and evaluate your model. For the best possible results, you should do this split on a time basis.

Let's say now that we have new columns date in our visit dataset:

```
user_id | product_id | visit | date
```

Here is a SQL code to to create a learning set as described above with strategy number three:

```
Select f*, 0 as target

from
  (
  Select user_id, product_id
  from
    (Select distinct user_id
      from visit
      where event_date > 'Your_Date_T - x days' and event_date < 'Your_Date_T')u,
    (Select distinct product_id from visit)p
  where mod(user_id+product_id,20)  -- Here it's a trick to randomly keep at most 1/20 of the "fake" created couples. You have to tune this so that your target is not too unbalanced. The less "real" visits you have, the more you will have to reduce the number of "fake" couples.
  )f

left join

(Select distinct user_id,product_id,1 as real_
  from visit
  where event_date > 'Your_Date_T - x days' and event_date < 'Your_Date_T')r

on f.user_id=r.user_id and f.product_id=r.product_id

where f.real_ is Null

UNION ALL

Select distinct user_id,product_id,1 as target
from visit
where event_date > 'Your_Date_T - x days' and event_date < 'Your_Date_T'
```

With this approach, once you trained a model based on your affinity scores, you will have to score all possible user/product couples. It can be extremely expensive if you have a huge catalog of products. That's why this approach better suits limited catalogs of products or content. We can think about ways to tackle this issue, but it will always mean limiting the set of available products. This limitation can be drawn by marketing rules for example.

We also see that the choice of the algorithm will be very important, because the scoring time will be completely different depending on this choice. A logistic regression could be a good choice in that case since it's a linear combination which allows a really fast scoring.

6 Visualization

Visualization in the context of recommendation engines serve two primary purposes:

1. When still in the exploration phases, visualizations can help reveal things about the data set or give feedback on model performance that would otherwise be difficult to see.

2. After putting the recommendation engine in place, visualizations can help convey useful information to business or product teams (e.g., which content does well but isn't being discovered, similarities between users' tastes, content or products commonly consumed together, etc.) so they can make changes or decisions based on this information.

The primary issue with visualizing this type of data is the amount of data present, which can make it difficult to cut through the noise in a meaningful way. But by the same token, a good visualization will help make sense out of lots of data from which it would be otherwise difficult to derive meaningful insights.

7 Iterate and Deploy

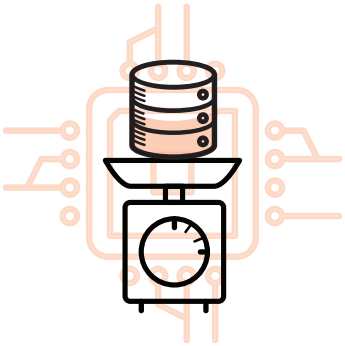
Recommendation engines that are working in a dev environment or sandbox don't do any good. It's all about putting the system into production so that you can begin to see the effect on the business goals you've laid out in Step 1, Understand the Business.

Additionally, keep in mind that the more data you have with which to feed the recommendation system, the better it can become. So with this type of data project perhaps more so than others, it's critical to evaluate performance and continue to fine-tune, like adding new data sources to see if they have a positive effect.

In fact, making sure your recommendation engine is built to adapt and evolve by regularly monitoring its performance is one of the most important parts of the process - a recommendation engine that isn't properly adjusting to tastes or new data over time likely will not help you ultimately achieve your initial project goal, even if the system performed well at first. Building a feedback loop to understand whether or not users care about recommendations will be helpful and provide a good metric for making refinements and decisions going forward.

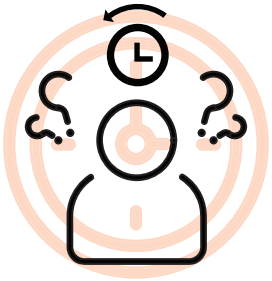
If recommendations are core to your business, constantly trying new things and evolving the initial model you've created will be an ongoing task; recommendation engines are not something you can create and cast aside.

What Are the Pitfalls?



SCALABILITY

It's important to create a recommendation engine that will scale with the amount of data you have. If it's built for a limited dataset and that dataset grows, computation costs grow exponentially, and the system will be unable to handle the amount of data. To avoid having to rebuild your recommendation engine later on, ensure from the beginning it is built to scale to expected data volumes.



NO SURPRISES

It's possible that after spending time, energy, and resources on building a recommendation engine (and even after having enough data and good initial results) that the recommendation engine only makes very obvious recommendations. The crux of avoiding this pitfall really harkens back to the first of the seven steps: understand the business need. If there isn't enough of a content long tail or no need for the system, perhaps reconsider the need to build a recommendation engine in the first place.

AGILITY

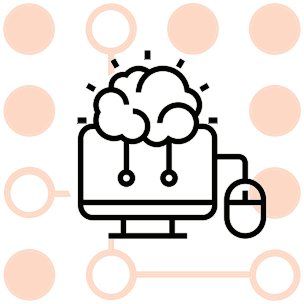
People's tastes don't stay static over time, and if a recommendation engine isn't built to consider this fact, it may never be as accurate as it could be. Similarly, there is a risk of building a recommendation engine that doesn't get better over time. As users continue to consume content and more data is available, your recommendation engine should learn more about users and adapt to their tastes. A recommendation engine not agile enough to continue to adapt can quickly become obsolete and won't serve its purpose.

Looking Forward

Basic recommendation engines have been around for quite some time, though they continue to get more complex and have been perfected by retail and content giants. But what's next? What are the latest trends and developments that businesses should consider if they are looking to develop a truly cutting-edge system?

CONTEXT-AWARE

Context-aware recommendation engines represent an emerging area of experimentation and research, aiming to provide even more precise content given the context of the user in a particular moment in time. For example, is the user at home, or on the go? Using a larger or smaller screen? Is it morning or night? Given the data available on a certain user, context-aware systems may be able to provide recommendations a user is more likely to take in those scenarios.



DEEP LEARNING

Deep learning is already in use by some of the biggest and most powerful recommendation engines in the world (like YouTube). But as the amount of data continues to skyrocket and more businesses find themselves up against a huge corpus of content and struggling to scale, deep learning will become the de facto methodology for not only recommendation systems, but all learning problems.

SOLVING THE COLD-START PROBLEM

Solving the cold-start problem is also something that cutting-edge researchers are starting to look at so that recommendations can be made for items on which there is little data. This is a critically important area for businesses with lots of turnover in content to examine so that they can successfully push items that will sell well (even before they know how that item will perform).

As new research and developments in the field come about, we will continue to update this guidebook; or stay up-to-date on the latest on our blog.

Conclusion

Recommendation engines can be an effective way to expose users to content they may not have otherwise found, which in turn can forward larger business goals like increasing sales, advertising revenues, or user engagement. But there are a few key points to find success with recommendation engines. Namely, recommendation engines should be, above all, necessary.

Building a complex system that requires experienced staff and ongoing maintenance when a simpler solution will do is a waste of data team resources that could be spent elsewhere for more impact. The challenge lies in building a system that will actually have business impact; building the system in and of itself shouldn't be the end goal.

Recommendation engines should also be agile. That is, adaptable and able to evolve as users do. Putting a recommendation engine into production isn't the final step in the process; rather, it's an ongoing evolution, looking at what works, what doesn't, thinking about additional data sources that might help make better recommendations, etc.

References

[1] https://blog.dataiku.com/2016/07/06/fundamental-steps-data-project-success

[2] https://gab41.lab41.org/the-nine-must-have-datasets-for-investigating-recommender-systems-ce9421bf981c

[3] https://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/45530.pdf

[4] https://kddfashion2016.mybluemix.net/kddfashion_finalSubmissions/Fashion%20DNA%20Merging%20Content%20and%20Sales%20Data%20for%20Recommendation%20and%20Article%20Mapping.pdf

data iku

Your Path to **Enterprise AI**

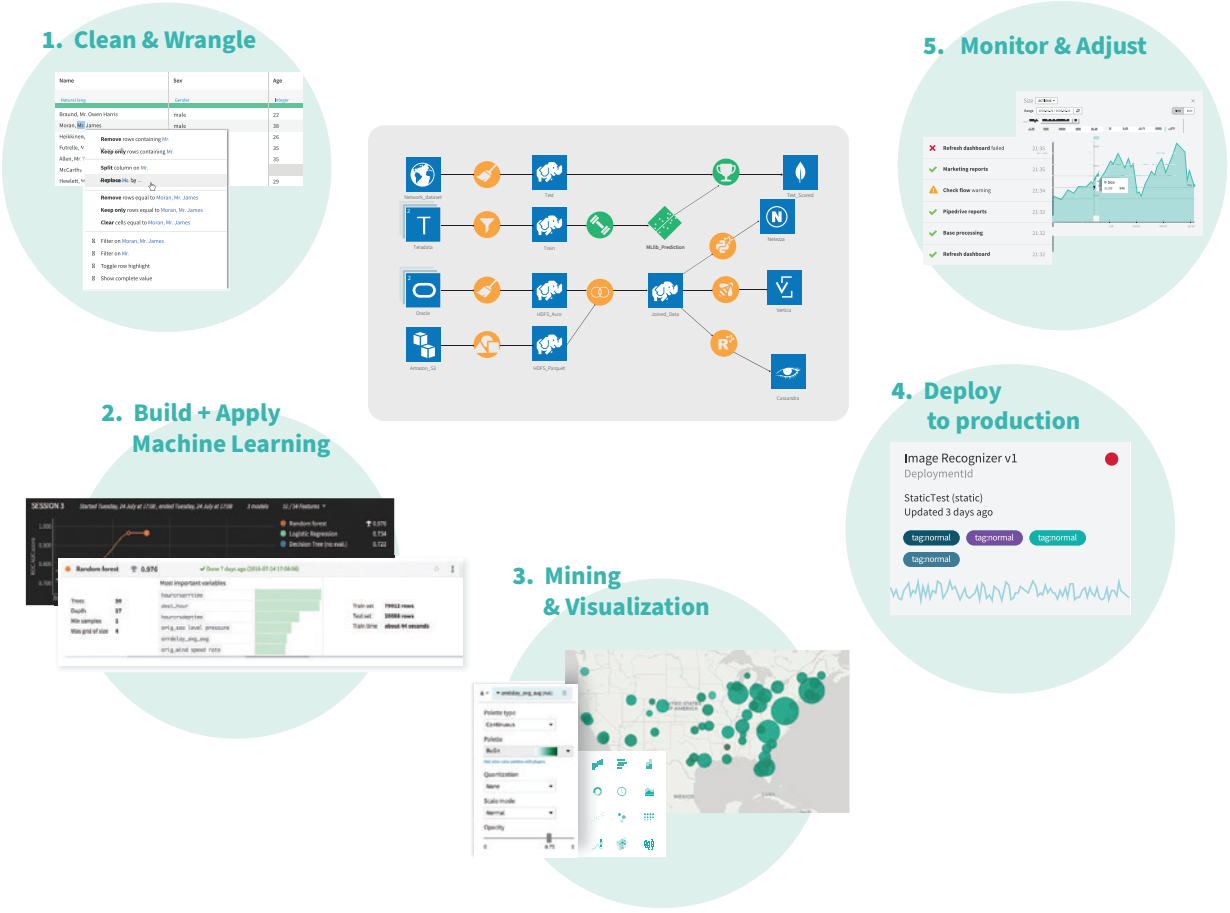
Dataiku is the platform democratizing access to data and enabling enterprises to build their own path to AI. To make this vision of Enterprise AI a reality, Dataiku is the only platform on the market that provides one simple UI for data wrangling, mining, visualization, machine learning, and deployment based on a collaborative and team-based user interface accessible to anyone on a data team, from data scientist to beginner analyst.

300+
CUSTOMERS

30,000+
ACTIVE USERS

*data scientists, analysts, engineers, & more

GE · Santander · FOX NETWORKS GROUP · Unilever · KUKA · BNP PARIBAS · paloalto NETWORKS · PREMERA BLUE CROSS · NXP · SEPHORA



GUIDEBOOK

www.dataiku.com